

**Computer-Aided Molecular Design with
Multi-Dimensional Characterization**

by

Robert Hartwell Herring III

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
12/13/2014

Keywords: molecular design, informatics, descriptors

Copyright 2014 by Robert Hartwell Herring III

Approved by

Mario R. Eden, Chair, Professor of Chemical Engineering
Maria L. Auad, Associate Professor of Polymer and Fiber Engineering
Allan E. David, Assistant Professor of Chemical Engineering
Christopher B. Roberts, Dean, Professor of Chemical Engineering

Abstract

In this work, a methodology for the solution of computer-aided molecular design (CAMD) problems with property models utilizing descriptors of varying dimensionality has been presented. The problems encountered within this field typically require the selection, or design, of pure chemicals, as well as mixtures, exhibiting a desired set of properties and attributes. These properties and attributes are captured through property models, which have widely varying forms. These property models are most often a function of molecular descriptors, which provide a quantitative reference to the structural features in a molecule. There are multitudes of descriptor types, each which can be immediately categorized based on the dimensionality of information they capture. This is one of the strengths of computer aided molecular design, the flexibility to develop a specific model for each property of interest. However, it often leads to the selection of very complex and widely differing property models for each property of interest. An ideal CAMD methodology would not restrict the property modelling stage to certain types of independent variables, and as such, could solve these problems on a single platform. The problem with developing such an algorithm is that the descriptors chosen are often of varying dimensionalities. Inclusion of descriptors beyond two-dimensional requires some consideration of the potential energy surface, or conformational space, for each candidate solution. In addition, the region in which to search for solutions becomes difficult to

identify because each property model has its own applicability domain, within which predictions can be made with increased confidence.

The approach presented within this dissertation, aimed at solving such problems, utilizes a fragment based descriptor known as the signature descriptor. Previous applications using this descriptor were shown to be successful in terms of solving the problem in an efficient manner while identifying novel solutions. Extension of this descriptor to include spatial information, along with the techniques necessary for using this data, is presented. This has allowed for the estimation of likely local energy minima without the conventional conformational analysis for each potential solution, which has been shown to be computationally intensive. The nature of signature descriptors, being fragment based, allows for an efficient description of which region in chemical space to search for solutions and also facilitates reconstruction of solutions matching a set of descriptor values. A description of previous approaches taken to solve problems of this nature has been outlined such that the benefits of the proposed technique could be exemplified. In addition, several studies have been provided to verify the proposed methodology.

Acknowledgements

Foremost, my success in the graduate program at Auburn University would not have been possible without the support and guidance provided by my advisor Dr. Mario Eden as well as the former department head, Dr. Chris Roberts. They were extremely helpful to me throughout my stay at Auburn and had unquestioning faith in my ability to perform as a graduate student. Mario accepted me into his group during a transitional phase for me and has provided an environment in which I was able to thrive as a researcher and develop as a professional. He also provided the opportunity for me to travel the world presenting my work and was there for support every step of the way. I am forever grateful for his guidance as an advisor and the friendship that has developed alongside it.

I would also like to acknowledge the support provided by my committee members, Dr. Maria Auad and Dr. Allan David. Their feedback and support during the development of my dissertation was invaluable and allowed me to maximize the quality of my work. I was fortunate to have an incredible group of students within my research group from which I could bounce ideas off of and also provide an occasional distraction from work when it was called for. This includes Dr. Subin Hada, Dr. Susilpa Bommareddy, Dr. Nishanth Chemmangattuvalappil, Dr. Wei Yuan, Dr. Charles C. Solvason, Dr. Zheng Liu, Mr. Colin Haser, Mr. Narendra Sadhwani, Mr. Vikrant Dev, Mrs. Sarah Davis, and Mr. Shounak Datta. I have a special appreciation for one of the closest friendships I have

developed in my lifetime with Dr. Subin Hada. We came from a small college in Mobile, AL and were destined to conduct research together here at Auburn.

Finally, I have to acknowledge the support provided to me by my family. My mother and father, Vicki Herring and Robert Herring Jr., have always been there for me along with my two sisters, Juli and Shelly, and step father, Bo Olsen. When the stresses of graduate life became too heavy, they were there for me and without them I could not have succeeded. Last but not least, my girlfriend, Lyssa Youngblood, was supportive and understanding of my long hours and late nights spent meeting the various challenges I encountered throughout my research.

Table of Contents

Abstract	ii
Acknowledgements	i
List of Figures	i
List of Tables	i
1. Introduction	2
1.1 Challenges and Motivation	3
1.2 Scope and Objectives	5
1.3 Significance of Research	7
1.4 Organization	9
2. Background	13
2.1 Computer-Aided Molecular Design Approach	14
2.2 Molecular Descriptors	18
2.2.1 0D-1D Descriptors	20
2.2.2 2D Descriptors	21
2.2.3 3D Descriptors	25
2.3 Molecular Modeling	35

2.3.1 Molecular Mechanics	36
2.3.2 Quantum Chemical Methods.....	38
2.3.3 Geometry Optimization	41
2.4 QSAR/QSPR	44
2.4.1 Variable Selection Techniques.....	45
2.4.2 Mapping Descriptors into Attribute Space	50
2.5 Fragment Based Property Models.....	58
2.5.1 Group Contribution Method.....	60
2.5.2 Pharmacophore Models.....	63
2.5.3 Signature Descriptor.....	64
2.6 Solution Techniques in CAMD.....	70
2.6.1 Database Search.....	70
2.6.2 Generate and Test.....	75
2.6.3 Programming and Optimization	78
3. Methodology	84
3.1 Deterministic Solution Approach.....	87
3.1.1 Identification of Property Models	88
3.1.2 Data Set Selection.....	89
3.1.3 Conformational Analysis.....	90
3.1.4 Spatial Signature Development	91

3.1.5	Compression of Spatial Information	91
3.1.6	Creation of Bonding Network	94
3.1.7	Generation of Structural Isomers.....	95
3.1.8	Generation of Conformational Isomers	98
3.1.9	Extension to More Complex Structures	104
3.2	Stochastic Solution Approach	107
3.2.1	Overall Genetic Algorithm Methodology	110
3.2.2	Generation of Starting Population	113
3.2.3	Fitness Calculation	115
3.2.4	Genetic Operators.....	117
4.	Case Studies	122
4.1	Solvent Design Study	123
4.1.1	Conclusions	127
4.2	Design of Alkyl Substituent for Rice Plant Fungicide.....	128
4.2.1	Conclusions	132
4.3	Geometry Estimation Technique Analysis.....	133
4.3.1	Analysis of Methodology in Organic Space.....	134
4.4	Structure Based Design of Non-Peptide Mimetics	139
4.4.1	Pharmacophore and Non-Peptide Mimetics.....	139
4.4.2	Model Information.....	140

4.4.3 Atomic Signature Development	141
4.4.4 Combinatorial Optimization.....	143
4.4.5 Conclusions	145
4.5 Solvent Design with Genetic Algorithm	145
4.5.1 Development of Spatial Signatures	147
4.5.2 Parameters Utilized	148
4.5.3 Results and Conclusions.....	149
5. Conclusions and Future Direction	153
5.1 Improved Simulation Techniques	155
5.2 Consideration of Proteins.....	157
5.3 Simulated Annealing.....	160
References.....	162
Appendix A – Python Code for Proposed Methodology	173
A.1 – Creation of Spatial Atomic Signatures from Directory.....	173
A.2 – Network Generation	179
A.3 – Molecular Signature Class with Feasibility Functions.....	183
A.4 – Genetic Algorithm.....	194
A.5 – Code for Expedited Molecular Mechanics Analysis.....	224
A.6 – Geometry Verification Code	226
Appendix B – Solutions to Pharmacophore Case Study.....	240

List of Figures

Figure 2.1 - Example of constitutional descriptor correlation.	21
Figure 2.2 - Example calculations of Wiener index.	23
Figure 2.3 - Overview of geometrical descriptors.	27
Figure 2.4 - Outline of CoMFA approach.	33
Figure 2.5 - Visualization of SAR approach.....	45
Figure 2.6 - Mean centering and scaling in PCA.....	53
Figure 2.7 - Dimensionality reduction achieved with PCA.....	54
Figure 2.8 - PLS regression visualization.	55
Figure 2.9 - Classification through the use of a decision tree.....	57
Figure 2.10 - Overview of inverse property prediction.	59
Figure 2.11 – Examples of first and second order groups	61
Figure 2.12 - Examples of a pharmacophore model.....	64
Figure 2.13 - Example of atomic signature descriptors of varying height.....	65
Figure 2.14 - Development of atomic signature up to height 3.	67
Figure 2.15 - Example of height three molecular signature enumeration.....	68
Figure 2.16 - Results of 2D and 3D similarity searches.	75
Figure 2.17 - Linkage of CAMD and molecular modeling.	78
Figure 2.18 - Flowchart for inverse QSPR workflow optimization approach.....	80
Figure 3.1 - Overview of methodology developed.....	87
Figure 3.2 – Cutoff criterion example.....	92
Figure 3.3- Example of compatible ‘bonding network’ edge.....	94
Figure 3.4- Example of consistency equation for nitroglycerine.....	96
Figure 3.5 – Generation of structural isomers.	97
Figure 3.6 – Example of fragment selection for geometry development.	100
Figure 3.7– Utilization of developed conformational isomers.	103

Figure 3.8 – Common Genetic Algorithm Methodology.....	109
Figure 3.9 – Proposed Genetic Algorithm Utilizing Spatial Signature Descriptors	111
Figure 3.10 – Atomic signatures for nodes in molecular graph.....	112
Figure 3.11 – Effect of α on Fitness Distribution	116
Figure 3.12 – Effect of Crossover on Size Distribution.....	118
Figure 3.13– Selection of Type of Mutation Operator	119
Figure 3.14 – Implementation of Crossover Operator	121
Figure 4.1 - Fungicide structure.....	128
Figure 4.2 – Geometry verification data set.....	135
Figure 4.3 – Geometry verification test set.....	135
Figure 4.4- Example potential energy diagrams for (A) butane and (B) butene	137
Figure 4.5 - Conformers after compression.	138
Figure 4.6 - Pharmacophore Model for 5-HT6.....	141
Figure 4.7- Example pharmacophore groups.....	143
Figure 4.8 - Solutions to antagonist design case study.	144
Figure 4.9 – Fitness as a Function of Generation	150
Figure 5.1 - Examples of Common Ionic Liquid Cations and Anions	157

List of Tables

Table 2.1 - Definitions of Tanimoto Coefficient and Euclidean Distance.	73
Table 3.1 – Cutoff criterion compression example.....	93
Table 4.1 - Property constraints for solvent design study.....	123
Table 4.2 - Property models for solvent design study.	123
Table 4.3 - All height-2 atomic signatures for linear alkanes.....	125
Table 4.4 - Solutions for solvent design study.....	127
Table 4.5 - Fungicide substituent property constraints.....	129
Table 4.6 - Fungicide study property models.	129
Table 4.7 - Molecular signature solutions to fungicide problem.....	131
Table 4.8 - Substituent solution isomers.....	132
Table 4.9 - Conformers identified with MC search.	136
Table 4.10 – Common Solutions Identified During Solvent Design Case Study.....	150

1. Introduction

The field of computer-aided molecular design (CAMD) has seen an exponential increase in the complexity of problems considered, which has been made possible through a paralleled increase in algorithms and hardware available to solve these problems. This area has such humble origins and has expanded through continued contributions from many fields including medicinal chemistry, computer sciences, computational chemistry, bioinformatics and chemical engineering. Problems ranging from the identification of optimal solid state catalysts to the synthesis of potent HIV-1 protease inhibitors have benefited from CAMD techniques and studies with ever-increasing complexity are being considered.

The ability to describe molecular structures accurately and uniquely is possible through the utilization of molecular descriptors. These descriptors can capture a variety of aspects such as charge distribution, globularity, size, and complexity of structures under consideration. These are the details necessary for correlation to the various properties and activities of interest, which is done through the generation of mappings between descriptor space and attribute space. What descriptors offer is the ability to translate the characteristics of a molecular structure into a numerical domain. With this information, one is able to apply the established techniques of numerical analysis towards solving CAMD problems. However, one limitation to these increasingly informative models is the ability to use them in an efficient manner. The original property models typically considered very simple descriptors, such as molecular weights or atom counts. This was sufficient for correlation to common physico-chemical properties, however, to

capture the variance of more complex properties, it became necessary to use more complex descriptors. This increase in complexity demands improved CAMD solution techniques. Some techniques benefited from a unique description of the problem such that various programming and optimization algorithms could be applied to scan larger chemical search spaces. Others relied on efficient handling and consideration of data for solving these more complex problems. However, the most successful techniques have taken advantage of both approaches.

1.1 Challenges and Motivation

The improved ability to accurately model molecular structures and their interactions with each other has provided a wealth of information which was previously unattainable. This new information is now being produced with incredible accuracy such that its correlation with experimental properties is resulting in models which can be successfully interpolated and extrapolated around the original data set. This provides an opportunity to identify chemical solutions, with a set of desired properties, which have not been included in the original data set. The search space, known as chemical space, for these potential solutions is vast as the number of just small organic structures has been estimated to be around 10^{60} (Kirkpatrick and Ellis, 2004). Chemical space represents all possible collections and arrangements of atoms producing unique molecular structures. Such a large search area demands efficient methods of scanning for solutions. In addition, the varying applicability of these models, or the confidence associated with a models prediction for a specific region of chemical space, must be taken into consideration to produce reliable results.

These improved molecular modelling techniques, in addition to providing increasingly accurate information, are also providing much more complex information. For example, the spatial characteristics, or three-dimensional information, of molecules under study are now accessible through simulation. Such information has necessitated the creation of new independent variables capable of systematically representing this data across a variety of molecules. The ability to utilize these models in a CAMD approach requires some technique for estimating these values in an efficient manner such that a larger region of chemical space can be considered for improved solutions. While it is possible to thoroughly examine the spatial characteristics of each structure in a given data set, such that a more accurate and predictive model can be produced, the benefits of being able to quickly estimate this information during the solution process are obvious. In addition, the conformational characteristics of a given molecule are often incredibly complex, having multiple accessible conformers. This must be taken into consideration during the solution process as well.

Alongside improved molecular modelling techniques, which have introduced new ways to characterize molecules, improvements in model development techniques have followed. Specifically, variable selection techniques, which identify the optimal set of independent variables useful in characterizing an attribute of interest, have become very important. With the introduction of spatial molecular descriptors, a multitude of topological and topochemical descriptors with increased complexity have been designed as well. Because of this, there are thousands of molecular descriptors available today, of which any combination could produce the optimal set of independent variables useful for characterizing a chosen attribute. With these improved variable selection techniques we

are finding that the optimal set of descriptors is often varying in the dimensionality of information captured. This means that we often have topological, or two dimensional, and topographic, or three dimensional, indices in one equation. This creates a challenge for designing a CAMD approach capable of handling these models efficiently. One can imagine that each type of property or attribute will have its own unique combination of molecular descriptors. The ability to handle multiple models at once, each with a widely varying set of information, creates a powerful CAMD technique as a more globally optimal solution can be identified when compared to considering only one property at a time. In addition, solving these equations in a single pass would allow for a more efficient search method than one that is iterative in nature.

1.2 Scope and Objectives

The requirements placed on a CAMD approach capable of handling the latest techniques in molecular modelling and model development include:

- 1) The ability to quickly estimate spatial capabilities of a molecule under consideration without extensive simulation efforts would allow for consideration of a larger region of chemical space.
- 2) The decision of which region of chemical space to scan, such that each potential solution would fall under or within the applicability domain of each property model utilized, must be made.
- 3) The ability to simultaneously consider descriptors of widely varying nature, ranging from topological to topographic, is a necessity.

Within this dissertation, a methodology for the solution of CAMD problems with multi-dimensional characterization is proposed. This technique utilizes a fragment based descriptor, known as a spatial atomic signature, to build solution structures with targeted properties and attributes. Previous approaches for solving problems of this nature were limited to database searches and inefficient generate and test techniques which relied on time consuming analysis of the potential energy surface of each potential solution. This technique will allow for the consideration of a much larger region of chemical space, with minimal compromise on the accuracy of estimation made for higher dimensional information. The approach applies a graph based representation of molecular structures in which fragments are utilized to build potential solutions, which allows for a very efficient representation and consideration of a defined region in chemical space along with its associated conformational capabilities.

The spatial information used during the CAMD algorithm is initially generated through an extensive conformational analysis of a given data set. Much of this information is overlapping, or over-defined, and a compression algorithm has been introduced to minimize the potential for combinatorial explosion. Combinatorial explosion is an effect seen quite often in the solution of combinatorial optimization problems of this nature. These extensive molecular modelling efforts are done up front as opposed to within the CAMD approach, which saves time and allows for a larger region of chemical space to be searched.

In addition, the canonical fragment based representation of molecular structures through the atomic signature convention allows for an efficient search through a defined region of chemical space. A canonical representation ensures that a given molecular

fragment can only be represented in one way, which ensures that potential solutions are only visited once and minimizes the computational load of a given CAMD problem. Also, since the fragments are developed from an overlapping data set, which consider structures familiar to all property models utilized, the likelihood of a solution falling within the applicability domain of all models is significantly increased.

It is the overall goal of this research to be able to solve CAMD problems utilizing multiple property models, each with descriptors of varying complexity. Specifically, the inclusion of spatial descriptors is focused upon and the efficient spatial characterization of molecules considered is established. The approach is based on a canonical fragment based descriptor to allow for a combinatorial optimization based approach which minimizes the potential for combinatorial explosion so larger regions of chemical space can be considered. The approach is designed so that a deterministic or stochastic search for chemical solutions can be applied for searching specific regions of chemical space with varying degrees of completeness and speed.

1.3 Significance of Research

Previously, the solution of problems of this nature was done in an iterative manner where the property models were tackled with increasing complexity. This is an inefficient technique for solving CAMD problems with multiple properties of interest. The presented methodology is able to efficiently consider all of these models at once, on a single platform, such that the problem can be solved in a single pass regardless of the number and complexity of molecular descriptors utilized. The space in which these problems are tackled also becomes very well defined with the nature of the proposed techniques. This space is defined in an efficient manner, such as to avoid the combinatorial explosion

often associated with previous attempts at searching a comparable region of chemical space.

In addition, the consideration of spatial characteristics within the chosen search space is often lacking. The typical approach was to identify a single conformer, which was identified as being a potential local conformational isomer. This information was then utilized in establishing the molecule's spatial characteristics and ultimately its likelihood of exhibiting the properties of interest. It has been established that the actual conformational status of a given molecule is best represented by a collection of conformers, each with a varying likelihood of existing in solutions. This likelihood was based on the internal, or conformational, energy associated with this conformer. As such, the most realistic representation of the spatial characteristics for a given molecule would consider several potential conformational isomers. This methodology allows for the consideration of multiple conformers, which have been shown to be representative of the given local energy minima. This is done in an efficient manner, such that thorough molecular modelling techniques are not necessary.

With the consideration of increasingly complex problems, a CAMD approach capable of handling this information has become necessary. The ability to model interactions between groups of molecules such as ligands and receptors has allowed for the development of interesting spatial descriptions of these phenomena. Also, the traditional descriptors, which are efficient and effective at characterizing simple physico-chemical properties, still have a place in the arena of CAMD. This methodology can handle previously established ideologies as well as state of the art descriptions of molecules, thus allowing for the solution of a wide range of CAMD problems.

1.4 Organization

The format of this dissertation is such that the background is first introduced to provide the reader with enough knowledge to skeptically interpret the proposed methodology. This background, in chapter two, includes topics such as molecular modeling, property models, molecular descriptors and CAMD solution techniques with many convenient examples provided. The first section in chapter two introduces the fundamentals of computer-aided molecular design. This develops the “big picture” from which increasing levels of detail can be uncovered. The second section discusses molecular descriptors, which provide the ability to mathematically represent molecular structures. These descriptors can be immediately categorized by the dimensionality of information they capture, ranging from 0-D to 4-D, and each of these dimensions are further discussed. The third section introduces molecular modelling techniques, which are necessary to develop molecular descriptors with dimensionality higher than two. These techniques can be based on molecular mechanical simulations, where classical mechanics are used to model molecular systems, or quantum mechanical simulations, which explicitly consider the electrons within the system being studied. Further, the application of molecular and quantum mechanics towards estimating the geometry of a given molecule is considered within the geometry optimization section of chapter two. Sections four and five of chapter two address the concept of property models as well as how they are developed. The model types considered include quantitative structure-property relationships and fragment based property models. Each unique model type has its own requirements for application within a computer-aided molecular design problem, often

requiring several solution techniques. These techniques are discussed within the final section of chapter two, and several examples are provided.

The developed methodology is covered next in chapter three and provides the details of the proposed technique. This includes the initial steps necessary for setting up the problem as well as the subsequent steps taken, ultimately leading to a solution set of molecular structures. Two distinct approaches were developed, and each is best suited for a specific type of molecular design problem. A deterministic approach was developed for smaller problems and this allows for the problem to be thoroughly considered as the entire solution space is searched. In addition, a stochastic technique was developed for larger studies in which it would be too time consuming to consider the entire search space. Within both approaches it is first necessary to identify the appropriate property models, select a data set, perform a conformational analysis on this data set, compress this information and ultimately arrive at a set of molecular fragments with which to proceed in generating solutions. Section one of chapter three covers these initial steps along with the proposed deterministic approach and section two covers the stochastic solution approach. The stochastic approach is an evolutionary algorithm which applies concepts familiar to natural selection. This requires the generation of a starting population along with several operators which alter the population through acts of crossover and mutation. The goal is to apply selective pressure and guide/transform the population to ultimately converge into a set of molecules with the desired characteristics.

This method is exemplified in chapter four, which introduces several case studies including: solvent design, fungicide substituent design and non-peptide mimetic inhibitor design. In addition to these case studies, another section has been included which

provides verification of the underlying assumption associated with this methodology. This assumption is that one can generate an estimation of a potential energy surface, by identifying several likely conformational energy minima, through the use of fragment geometry information. This is one key idea responsible for the increased efficiency associated with the novel approach. Within the appendix, python code for application of each of the discussed techniques can be found.

The deterministic approach towards solving computer-aided molecular design (CAMD) problems with multidimensional descriptors has been published in *Computer Aided Chemical Engineering* (Herring et al., 2012a; Herring et al., 2012b). This initial work allows for the solution of CAMD problems, utilizing descriptors of varying complexity and dimensionality, on a single platform. This approach was extended to consider more complex structures and design characteristics with the structure based design of non-peptide mimetics (Herring et al., 2013), which was published in *Computer Aided Chemical Engineering*. In addition the stochastic approach was exemplified through two publications in *Computer Aided Chemical Engineering* (Herring and Eden, 2014 a,b). Many of the underlying techniques associated with this thesis were also applied in several related studies. For example, an interesting technique for the development of a quantitative structure-property model relating solvent structure to crystal morphology was also developed (Haser et al., 2014) and provides an excellent example of property model development techniques. Also, some concepts of generating solution structures, relating to ionic liquids, from molecular fragments using computer-aided techniques have also been developed (Hada et al., 2013). In addition the quantum chemical characterization of ionic liquid properties has been utilized within a molecular

design application (Davis et al., 2014). Additionally, an invited publication encompassing the techniques and application of the developed stochastic evolutionary molecular design approach will soon be published in a special issue of *Computers & Chemical Engineering*.

2. Background

The background section of this dissertation will present the ideas and techniques necessary for an understanding of the proposed methodology. In addition, it will provide a comparison through which to exemplify the benefits and novel techniques within this approach. The first section introduces the concept of computer-aided molecular design and covers the basic steps seen in problems approached within this field. The second section discusses the various types of molecular descriptors utilized, each of which has its own strengths and weaknesses in characterizing molecules. Descriptors can be categorized based upon the dimensionality of information they capture and this ranges from 0D to 4D, so far. This section is important as it compares each descriptor type, while offering several examples, such that the requirements for developing and using these descriptors can be addressed. The third section introduces various concepts in molecular modeling, including molecular mechanics, quantum mechanics, and geometry optimization. These techniques are necessary for estimation of spatial and electronic properties for molecules in a molecular design study. The fourth background section covers the various types of quantitative structure property (activity) models, and the techniques used to develop these. The approach taken to develop a model of this type can be broken into a few basic steps including variable selection, mapping (e.g. regression analysis), and model verification. Without these models, the design of molecules with desired properties and activities would be much more limited. The fifth section discusses the concept of fragment based property models. The three types covered in detail here are group contribution, pharmacophore and atomic signature based models. Fragment based

property models are considered separately for two reasons: (1) They have a special usefulness in the inverse property model approach since molecular fragments allow the enumeration of solution structures to be done in a much more efficient manner and (2) The methodology proposed in this thesis is based on the utilization of fragment based descriptors, namely the signature descriptor. The final section presented within the background covers the various techniques useful for solving molecular design problems. These have been broadly categorized as being database searches, generate and test approaches and programming/optimization. Several modern examples have been provided for each approach type for a hands on explanation of the pros and cons of each technique. It is the author's intent that, after reviewing these sections, consideration of the proposed methodology will become more tangible and its novel techniques will be more readily noticed.

2.1 Computer-Aided Molecular Design Approach

Computer-aided molecular design (CAMD) involves the selection or identification of molecules with an increased likelihood of exhibiting a set of desired characteristics or attributes. This area falls under the more generalized category of product design, which can further include the consideration of mixture design and sometimes process design. Cussler and Moggridge [1] have suggested these four steps in the product design process:

1. Define needs;
2. Generate ideas to meet needs;
3. Select among ideas;
4. Manufacture product.

The first step is to identify the consumer/customer needs and this can be anything from common macroscopic molecular properties to less tangible attributes such as feel, smell, or color. Physicochemical properties represent explicit property constraints because their values can be determined directly through a model or even determined experimentally. Another form of property constraint includes those which are less tangible, as mentioned earlier, and is referred to as an implicit property. These properties cannot be measured or predicted by a model and must be inferred through databases, past knowledge, and other measured or predicted properties. Once the relevant properties have been identified, it is also necessary to set certain bounds on their acceptable values. This is often done by establishing lower and upper bounds on the properties considered, although sometimes only one bound is necessary. For example, toxicity is often represented by the concentration resulting in a fifty percent mortality rate, LC_{50} , for a given test set and this property only requires a lower bound since an increased value represents increased chemical safety. Other properties, such as the boiling point of a solvent, must fall within certain acceptable limits and are more adequately constrained with an upper and lower bound. It is necessary to ultimately relate these attributes to the underlying chemical structures. The first through third steps represent the core of the molecular design approach, which is where mixture design problems would be considered in addition to single-component molecular design. The inclusion of mixture design necessitates the use of certain models for estimating the mixture properties, often a function of the individual component properties. The fourth step involves the design of a process which can create the desired product in a safe and economic manner. When product and process design are done simultaneously this is known as an integrated

approach and it allows for the identification of optimal component and process specifications. For the purposes of this defense, CAMD will consist of the first through third steps while excluding any mixture design applications.

Once the desired attributes, e.g. property bounds, have been decided, what remains is to identify solutions exhibiting these attributes and choose the optimal solution(s). To do this, structure-attribute relationships, also referred to as property models, are utilized. There are many types of property models but they all aim to create a mathematical relationship between the underlying chemical structure and the property of interest. Molecular descriptors, which capture various features of the molecular structure, are used as independent variables and the properties are dependent variables in this case. When used in a forward manner, these models can predict the property exhibited by a certain molecule within the applicable domain (AD) of that model. The AD can be defined in many different ways and it represents a region of space where an increased confidence in the predicted property value can be expected, as long as the molecule lies within this space. This is pointed out as being important in the CAMD approach since many times the molecules of interest are not available in the test set, meaning their properties are unknown and reliable property estimations are crucial.

Achenie et al. (2003) have provided the following generic mathematical programming representation of the typical CAMD problem:

$$F_{OBJ} = \max\{C^T y + f(x)\} \quad (1)$$

s.t.

$$h_1(x) = 0 \quad \dots \text{process design specs} \quad (2)$$

$$h_2(x) = 0 \quad \dots \text{process model equations} \quad (3)$$

$$h_3(x) = 0 \quad \dots \text{CAMD specifications} \quad (4)$$

$$l_1 \leq g_1(x) \leq u_1 \quad \dots \text{process design constraints} \quad (5)$$

$$l_2 \leq g_2(x) \leq u_2 \quad \dots \text{CAMD constraints} \quad (6)$$

$$l_1 \leq By + Cx \leq u_3 \quad \dots \text{logical constraints} \quad (7)$$

In the equations above, x represents a vector of continuous variables and y represents the vector of binary integer variables. Harper et al. (1999) have provided a classification for the various CAMD approaches and this includes (a) Database search, (b) Generate and Test, and (c) Mathematical programming and genetic algorithm. The equations utilized in each type of approach will help clarify the techniques through mathematical examination. A database search would satisfy only equation (6) above and this approach would be limited to the consideration of only existing molecules. The generate and test approach relies on equation (4) to generate feasible molecules and equation (6) to test if they are within the property bounds desired. This approach, while more computationally intensive than the conventional database search, introduces the opportunity to consider novel structures. With this new opportunity comes a challenge, which is to decide how to generate these new structures such that they fall reasonably within the AD of any property models utilized. Most attempts at this have come from fragment based approaches in which molecular fragments are used as building blocks to generate molecules within a controlled region of chemical space. The generate and test approach is most simply an exhaustive effort such that all feasible molecules, within the chemical space designated by the chosen building blocks, are tested. This often leads to what has been termed ‘combinatorial explosion’, which results from a combinatorial mathematics

problem becoming too large to solve in a reasonable amount of time because of the sheer number of possible combinations. When this is the case, many have turned to the third approach type, mathematical programming and genetic algorithm, which aim to alleviate this problem. The programming approach takes many different forms depending on the nature of the constraint equations involved, e.g. being linear or nonlinear. When the CAMD problem can be defined in terms of linear equations, this often allows for the identification of a globally optimal solution through methods such as the Simplex Method (Nelder and Mead, 1965). Otherwise, there are equivalently many techniques for the solution of non-linear programming problems including an array of stochastic techniques which are adept at handling the combinatorially large and highly non-linear problems encountered in CAMD. Overall, the adoption of CAMD methodologies has proven a very fruitful effort in terms of identifying and understanding the effects of varying molecular structure and has also saved time and expenses in the development of new chemical solutions.

2.2 Molecular Descriptors

Molecular descriptors provide a way to numerically represent certain features of a molecule, allowing for the mathematical characterization of structures such that the desired properties and activities are defined as a function of these descriptors. This is more formally expressed by Todeschini and Consonni (2009) as: "The molecular descriptor is the final result of a logic and mathematical procedure which transforms chemical information encoded within a symbolic representation of a molecule into a useful number or the result of some standardized experiment." Descriptors can be initially separated into two distinct categories: experimental measurements, often

physicochemical properties like boiling point, and theoretical molecular descriptors, which are derived from symbolic representations of a molecular structure. While the physicochemical properties might not be available for all compounds, theoretical descriptors can be calculated for any possible chemical structure and contain no statistical error due to experimental noise. Theoretical molecular descriptors can further be classified by the dimensionality of information they capture. This classification scheme results in the following categories of theoretical descriptors: 1) 0D-descriptors (i.e. constitutional and count descriptors), 2) 1D-descriptors (i.e. list of structural fragments), 3) 2D-descriptors (i.e. graph invariants) 4) 3D-descriptors (e.g. quantum-chemical descriptors and surface descriptors) 5) 4D-descriptors (e.g. CoMFA derived descriptors). Comparative molecular field analysis (CoMFA) was developed by Cramer et al. (1988) and generates a structure/activity correlation based upon the three-dimensional steric and electrostatic fields of a molecule. Extension of descriptors to include four-dimensional information typically requires the three-dimensional analysis of a set of conformational isomers; This information can then be collectively utilized to generate the respective four-dimensional descriptors. In addition, there can be theoretical descriptors which are not cleanly categorized as above. One example of this would be the weighting of conventional topological indices by geometric information as applied in the method of ideal symmetry (Toropov, 1998). These indices were utilized in a predictive manner to calculate the boiling points of a series of alkanes and showed a better performance than the original topological descriptors.

Molecular descriptors have an inherent level of degeneracy that depicts how well individual molecules are differentiated based solely upon their descriptor values. This

feature is very important when the model developed will be utilized in an inverse manner to predict structures meeting the desired property/attribute value. Models built with descriptors higher in degeneracy would have many more solution structures than one built with less degenerate descriptors. Since the search space for such applications can often be extremely large, it is desirable to reduce the solution set as much as possible. A general trend of decreasing degeneracy with increasing descriptor dimensionality can be observed. This trend is followed by an often significant increase in computational demands brought about by the inclusion of 3D and 4D descriptors.

In addition to having a certain level of degeneracy, descriptors can also be characterized by their invariance properties. Invariance refers to the ability of the descriptor calculation algorithm to give a consistent value regardless of the form of molecular representation utilized. A minimal requirement for molecular descriptors is invariance to molecular numbering or labeling. The specific case of chemical invariance considers whether or not the various atom types of a structure affect the descriptor value. For example, topological descriptors utilizing atom type in their calculation are known as topochemical indices, and those relying solely on connectivity information are known as topostructural indices.

2.2.1 0D-1D Descriptors

0D-descriptors, also known as constitutional descriptors, are the simplest to compute and still offer reasonable discrimination power for specific situations. Some example of descriptors of this nature would be molecular weight, bond counts, atom counts, and fragment counts. The example shown in Fig. 2.1 represents the utilization of carbon count, a 0D descriptor, in creating a linear property model for the boiling point of a series of

alkanes (C1-C7). This simple technique was able to account for 97% of the variance seen in the data.

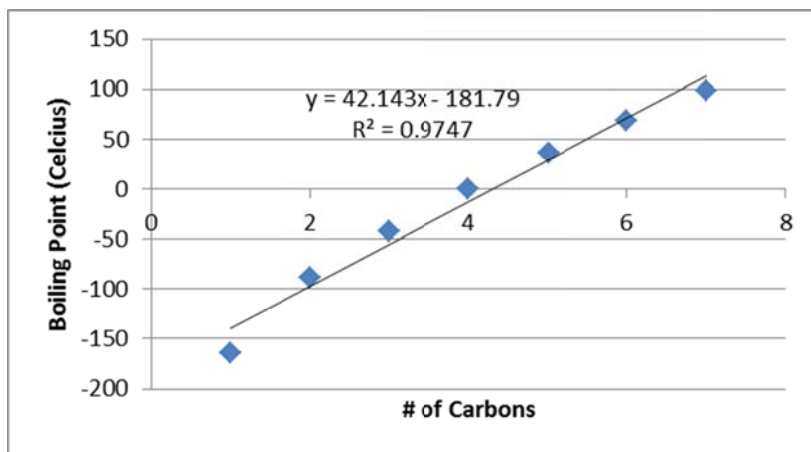


Figure 2.1 - Example of constitutional descriptor correlation.

1D-descriptors essentially account for certain structural fragments found within the molecules under consideration. This can be a complete or partial list of functional groups, substituents, etc. This method of molecular classification is often used to search large chemical databases in an expedited manner to identify structures with a certain level of ‘similarity’ (e.g. containing the desired pharmacophoric groups identified as important towards a certain biological activity).

2.2.2 2D Descriptors

Molecules can be represented as graphs and this approach is often termed chemical graph theory (Balaban, 1976; Trinajstic, 1992; Rouvray, 1971). A molecular graph G can be described by a set of vertices V , representing the atoms, and a set of edges E , representing the bonds. In addition, the fragments present in a molecular graph can be represented as sub-graphs in the same manner. A number of descriptors can ultimately be calculated

from this representation of a molecule and they are collectively known as graph theoretical or topological indices.

Two vertices connected by an edge are considered to be adjacent and the adjacency matrix **A** can uniquely describe a molecular graph. The adjacency matrix has elements a_{ij} equal to 1 for all adjacent vertices and 0 otherwise. A path, in this context, is a succession of non-repeating edges such that there is no discontinuity from one point to another. With this in mind, for each pair of vertices in a chemical graph there exists at least one path connecting them. The distance matrix **D** represents the shortest path d_{ij} between all pairs of vertices in a graph. Both matrices are symmetrical with respect to their main diagonals and have diagonal entries of zero since a vertex is unique and cannot be connected to itself. These matrices are often utilized as the intermediate representation of a pure chemical graph from which topological indices are derived.

The Wiener index (Wiener, 1947a,b), denoted by W , was among the first, and most successful, topological indices utilized in structure property correlations. This topological index is calculated from the distance matrix and is essentially the half-sum of all entries in this matrix, being symmetric. As developed in Fig. 2.2, it can be seen that more compact molecular graphs will have a smaller W value.

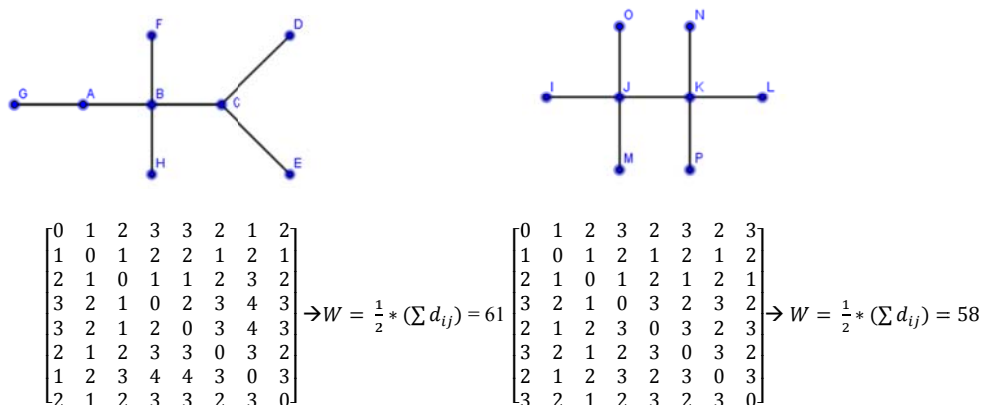


Figure 2.2 - Example calculations of Wiener index.

Another widely used descriptor type within the 2D, or topological index, domain is the connectivity index, which is considered the most successful index and has many different forms. A specific nomenclature has been developed to differentiate and identify these indices which are all denoted by χ . Two superscripts and one subscript are used to further define the index type. The left-side superscript can attain a value of either zero or any positive integer and designates the index order. The right side subscript (P, C, PC, or CH) specifies the subclass of molecular connectivity index which could be path, cluster, path/cluster, or chain-type. The path type index is assumed when no right side subscript is noted. Each index calculation is based only on the non-hydrogen atoms within a molecule and these atoms are represented by their atomic δ value, which is equal to the number of adjacent non-hydrogen atoms. For example, the first-order ${}^1\chi$ molecular connectivity index is calculated as shown in Eq. (2.1) where i and j correspond to pairs of adjacent non-hydrogen atoms and the summation is over all bonds in the graph.

$${}^1\chi = \sum (\delta_i \delta_j)^{-0.5} \quad (2.1)$$

$$\delta_i = (Z_i - H_i)/(Z_i - Z_i^v - 1) \quad (2.2)$$

The valence connectivity index is an extension to the original connectivity index and represents an attempt to account for the chemical nature of vertices, or atoms, in a molecular graph. In this case, vertices are no longer represented by their degree and are weighted by valence delta values δ_i , which are calculated as shown in Eq. (2.2) where Z_i^v is the number of valence electrons in atom i , Z_i is its atomic number, and H_i is the number of hydrogen atoms attached to atom i . The most recent extension of the original connectivity index is the electrotopological state index (Kier and Hall, 1999). The idea with this new index is to consider that each atom within the molecule resides in a field composed of every other atom and the result of these interactions is modification of the intrinsic state of that atom to produce its bonded state within the context of the whole molecule. The resulting electrotopological index combines the electronic and topological effects acting on each atom within its field. The electrotopological state (E-state) of an atom in a molecule is formulated as an intrinsic value, I_i , plus a perturbation term, ΔI_i , arising from the electronic interaction and modified by the molecular topological environment of each atom in the molecule. The intrinsic value is calculated as shown in Eq. (2.3) where δ and δ^v are the previously discussed delta and valence

$$I = \left[\left(\frac{2}{N} \right)^2 \delta^v + 1 \right] / \delta \quad (2.3)$$

delta values while N represents the principal quantum number. The field effects are calculated as a perturbation on the original atoms intrinsic value as shown in Eq. (2.4)

$$\Delta I_i = \sum_{j=1}^N (I_i - I_j)/r_{ij}^2 \quad (2.4)$$

where N represents all atoms in the structure and r_{ij} is the topological distance, or number of bond, between atoms i and j . The resulting electrotopological state of each atom, S_i , is then calculated as a summation of its intrinsic value and perturbation value. This descriptor is differentiated from the previously applied ‘whole molecule’ descriptors in that its value for specific atoms was utilized by itself, while the rest of the structure was captured through the perturbation adjustment.

2.2.3 3D Descriptors

Just as 2D descriptors were termed topological indices, 3D descriptors are known as topographic indices and they represent geometry dependent invariants of molecular graphs. These descriptors were introduced because conventional topological indices could not account for spatial molecular information such as stereoisomerism (e.g. cis and trans) and molecular geometry estimations became more feasible through improvements in computational power and geometry development algorithms. These descriptors require information about the three dimensional, or geometric, arrangement of atoms in a molecule and there are multiple techniques, with varying accuracy, available to estimate this information. Experimentally, the molecular geometry can be obtained by various spectroscopic and diffraction methods. Infra-red, microwave and Raman spectroscopy can be used to obtain information about the geometry of a molecule based upon the vibrational and rotational absorbance detected by these techniques. In addition, x-ray

crystallography, neutron diffraction and electron diffraction can be used to obtain information about the structure of crystalline solids based on the distance between nuclei and concentration of electron density. There are many techniques in the field of molecular modeling that can be used for the computational based determination of molecular geometry. These techniques can generally be categorized as empirical (e.g. molecular mechanics), semi-empirical and *ab-initio* techniques. Empirical applications use classical mechanics to model molecular systems and rely on force fields to provide information about the feasibility of a suggested molecular structure. These methods are termed ‘empirical’ because the force-field utilized has been parameterized to fit experimental data. Semi-empirical quantum chemistry methods often have increased accuracy over purely empirical methods because they consider some form of electron correlation, whereas this information is not explicitly considered in an empirical approach. Semi-empirical methods offer a nice middle ground for molecular geometry determination with reasonable accuracy and computational complexity. The ‘semi-‘ part of semi-empirical refers to the fact that the two-electron part of the Hamiltonian is not explicitly included, yet has been parameterized to fit either experimental or *ab-initio* results. This leads to the most computationally demanding, and often most accurate, technique for molecular geometry determination, termed *ab-initio* quantum chemistry. Ab-initio means ‘from first principles’ and was a term first coined by Allen et al. (1960). These techniques do not rely on experimental data and electrons are explicitly represented as the Schrodinger equation is solved to obtain very accurate geometry estimations. The details of these various techniques will be covered in further detail in

section 2.3. An overview of the various types of geometric descriptors is provided in Figure 2.3.

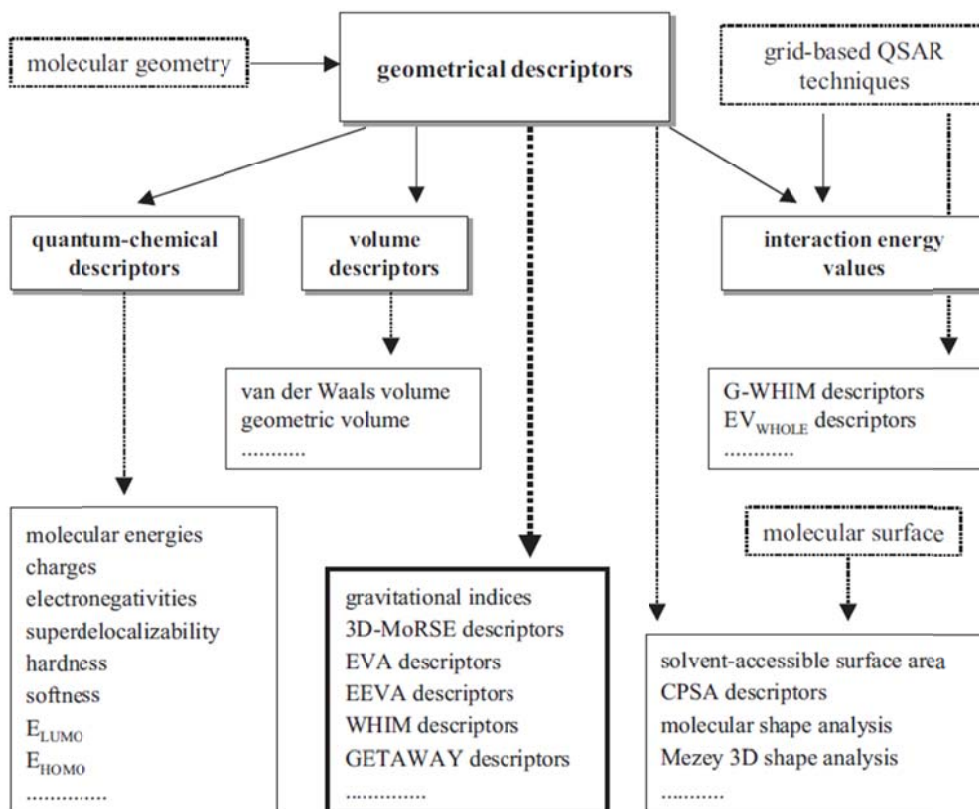


Figure 2.3 - Overview of geometrical descriptors.

The inclusion of spatial molecular information in graph theoretical descriptors was first introduced by Randić (1987) who used the term topographic descriptor. Another turning point in the development of topographic descriptors was given by Bogdanov et al. (1989), who calculated the three-dimensional Wiener number, which is an extension of its original topological counterpart. There are several other possibilities for calculating variants of the original Wiener index which might include using the summation of bond lengths between respective atoms as well as Euclidean distances, as done by Bogdanov et

al. (1989). However, a study by Castro et al. (2002) considers the relationship between these descriptors in the context of utilization in property correlation. It was found that there is a strong linear correlation between the original Wiener index and each of its modified forms. This leads to the conclusion that the original definition of this descriptor is sufficient for property correlation, and the introduction of variants on this technique introduce the possibility for only slight improvements in a given QSPR model. This study reinforces the concept that descriptors, when utilized in structure property correlations, should be linearly independent.

Many of the topographic descriptors were developed as an extension upon existing topological based concepts. This was the case for Diudea et al. (1995) who proposed two types of topographic indices of centrality and centrocomplexity which were based on 3D distances provided by molecular mechanics calculations. The topological counterparts to these types of descriptors begin with the ‘through bond’, topological distance matrix as well as the layer matrix, **LM**. The topological distance matrix is a symmetric matrix **D** whose entries, d_{ij} , correspond to the number of bonds between atoms i and j . The ‘through space’ matrix utilized to develop these novel topographic descriptors is symmetric just like the original topological matrix, however, Euclidean distances are used in place of the number of bonds between two respective atoms. The layer matrix is a bit more complex than the distance matrix and it collects the properties of vertices u located on concentric shells (layers) $G(u)_j$, at a distance j around the vertex i in the graph G , and can be defined as seen in Eq.(2.5) and (2.6):

$$lm_{ij} = \sum_{u \in G(u)_j} m_u \quad (2.5)$$

$$LM(G) = \{lm_{ij}; i \in [1, n]; j \in [0, d]\} \quad (2.6)$$

In the above equations, m and M are labels for a given property and the corresponding matrix, respectively; n is the number of vertices in the graph; and d stands for the diameter of the graph, which is the largest topological distance in the graph. The new three-dimensional layer matrix follows the same convention as its topological counterpart as seen in Eq.(2.5) and Eq.(2.6), however, m_u is now defined as in Eq.(2.7).

$$m_u = \sum_{\text{all } v \in G} 3d_{uv} = 3D_u \quad (2.7)$$

The two new local vertex invariants (LOVI's) of c (centricity) and x (centrocomplexity) have been defined as shown in Eq.(2.8) and Eq.(2.9), respectively.

$$c(LM)_i = \left[\sum_{j=1}^{ecc_i} (lm_{ij})^{j/dsp} \right]^{-1} \quad (2.8)$$

$$x(LM)_i = \left[\sum_{j=0}^{ecc_j} lm_{ij} 10^{-zj} \pm l_i \right]^{\pm 1} t_i \quad (2.9)$$

$$l_i = f_i \left(\frac{lm_{i0}}{10} + \frac{lm_{i1}}{100} \right) \quad (2.10)$$

$$f_i = \sum_u (c_{iu} - 1) \quad (2.11)$$

In the above equations, ecc_i is the eccentricity of vertex I (the maximal topological distance from vertex I to any vertices in the graph); dsp is a specified topological distance, usually larger than the diameter of the graph; z is the number of digits of the max lm_{ij} value in the graph; l_i is a local parameter for multiple bonds; f_i is a multigraph factor, with c_{iu} being the conventional bond order; t_i is a weighting factor accounting for heteroatoms. These new indices were able to correlate well with the van der Waals surface area for a set of 17 geometric heptane isomers, thus exemplifying their ability to differentiate conformational isomers. In addition, the indices were able to correlate with the toxicity of ethers on mice. This exemplifies the complexity of topographic descriptors that have been developed in the last few decades as well as their abilities to correlate well with various properties and activities of interest.

The previously described topographic descriptors were all based upon the Euclidean distance matrix. One approach by Estrada and Ramirez (1996) introduces a new topographic index derived from the three-dimensional analogue of the conventional topological edge matrix. The edge matrix \mathbf{E} is a square and symmetric matrix in which the rows and columns correspond to edges, or chemical bonds, within a molecular graph G . The non-diagonal entries of this matrix are either ones or zeroes depending on whether the corresponding bonds are touching or not, respectively. This approach uses molecular graphs with edges weighted by bond orders calculated from quantum chemical methods. The bond orders in this method are called valence indexes and are calculated as shown in Eq.(2.12)

$$\rho_{AB} = \sum_{\lambda}^A 2p_{\lambda\lambda} - \sum_{\lambda}^A \sum_{\sigma}^B p_{\lambda\sigma}^2 \quad (2.12)$$

where,

$$p_{\lambda\sigma} = 2 \sum_1^{occ} C_{i\lambda} C_{i\sigma} \quad (2.13)$$

are the elements of the density matrix and eigenvectors $C_{i\lambda}$ sum is over all occupied orbitals. Elements of the edge adjacency matrix for the weighted molecular graph are defined in a more complex way. Let e_i and e_j be two adjacent edges in G. If e_i is incident with vertices v_a and v_b , and e_j is incident with v_b and v_c , then the elements e_{ij} and e_{ji} of the **E** matrix are ρ_{BC} and ρ_{AB} , respectively. With bond orders defined in this manner, edge degrees, $\rho(e_i)$, are defined as the sum of elements of i th row in **E** matrix as in Eq.(2.14):

$$\delta(e_i) = \sum_j e_{ij} \quad (2.14)$$

The topographic edge connectivity index $\epsilon(\rho)$ was developed from this new matrix and is calculated as shown in Eq.(2.15):

$$\epsilon(\rho) = \sum_s [\delta(e_i)\delta(e_j)]_s^{-1/2} \quad (2.15)$$

This descriptor was utilized to generate regressions for the molar refractivity of a series of 69 C₅-C₉ alkanes. Calculation of the index was performed by using bond orders

calculated from the quantum chemical semi-empirical method PM3 (Stewart, 1991). Molar refractivity was chosen as an important property because of its ability to model the steric and hydrophobic interaction between drugs and biological receptors. This descriptor was also applied in a QSAR study in which the cumulative urinary excretion, in humans, of unchanged drug expressed as a percentage of the administered dose was regressed against this variable. The model was able to more accurately characterize the phenomena utilizing the topographic descriptor when compared to previous studies (Testa and Salvesen, 1980) which used the n-heptane-water partition coefficient as an independent variable.

In addition to the plethora of alignment-independent topographic descriptors mentioned before, there exists a group of descriptors which require some degree of alignment between each molecule in a data set. One prominent example would be the comparative molecular field analysis (CoMFA) technique (Cramer et al., 1988). This work began with the realization that, at the molecular level, the interactions which produce an observed biological effect are usually non-covalent; and molecular mechanics force fields, which account for these non-covalent interactions as steric and electrostatic forces, can account for a great variety of molecular properties. The CoMFA approach has been outlined in Figure 2.4.

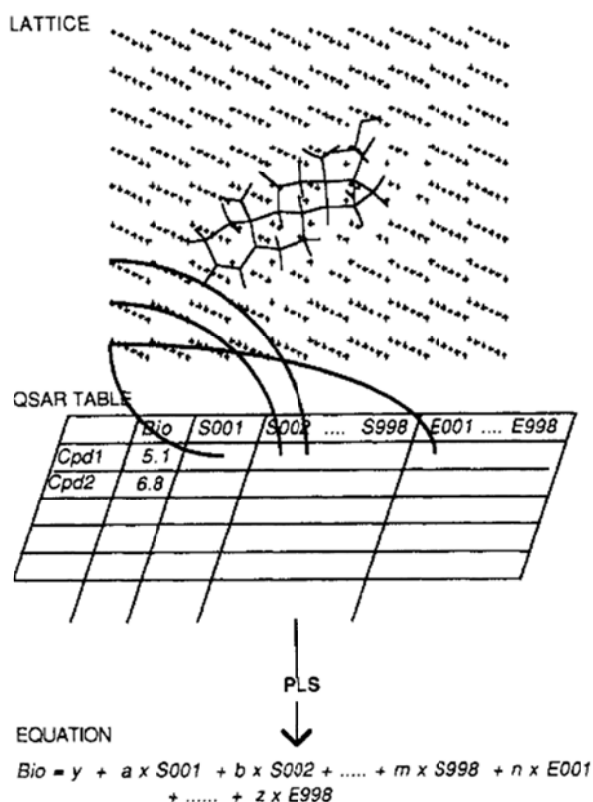


Figure 2.4 - Outline of CoMFA approach.

Steric and electrostatic interaction energies are calculated at each of the lattice points, seen in Fig.(2.4), between the compound of interest and a “probe atom.” This probe atom has the van der Waals properties of an sp^3 carbon atom, which includes the volume and summation of attractive/repulsive forces seen within this carbon, with a charge of +1.0. This information is stored in a table, in which there are many more columns (data points) than rows (compounds). Such a data format is well-suited to the application of the partial least-squares (PLS) methodology for development of a quantitative structure activity relationship. PLS regression is a statistical method that finds a linear regression model by projecting the predicted variables and the observable variables to a new space. The most important aspect of an alignment-dependent model is the actual molecular alignment

process. Previous applications relied on a chemist to align the molecules, however, this methodology has implemented a new “Field Fit” procedure in which the RMS difference in the sum of steric and electrostatic interaction energies, averaged across all lattice points, between one molecule and some template or set of molecules, is minimized with respect to the six rigid-body degrees of freedom and/or any user-specified torsion angles. This approach was tested (Cramer et al., 1988) on a data set of 21 various steroids, which have experimentally determined affinities to corticosteroid- and testosterone-binding globulins. Since the goal of the CoMFA methodology is to generate models capable of predicting the activities, or binding affinities, of compounds not in the training set, the binding affinity of ten steroids (not in the original training set) was predicted by the model developed. The predictive R^2 value for this example was 0.65, which was higher than any model developed using the conventional QSAR methodology with various descriptors. In this case, the descriptors are not as ‘clear cut’ as previous approaches but they are initially represented as the energies of repulsion (steric and electrostatic) between the test molecule and the probe atom, calculated at the various lattice points. This information is further turned into latent variables, which captures the most important aspects, within the lattice, accounting for variance in the predictor variables (i.e. affinities in this case).

The Comparative Molecular Similarity Indices (CoMSIA) (Klebe et al., 1994) approach is similar to CoMFA in that an atomic probing throughout a grid lattice is utilized. However, CoMSIA uses a different potential function, which is Gaussian-type, to model the repulsion. The use of a Gaussian-type potential function instead of the previously applied Lennard-Jones and Coulombic functions allows for more accurate

information to be obtained from points within the molecular structure. This was one drawback with the CoMFA approach because unusually large energy values were obtained at these points, because of the nature of the potential function used, and cut-offs had to be applied so the data wasn't skewed. There are several other probe based techniques available for characterizing a set of molecular structures with respect to a predictor variable (typically a biological property) and these include Comparative Molecular Moment Analysis (CoMMA) (Silverman and Platt, 1996), VolSurf (Cruciani et al., 2000), and Grid-Independent Descriptors (GRIND) (Pastor et al., 2000). Each technique has its own strong points and limitations, but they all represent the realization that shape based descriptors excel at describing interaction based properties (e.g. binding affinity).

2.3 Molecular Modeling

Molecular modeling encompasses all of the techniques and tools useful for modeling the motions and interaction of molecules. These techniques are used in the fields of computational chemistry, drug design, computational biology, materials science, and now many engineering fields for studying molecular systems ranging from single small molecules in the gas phase to large biological molecules (e.g. receptor ligand complexes) and material assemblies. There are many approaches available for the treatment of molecular structures ranging from modeling atoms as the smallest individual unit (in the molecular mechanics approach) to explicitly modeling the electrons in each atom (in the quantum chemistry approach). The information gained from these techniques is useful in the development of three-dimensional descriptors, which have applications in a wide variety of structure-activity (property) correlations.

2.3.1 Molecular Mechanics

Molecular mechanics refers to the use of classical mechanics in describing the motions of atoms and molecules. These models treat atoms as point charges with the nucleus and associated electrons combined as one. The energy associated with a molecular structure, which is used to measure its likelihood of occurrence, is calculated through the means of a force-field. Within this force-field, otherwise known as a potential function, varying terms are used to summarize the potential energy associated with a collection of atoms. Each atom in the molecule is represented by its coordinates, which can be internal or external. External coordinate systems simply provide an x, y and z coordinate value for each atom within a Cartesian coordinate system. Internal coordinates make use of the inherent nature of these many body systems by referring to bond-lengths, bond angles, and torsional angles. Force-fields are most often defined in terms of internal coordinates, whereby energetic penalties are associated with the deviation of bonds and angles away from their 'preferred' or 'equilibrium' values. In addition, the force field contains terms that describe the non-bonded interactions between each atom. A simplified representation of a typical force-field can be seen in Eq.(2.16).

$$E_{Total} = E_{bonds} + E_{angle} + E_{dihedral} + E_{non-bonded} \quad (2.16)$$

$$E_{non-bonded} = E_{electrostatic} + E_{van\ der\ Waals} \quad (2.17)$$

It can be seen that the total internal energy, as provided by the force field, consists of terms related to bond lengths, angles, dihedral angles as well as non-bonded interaction. These non-bonded interactions, as shown in Eq.(2.17), represent the summation of electrostatic and van der Waals forces. Van der Waals forces are typically modeled by the

Lennard-Jones potential, which adequately describes the forces of attraction and repulsion felt between two atoms. Furthermore, the electrostatic forces are typically modeled by Coulomb's Law, which accounts for interactions between electrically charged atoms. A more expanded form of a generalized force-field can be found in Eq.(2.18).

$$\begin{aligned}
 E(r^N) = & \\
 & \sum_{bonds} \frac{k_i}{2} (l_i - l_{i,0})^2 + \sum_{angles} \frac{k_i}{2} (\theta_i - \theta_{i,0}) + \sum_{torsions} \frac{V_n}{2} (1 + \cos(n\omega - \gamma)) + \sum_{i=1}^N \sum_{j=i+1}^N (4\varepsilon_{ij} [(\frac{\sigma_{ij}}{r_{ij}})^{12} - (\frac{\sigma_{ij}}{r_{ij}})^6] + \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}})
 \end{aligned} \tag{2.18}$$

In Eq.(2.18) E stands for the potential energy of the system, consisting of a collection of atoms and/or molecules, and r^N represents the coordinates of all N atoms within this system. The first term represents the energy contribution from all bonded atoms in the system, and these are modeled by a harmonic potential function. $l_{i,0}$ stands for the equilibrium bond length between any two atoms. The second term, which represents the energy contribution from three consecutive atoms connected by two bonds, also has an equilibrium angle which is dependent on the atoms involved. Any deviation from this angle represents an increased contribution to the total energy of the system. The third term accounts for energy contributed by all torsional angles within the system. Last but not least, the fourth term summarizes all energy contributions from non-bonded atomic interactions. The fourth term is a summation of the previously mentioned van der Waals and electrostatic forces and accounts for 'through space' interactions.

A force-field is typically parameterized, which means that the variables in Eq.(2.18) are calculated, with a specific set of structures and properties. The goal of this

parameterization is to be able to most accurately reproduce the given properties within the region of chemical space represented by chosen set of molecular structures. This might be, for example, useful for accurately deciding the geometries of a series of hydrocarbons or predicting the infra-red spectra for a set of ionic liquids. The point is that the information within these parameterized force-fields is only useful for the task around which it was designed. It is more common to develop a force-field for a set of molecules than for a single molecule due to the considerable effort required. As such, the ability of a single force-field to accurately model a larger group of structures is known as its transferability. Often times, an increase in transferability leads to a decrease in accuracy. Therein lies one of the fundamental trade-offs when developing a given force-field. Also, the format of Eq.(2.18) is not the only way to represent this model and is by no means the most accurate. These models are ‘empirical’ and as such the model format is developed to have the most accurate results possible. (Leach, 2001)

2.3.2 Quantum Chemical Methods

Quantum mechanics (QM) attempts to describe molecules in terms of interactions between nuclei and electrons. The molecular geometry is determined through identifying a minimum energy arrangement of nuclei in a molecule or set of molecules. This process has been made much more reasonable through a series of approximations upon the original formulation based upon the Schrodinger equation shown in Eq.(2.19).

$$\hat{H}\Psi = E\Psi \tag{2.19}$$

In Eq.(2.19), Ψ is a many-electron wavefunction and \hat{H} is the Hamiltonian operator, also known more simply as the Hamiltonian, which can also be represented as in Eq.(2.20).

$$\begin{aligned} \hat{H} = & -\frac{1}{2} \sum_i^{\text{electrons}} \nabla_i^2 - \frac{1}{2} \sum_A^{\text{nuclei}} \frac{1}{M_A} \nabla_A^2 - \sum_i^{\text{electrons}} \sum_A^{\text{nuclei}} \frac{Z_A}{r_{iA}} \\ & + \sum_{i < j}^{\text{electrons}} \sum_j \frac{1}{r_{ij}} + \sum_A^{\text{nuclei}} \sum_B^{\text{nuclei}} \frac{Z_A Z_B}{R_{AB}} \end{aligned} \quad (2.20)$$

In Eq.(2.20) Z is the nuclear charge, M_A is the ratio of mass of nucleus A to the mass of an electron, R_{AB} is the distance between nuclei A and B , r_{ij} is the distance between electrons i and j and r_{iA} is the distance between electron i and nucleus A . This equation cannot be solved exactly for even a simple two-electron system such as a helium atom or a hydrogen molecule and approximations must be introduced. One such short-cut is known as the Born-Oppenheimer Approximation (Born and Huang, 1988), which assumes that the motion of the electrons is much faster than that of the nuclei thus decoupling the two and producing the “electronic” Schrodinger equation. Even with this simplification, the electronic Schrodinger equation is still intractable and further approximations must be made. The Hartree-Fock approximation (Slater, 1930) was made and insists that the electrons move independently of each other. This results in the total wavefunction being written in the form of a single determinant, also known as a Slater determinant. This also leads to a set of coupled differential equations known as the Hartree-Fock equations, each involving the coordinates of a single electron. At this point, the numerical solution to these equations is possible, however further approximations have been introduced to transform them into a set of algebraic equations. The linear combination of atomic

orbitals (LCAO) (Clark and Koch, 1999) was the next step on the road to creating a more tractable representation of a molecule through the quantum chemical formalism. The Hartree-Fock and LCAO approximations, when applied to the electronic Schrodinger equation, ultimately lead to the Roothaan-Hall equations (Roothaan, 1951). Methods resulting from the solution of the Roothaan-Hall equations are termed Hartree-Fock models as well as *Ab Initio* (“from the beginning”). These models allow for the evaluation of first and second derivatives of energy which make both geometry optimization and determination of vibrational frequencies, respectively, possible.

Often times, solutions generated through means of a Hartree-Fock model result in an overestimation of electron-electron repulsion energies because pair-wise electron interactions are not directly considered and they have a tendency to “get in each other’s way.” This situation is corrected for through electron correlation, which accounts for coupling of electron motions and leads to a lessening of the electron-electron repulsion energy. There are many different techniques for this approach. One is known as a Density Functional model (Becke, 1988) which introduces an approximate correlation term in an explicit manner without being significantly more costly than Hartree-Fock models. Configuration interaction models (Sherrill and Schaefer, 1999) and Moller-Plesset models (Moller and Plesset, 1934) extend the flexibility of Hartree-Fock models by mixing ground-state and excited-state wavefunction, however they are significantly more costly than the Hartree-Fock models.

Semi-empirical models follow directly from the original Hartree-Fock models and represent a simplification that allows the solution of much larger problems. The size of the problem is greatly reduced by only considering valence electrons and ignoring the

core electrons. The central approximation, which allows greatest reduction in overall computation, insists that atomic orbital residing on different atomic centers do no overlap and this is referred to as the Neglect of Diatomic Differential Overlap (NDDO) approximation (Pople et al., 1967). Additional approximations are introduced to further simplify calculations and provide a framework for the introduction of empirical parameters. These parameters can be fitted to experimental data as well as ab initio calculations. Some popular examples of semi-empirical models include Austin-Model 1 (AM1) (Dewar et al., 1985) and Parameterized Model number 3 (PM3) (Stewart, 1989).

2.3.3 Geometry Optimization

The way in which the potential energy of a molecule varies with its atomic coordinates is known as the potential energy surface. Geometry optimization, also known as energy minimization, encompasses various techniques useful for exploring this very complicated potential energy surface (PES) in search of a minimum energy conformation, or arrangement of atoms. For a system with N atoms, the energy is a function of the $3N-6$ internal or $3N$ Cartesian coordinates which is a highly dimensional set of information. There is particular interest in minimum points on the PES as they represent the most stable conformations associated with a molecule. In most cases, there are many local minima; however, there is only one global minimum. In addition to minima, another point of interest on a PES is known as a saddle point, which corresponds to the highest point on the pathway between two minima where the arrangement of atoms is known as a transition structure. (Leach, 2001)

There are many methods useful for the exploration of a PES, which can be defined in terms of internal or Cartesian coordinates. Minima are typically found using numerical

methods since the application of analytical methods becomes too computationally demanding for most systems. These techniques are applicable in both molecular and quantum mechanical representations and can initially be differentiated based upon whether or not derivatives of the energy with respect to coordinates are calculated. Various techniques are adept at exploring the PES in their own unique way. For example, some methods may only be able to move 'downhill', corresponding to a decrease in energy, but others may move in either direction. One example of a non-derivative minimization method is known as the simplex method (Nelder and Mead, 1965), which moves around the PES in a fashion that has been likened to the motion of amoeba. This method is most suitable for determination of an energy minimum when the starting point configuration is very high in energy; however, it is rarely able to escape local minima traps. The techniques in which derivatives are utilized can be immediately differentiated based on whether or not they calculate the first or second order derivative of the PES. First order minimization algorithms that are most frequently used in molecular modeling are the method of steepest descents and the conjugate gradient method. The starting point for most of these techniques is generated from user input, typically with the help of experimental data. The steepest descent method (Curry, 1944) moves in the direction parallel to the net force, which corresponds to walking straight downhill. In the conjugate gradients method (Reeves and Reeves, 1964), the gradients at each point are orthogonal but the directions are conjugate. Second order derivative methods utilize the second order derivative, which provides information about the curvature of a function, in addition to the first order derivative of the PES. The Newton-Raphson method (Roger, 1987) is the simplest second-order method and is more suited towards smaller molecules.

These are the techniques available for aid in the determination of conformational minima. This information is useful in the development of three-dimensional descriptors and structure-property (activity) relationships as well a plethora of other applications. Conformational analysis is the study of the conformations of a molecule and their influence on its properties. The various conformations that a molecule can obtain are interconverted by rotation about a single bond, or a collection of bonds. A conformational search has the objective of identifying the preferred conformations of a molecule, which determine its behavior. There are many techniques for conducting these searches and they can be categorized as systematic or stochastic. While the systematic approach is often more likely to identify a complete collection of minima, a stochastic approach is more easily applied when the PES is very complicated. The relative populations of a molecule's conformations can be estimated through statistical mechanics via the Boltzmann distribution, which is also known as a Gibbs distribution (Gibbs, 1902). However, it must be noted that many of these simulations, in their most basic form, correspond to phenomena in the gas phase at relatively low temperatures and thus do not account for many solvation or interaction effects. In addition, in the case of spatially active molecules (e.g. ligand receptor complexes), the active conformation may not even correspond to any of the identified local minima. Fortunately, there are many techniques available for increasing the 'realness' of these simulations and they are becoming increasingly applied in studies today as computational capabilities are rapidly improving. (Leach, 2001)

2.4 QSAR/QSPR

The purpose of a structure activity relationship (SAR) is to create a mapping between the structural characteristics of a group of compounds and a desired activity. The same can be said for structure property relationships (SPR's), which aim to characterize the effect of molecular structure on the bulk properties exhibited by the molecules under consideration. Corwin Hansch can be considered the first pioneer in this field as his work expanded the boundaries of how these relationships were formulated. Hansch et al. (1962) initially suspected that the partition coefficient of various compounds, along with other parameters, could be used to characterize their relative biological activity. Though, his greatest contribution to the field follows in the persistent manner in which he applied this concept to develop models in drastically new and more accurate ways than ever attempted before. His realization that SAR's should not be limited to certain independent variables and fits, such as expanding from linear to parabolic models, paved the way for a successful marriage between the development of these models with various mathematical/statistical techniques (Hansch, 1969). This trend has continued into modern day approaches such that developments from various fields, including pattern recognition, machine learning, artificial intelligence and molecular modeling etc., have been borrowed for the improvement of property/activity models. Another turning point in the development of SAR's was initiated by Kier et al. (1975) when the molecular connectivity index was introduced and shown to have strong correlations to physicochemical properties (Hall et al., 1975) as well as biological activities (Kier and Murray, 1975). This ushered in a genre of many new molecular descriptors and paved the

way for a plethora of techniques aimed at differentiating molecular structures through mathematical invariants in addition the previously used physico-chemical properties.

Regardless of the sophistication of steps taken to develop and utilize these models, the process can be broken into three distinct phases: 1) calculating molecular descriptors for structures in the training set 2) choosing the most informative molecular descriptors and 3) utilizing the chosen descriptors as independent variables to create a mapping into property/activity space. This approach is visualized in Fig. (2.5) (Dudek et al., 2006).

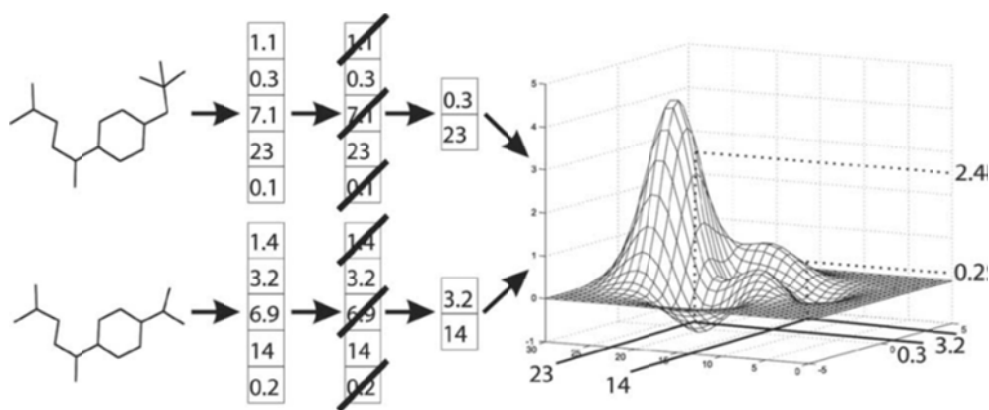


Figure 2.5 - Visualization of SAR approach.

2.4.1 Variable Selection Techniques

The automated selection of descriptor variables for use in a property model can ultimately fall into two categories (Guyon and Elisseeff, 2003). One technique, the wrapper approach, involves the identification of an optimal subset of descriptors based on the creation and ranking of a series of models. The other, known as filtering, does not construct models in the selection process as features are evaluated using other criteria. This is a necessary step in the development of most structure activity relationships due to the large number of descriptors available for correlation with the result of interest.

The feature reduction technique applied by Merkwirth et al. (2004) utilizes the principle of clustering, whereby variables are pooled into clusters based on similarity. They begin by initially removing all constant and low-entropy variables, and variables are then divided into clusters in which the absolute value of the pairwise correlation coefficients exceeds 0.98. The next step is to discard all variables, except a randomly chosen one, from each of the developed clusters. This approach is beneficial when the number of input variables is large compared to the number of observations. In addition, many machine learning techniques have a larger time complexity than linear in the number of observations and/or input variables which prohibits the consideration of large data sets. The next step is a forward stagewise selection procedure, which is a greedy-type algorithm that iteratively constructs a subset of relevant variables. The approach begins by selection of a random variable from the initially reduced set. Next, the leave-one-out (LOO) error for all combinations of one of the remaining variables with the variables in the current subset is calculated. The variable which improves the LOO error the most is selected and the process is repeated until either a predefined number of variables have been selected or there is no further improvement in the calculated error. (Merkwirth, 2004)

In a study by Venkatraman et al. (2004), the use of information-theoretic approaches based on the concept of mutual information gain has been applied to identify an optimal subset of descriptors for further correlation with a given biological activity. Since mutual information is a nonlinear statistical criterion, it is able to measure the interdependence of random variables without relying on established assumptions about their underlying relationships. This approach relies on two heuristic criteria during feature selection,

namely: (1) Feature should be comparatively informative about the output and (2) Feature should not be strongly dependent on other features selected. The measure of mutual information between two random variables A and B represents the amount of information about A contained in B and vice versa. When the random variables are independent of each other, the mutual information, defined in Eq.(2.21), is zero. The marginal probabilities for the two features are represented by $P(a)$ and $P(b)$,

$$I(A, B) = \sum_{a,b} P(a, b) \log \frac{P(a, b)}{P(a)P(b)} \quad (2.21)$$

while $P(a, b)$ gives the joint probability. Mutual information measures the distance between the joint probability and the joint probability under the assumption of independence, $P(a)P(b)$. This technique is most suitable to problems where both descriptors and activities are categorical. In such a case where the continuous numerical variables are utilized, discretization schemes must be applied to approximate the variables.

The above techniques are examples of filtering methods for variable selection. This approach is quite useful for the selection of variables with decreased interdependence (i.e. colinearity) while maintaining a strong correlation with the property or activity of interest. However, a more rigorous approach for model development, known as the wrapper method, exists. These descriptor selection techniques operate in conjunction with a mapping algorithm. One prominent technique, known as Genetic Algorithm (Siedlecki and Sklansky, 1988), stands out for this approach and is an efficient method for sampling large descriptor spaces. Genetic algorithm mimics the process of natural evolution

whereby a population is guided towards a higher degree of fitness, as often measured by the error of the model generated, through operations of mutation and crossover. Each member of the population is represented by a chromosome, within which each position usually corresponds to the absence or presence of a specific variable through the binary notation. Individual chromosomes with an increased measure of fitness, typically measured by the prediction capabilities of the model resulting from the descriptors represented within the chromosome, are selected for the conventional operations of crossover and mutation. Mutation typically involved the change of binary variables within the chromosome to either a 0 or 1, the opposite of its initial state; and crossover involves the selection of two chromosomes which are cut and recombined at one (single-point crossover) or more points. However, the success of a GA relies on the careful tuning of several probability parameters such that the solution space can be effectively explored and early convergence to a homogenous population, occupying a local minimum, is not met.

The genetic algorithm falls into a category known as ‘stochastic programming’, in which several successful techniques have been developed for the solution of problems with large, multivariate solution spaces. Another similar technique for variable selection is known as simulated annealing (SA) (Kirkpatrick et al., 1983), which is also a stochastic technique, has had great success in QSAR development (Sutter et al., 1995; Itskowitz and Tropsha, 2005). SA was inspired by the physical process of annealing in metallurgy, which involves the heating and cooling of a material to increase the size of its crystals and reduce their defects. The rate at which a material is cooled will affect the decrease in free energy associated with the underlying crystals, which also affects their

size and purity. The effect of slow cooling within the SA algorithm allows for a slow decrease in the probability of accepting worse solutions as it explores the solution space. With such an approach, the algorithm is initially allowed to move more freely around the solution space to avoid being trapped in local minima. As the algorithm proceeds it has an increased “strictness” for accepting new solutions. Just as in the genetic algorithm approach, SA aims to minimize the error of a resultant model by iteratively changing the subset of selected descriptors. In this case, some percentage of features (e.g. descriptors) is exchanged for others and this new subset is tested for its ability to model the desired output. The decision of whether or not to utilize the newly chosen set of descriptors depends on a probability function based on the Boltzmann distribution. The power of the SA method stems from altering the temperature term in the Boltzmann distribution. As the algorithm proceeds, the temperature is decreased so that the acceptance of worse solutions becomes less likely. This often results in the identification very high quality solutions to the problem at hand.

The two previously mentioned approaches of GA and SA were stochastic in nature. In contrast, there are several deterministic approaches which more thoroughly explore the descriptor space under consideration. Forward Feature Selection (Kittler, 1978) is one such technique and begins with identification of a single feature that leads to the best prediction. Features are subsequently added to the current subset and the errors associated with each model are quantified. The feature which results in the lowest error for the resultant model is selected to be included in the current subset and the process ends when a specified number of features have been identified. Sequential Backward Feature Elimination (Kittler, 1978) could be considered the inverse of this approach. In this

approach, the full set of features is used as a starting point and all subsets of features resulting from removal of a single feature are analyzed for error. The feature that leads to a model with the highest error is removed from the current subset, which is initially large and decreases in size as the algorithm proceeds. The algorithm eventually ends when the specified number of features has been eliminated. This approach, while much more computationally vigorous, often leads to better models than its counterpart, Forward Feature Selection.

2.4.2 Mapping Descriptors into Attribute Space

Once a set of descriptors has been decided upon, the next step is to create a mapping between the activity/property of interest and the descriptor values. The variety of mapping methods available can be initially categorized based on whether a linear or non-linear relationship is created. Another distinction can be made based upon the nature of the property/activity; when this value is a continuous variable, a regression must be done, whereas when the result is a category (e.g. active or inactive) this is known as a classification problem. In a regression, the dependent variable is modeled as a function of the molecular descriptors. In a classification scheme, the resulting model is defined by a decision boundary, which separates the various classes within the descriptor space.

Linear models are usually sufficient for creating activity relationships for a dataset of similar compounds. They have the benefit of being much easier to develop and interpret when compared to other methods. The most common technique for the creation of a linear property model has been Multiple Linear Regression (MLR). This approach models the predicted response, Y , by means of a set of descriptor variables, X , through the relationship shown in Eq.(2.22).

$$Y_{M \times L} = X_{M \times K} \cdot \beta_{K \times L} + E_{M \times L} \quad (2.22)$$

where, M = the number of rows of sample readings of observations

L = the number of columns of measured response properties

K = the number of columns of descriptor variables

β = the regression coefficients or sensitivities matrix

E = the error or residual matrix

There have been three cases, as described by Geladi and Kowalski (1986), for the solution of β in Eq.(2.19):

1. $K > M$: There is no unique solution for β as infinite numbers of solutions exist, unless one deletes predictor variables.
2. $K = M$: There is one unique solution provided that X has full rank.
3. $K < M$: There is no exact solution for β , however, a solution can be achieved by minimizing the residual in the following equation:

$$E = Y - X \cdot \beta$$

The most popular technique, known as the ordinary least-square (OLS) method, identifies the regression coefficients by maximizing the model sum of squares and minimizing the residual sum of squares. Using this approach, β can be estimated by:

$$\hat{\beta} = (X^T \cdot X)^{-1} \cdot X^T \cdot Y \quad (2.23)$$

where the superscript T symbolizes the transpose of a matrix.

When the number of X -variables, or descriptors, is large compared to the number of observations, this can lead to a singular ($X^T X$) matrix whose inverse does not exist. This

happens when the number of unknown variables is greater than the number of equations, leading to an underdetermined equation system which has an infinite number of solutions for β . One solution to this problem would be to apply various variable selection techniques. In addition, multivariate projection methods like PCA (principal component analysis) and PLS (partial least squares) can also be utilized to overcome such a difficulty.

PLS and PCA are methods suitable for overcoming problems in MLR associated with multicollinear or over-abundant descriptors. These techniques compress a large quantity of data and extract the information by projecting them into a low-dimensional subspace that summarizes the most relevant information (Wold et al., 1996; MacGregor et al., 1995). When the data set used is highly dimensional and very noisy with a small number of samples, PCA is an appropriate method for dimensionality reduction after which the regression model can be developed with the new latent variables through principal component regression (PCR). Prior to PCA, the data often needs to be pre-treated through a variety of techniques such that it becomes more suitable for further analysis. It is common practice to initially mean-center and scale the property variables, which is visually represented in Figure 2.6 (Eriksson et al., 2006).

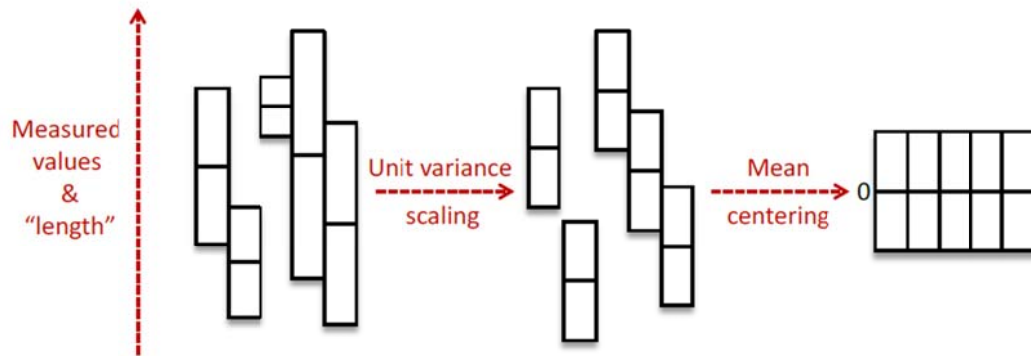


Figure 2.6 - Mean centering and scaling in PCA.

This technique ensures that no variable is allowed to dominate, in its interpreted importance, over another because of an increased length (difference in highest and lowest values) or mean value. Once the data has been prepared, or pre-treated, for further analysis, the PCA process then calculates a set of principal components (PCs) by transforming the original, correlated, variables into a new set of uncorrelated ones. The first PC is the linear combination of the standardized original variables that have the greatest possible variance and each subsequent PC is a linear combination of the standardized original variables that have the greatest possible variance, while being orthogonal to and having zero correlation with all previously defined PC's. This orthogonality constraint ensures that each variance-based axis is independent. Typically, the first three PCs capture most of the variance seen in the original data set (around 80-90%). Figure 2.7 helps visualize the dimensionality reduction achieved through PCA.

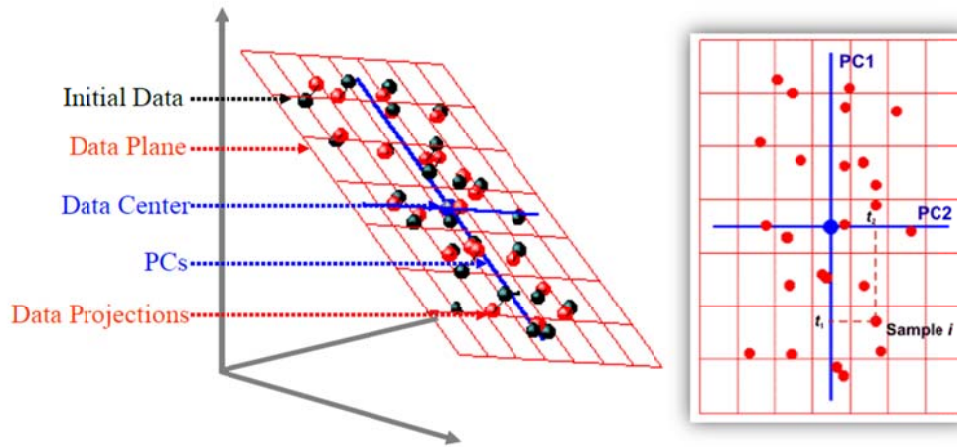


Figure 2.7 - Dimensionality reduction achieved with PCA.

The loading matrix contains the coefficients in the linear combination of the original variables defining the PCs. This can be mathematically represented as shown in Eq.(2.24).

$$\mathbf{X}_{M \times K} = t_1 \cdot p_1^T + t_2 \cdot p_2^T + \dots + t_K \cdot p_K^T = \mathbf{T}_{M \times K} \cdot \mathbf{P}_{K \times K}^T \quad (2.24)$$

Where, \mathbf{T} = the score matrix with mutually orthonormal columns

\mathbf{P} = the loading matrix with mutually orthonormal columns

PLS is a regression extension of principal component analysis and it generalizes and combines different features from both PCA and multiple linear regressions (MLR). In addition to relating the two data matrices, of descriptors and response variables, PLS also models the common structure between them which often provides better results than those obtained with the traditional multiple regression approach. Figure 8 provides a

visualization of the PLS process whereby two “PCA-like” models are created for both the descriptor and response information which are then connected through an inner relationship to provide the PLS model.

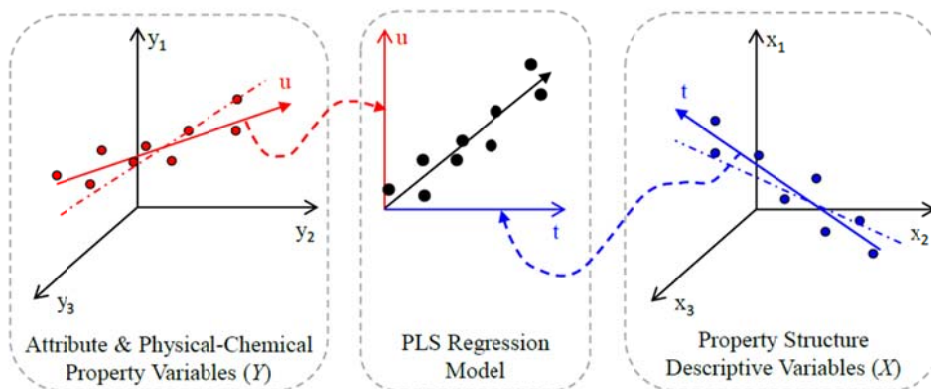


Figure 2.8 - PLS regression visualization.

The score plot, which is a two-dimensional representation of the data, plots the second principal component score against the first principal component score and can be used to identify clusters and unusual observations in the dataset. The score plot in Figure 2.8 shows a linear relationship between predictors (x) and responses (y), however, there may be non-linearities. This plot can also be used to rank the observations according to each of the principal component scores. The dashed-dot line seen in the outer pictures of Figure 8 represents the projection if PCA were performed on X and Y individually.

Both techniques of PCR and PLS aim to avoid collinearity problems which would allow one to work with a number of variables that is greater than the number of samples. A comparison of the two techniques (Wentzell and Vega Montoto, 2003) has revealed similar prediction capabilities, however, PCR tends to yield higher precision (degree of

closeness of the measured values to each other) while PLS yields higher accuracy (degree of closeness of a measured value to the actual value).

Support vector machines (SVM's) (Cortes and Vladimir, 1995) are one type of supervised learning model with an associated learning algorithm that can analyze data and recognize patterns. This technique can be used in both classification and regression analysis, both linear and nonlinear, and as such is very flexible. An SVM constructs a hyperplane, or set of hyperplanes, in a highly dimensional space such that the distance to the nearest training data point is maximized. Often times, the data set might not be linearly separable and the option of mapping the original finite-dimensional space into a much higher-dimensional space is made possible through the use of kernel functions. These functions lower the computational load associated with moving between the two mapped spaces by ensuring that dot products are easily computed in terms of the original variable space. Also, slack variables are introduced and they are subject to optimization to allow for a better fit than linear approaches in many cases. Even though erroneous classification cannot be avoided, it is penalized and the misclassified compounds become support vectors themselves. By training a linear classifier in kernel space a classifier, which is nonlinear with respect to descriptor space, is obtained. SVM methods have also been extended to handle regression type problems (Smola and Scholkopf, 2004). Contrary to typical regression methods, however, the predicted values are penalized only if their absolute error exceeds a certain user-specified threshold. Thus, the regression model developed by these means is not optimal in terms of the least-square error.

Decision trees (Quinlan, 1986), another type of non-linear mapping technique available for the development of structure-activity (property) relationships, differ from

most algorithms by their connection to logic-based and expert systems. This model consists of a tree-like structure containing the conventional nodes and links. Nodes form a hierarchical pattern, with several child nodes stemming from a common parent node. A node with no children is referred to as a leaf. Each node typically refers to a specific descriptor and a test is made such that the results of the test direct the algorithm to a specific child node. This continues towards the leaves of the tree and the final decision is based on the activity class associated with that leaf. The diagram in Figure 2.9 represents the classification of a compound, based on three descriptors, as being either active or inactive.

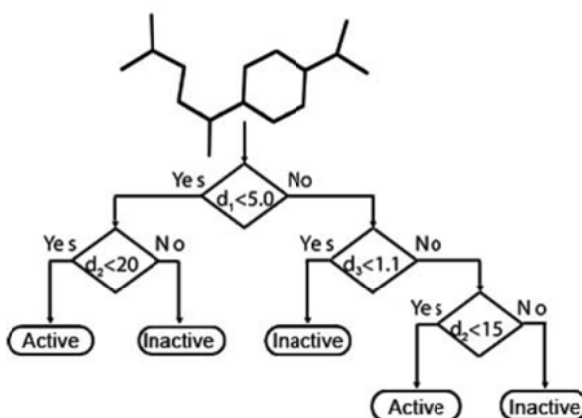


Figure 2.9 - Classification through the use of a decision tree.

Training a decision tree model begins with choosing the test for the root node. This test is chosen on the basis of its ability to categorize compounds most effectively into their activity classes. If the initial root node test is able to correctly classify all compounds then the tree is finalized, however, this is most often not the case. As such, several more tests are made, spanning out from the original node, in an iterative manner

creating several more nodes. Each node is considered for classification as a leaf when the majority (ideally all) of compounds passing through that testing route is correctly classified. One important decision that needs to be made in the development of these trees is which test should be introduced at which point. This can be restated as ‘which descriptor would provide the best discrimination criteria at this point’, and in this case descriptor ranking is typically applied. Once this descriptor, or test, has been decided upon, it is next necessary to introduce a decision rule that separates the compounds into various activity classes. Decision tree methods typically lead to suboptimal error rates when compared to other non-linear methods, most likely due to the reliance on a single feature in each node. While the conventional application of decision trees is for classification type problems, they can also handle regression problems (Breiman et al., 1984) by associating each leaf with a numerical value instead of the categorical class. This methodology has been tested in a study (Svetnik et al., 2005) on a wide range of targets, including COX-2 inhibition, blood-brain barrier permeability, CDK-2 antagonist activity, dopamine binding affinity, logD and toxicity. While they were outperformed by support vector machines and ensembles of decision trees, they did often perform better than PLS or naive bayes classifier, which is a simple probabilistic classifier based on application of Bayes’ theorem with strong independence assumptions between the features.

2.5 Fragment Based Property Models

In reference to Figure 2.10, it can be seen that there is a link between property space and chemical space, seen on the left, through the use of molecular descriptors. It is of special interest when the intermediate variables that link these two spaces have some structural

reference. This significantly aids in the identification of solutions during the application of inverse property prediction, which is one technique in the field of molecular design. This technique will be covered in detail in section 2.6, however, its consideration here is useful for the introduction of fragment based property models.

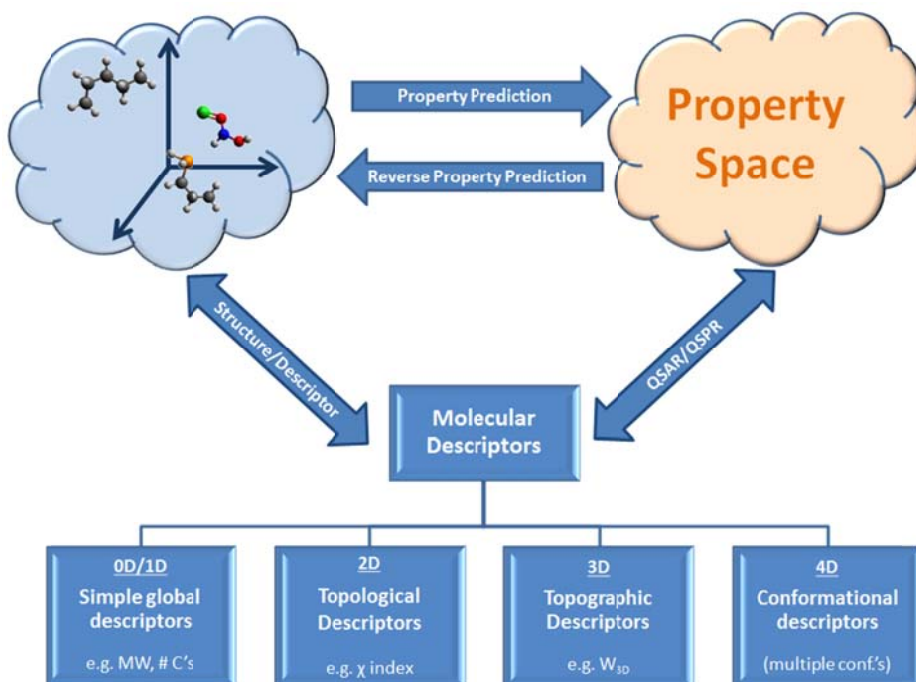


Figure 2.10 - Overview of inverse property prediction.

Often times, the molecular design problem is solved in descriptor space. This means that the resulting solutions will be in terms of these descriptors. Since the ultimate goal is to generate a structure, or set of structures, with an increased likelihood of having the desired attributes, this means that these solutions will have to be translated into chemical space. Therein lies one of the more difficult sub-problems associated with molecular design. However, when these molecular descriptors have a structural reference this aids in the generation of these solutions. This allows one to assemble the possible molecular

structures, subject to structural constraints, from these fragments as if they were building blocks. Several techniques taking advantage of this concept will be discussed in the sections below.

2.5.1 Group Contribution Method

One of the most widely used property prediction techniques, which has had widespread success in modeling and predicting a plethora of properties, is known as the group contribution method (Joback and Reid, 1983; Constantinou and Gani, 1994). This technique is based on the concept that the summation of contributions from various functional groups, or structural fragments, can account for the property of a molecule. This allows one to develop and train a model on an existing data set, which can then be used to predict the properties of molecules not in the original training set. This is a very attractive idea since it is unlikely that we will ever synthesize and characterize even a fraction of the accessible chemical space in the near future. The simplest form of group contribution method is the determination of a component property by simply summing up the group contributions as in the example for boiling point shown in Eq.(2.25).

$$T_B = 198.2 + \sum G_i \quad (2.25)$$

Eq.(2.25) takes on a linear form where G_i represents the summation of contributions from each group i . These contributions can be any positive or negative real number. This approach works well in a limited range of components but, unfortunately, leads to large errors when applied outside its applicability domain. The smallest structural fragment represented within the group contribution methodology is known as a first-order group. In addition to containing first-order groups, a group contribution property model can also

contain higher order contributions which can improve the overall accuracy of the model. Second order groups contain collections of first order groups and capture the interactions between those groups. Examples of first and second order groups are shown for several molecules in Figure 2.11.

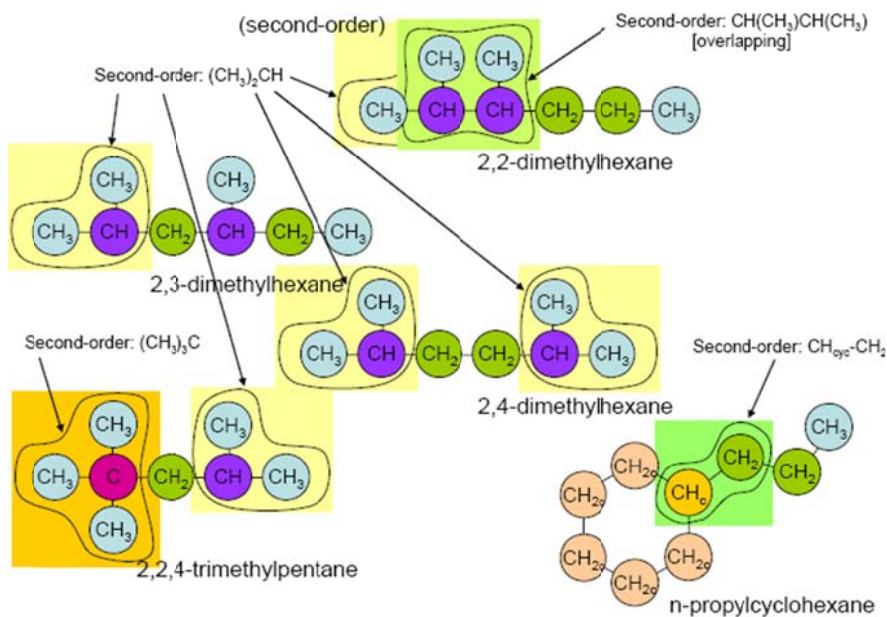


Figure 2.11 – Examples of first and second order groups

Eq.(2.26) represents a generalized group contribution model which accounts for higher order groups.

$$f(x) = \sum_i N_i C_i + \sum_j M_j D_j + \sum_k O_k E_k \quad (2.26)$$

where, C_i = the contribution for first-order group i

N_i = the number of occurrences of first-order group i

D_j = the contribution from second-order group j

M_j = the number of occurrences of second-order group j

E_k = the contribution from third-order group k

O_k = the number of occurrences of third-order group k

In the above formulation, second order groups can be estimated from first order groups and correct for the interactions between first order groups. Also, third order groups can be derived in a similar manner and will help to correct for poly-functional compounds with more than four carbon atoms in the main chain. In addition to introducing higher order groups for improved accuracy, there are also group interaction parameters available. This technique is useful when a simple additive method is not sufficient to capture the property of interest; however, it requires many more model parameters to be enumerated in exchange for this accuracy. One example of such an approach is known as the UNIFAC (UNIQUAC Functional-group Activity Coefficients) method, which estimates activity coefficients (Fredenslund et al., 1975).

There are specific techniques available for the enumeration of structures matching a set of structural fragments. One such approach by Constantinou et al. (1996) treats the initial set of first order groups as a vector in what could be considered a 'fragment-space'. The feasibility of this collection of fragments towards generating a complete molecular structure is tested against graph-theoretical rules based on chemistry concepts. If determined to be a feasible structure, the vector refers to at least one structural isomer and there are often several which need to be enumerated. This concept is known as degeneracy, and while it is usually undesirable in molecular descriptors it is often quite

necessary to generate a model with some degree of conciseness and transferability. This problem becomes much more difficult with the consideration of higher order groups.

2.5.2 Pharmacophore Models

A pharmacophore model can be considered a fragment based model as it often specifies a set of structural features necessary for a certain biological activity. The IUPAC definition of a pharmacophore is “an ensemble of steric and electronic features that is necessary to ensure the optimal supramolecular interactions with a specific biological target and to trigger (or block) its biological response (Wermuth et al., 1998).” These models provide information on how a set of structurally diverse ligands can ultimately have the same effect on a given receptor. Additionally, pharmacophore models are extremely useful in the identification of novel compounds with an increased likelihood of exhibiting the desired binding characteristics. Some typical pharmacophore features include hydrophobic centroids, aromatic rings, hydrogen bond acceptors or donors, cations and anions. These points may be located on the ligand itself or sometimes projected to points around the ligand. The steps necessary to develop a pharmacophore, such as the one shown in Figure 2.12 are: 1) Select a set of molecules with varying bioactivity; 2) Perform a conformational analysis and identify the most likely bioactive conformations; 3) Superimpose the chosen conformers; 4) Develop an abstract representation of the most common features identified during the superimpose step; 5) Validate the chosen model on a chosen set of molecules with known bioactivity. Often times a three-dimensional model can be developed by calculating the optimal distances between each of the identified pharmacophoric features as shown in Figure 2.12(A). Figure 2.12(B)

represents an overlay of a given pharmacophore model with a structural representation of each point of interest shown in an example molecule.

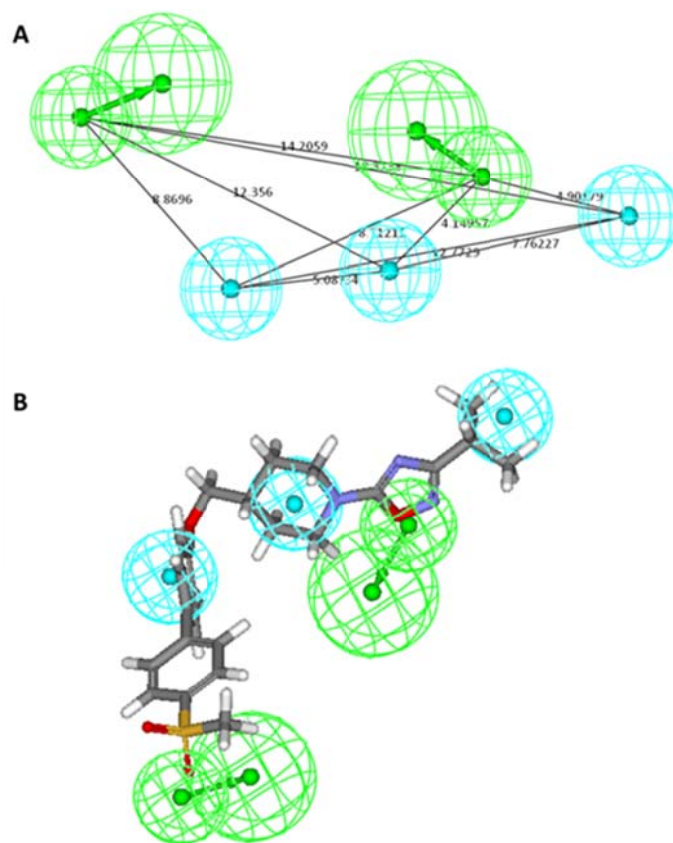


Figure 2.12 - Examples of a pharmacophore model.

2.5.3 Signature Descriptor

The signature descriptor, originally proposed by Visco et al. (2002, 2003) is a fragment based descriptor which encodes the environment around a central atom up to a pre-defined height, h . In the context of chemical graph theory (Trinajstić, 1983), with atoms being nodes and bonds being edges, an atomic signature descriptor can be seen as a subgraph of its complete parent graph. An example of this is shown below in Figure 2.13.

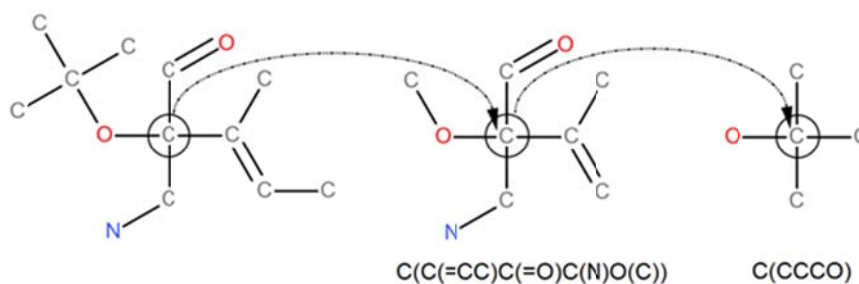


Figure 2.13 - Example of atomic signature descriptors of varying height.

In Figure 2.13, the height two atomic signature of the carbon atom encircled on the left is represented by the graph shown in the middle. In addition, the string based representation of this atomic signature is shown below its respective graph. The height one atomic signature for this carbon atom is seen on the far right, with its atomic signature shown below as well. Each atom within a molecule would have its own atomic signature and the summation of these atomic signatures would represent the molecular signature. This is mathematically represented in Eq.(2.27) where ${}^h\sigma_G({}^hX_i)$ is a base vector, ${}^h\alpha_i$ is the number of atoms having the signature of each fragment represented within the base vector, hK_G is the number of base vectors and ${}^h\sigma_G$ represents the molecular signature.

$${}^h\sigma(G) = \sum_{x \in V_G} {}^h\sigma_G(x) = \sum_{i=1}^{{}^hK_G} {}^h\alpha_i {}^h\sigma_G({}^hX_i) \quad (2.27)$$

The systematic procedure for the construction of an atomic signature was developed by Visco et al. (2002) and is explained below:

1. For the atom x for which the atomic graph is constructed, all the atoms and bonds will be shown up to the height h in the subgraph, ${}^hG(x)$.

2. All the atoms (vertices) in the graph are labeled in a canonical order starting with atom x .
3. Construct the tree that spans over all edges in the subgraph. The root of the tree is the atom x itself. The tree is constructed one layer at a time up to level h . It is possible to have one vertex more than once in the graph. However, no edge should be repeated in the same graph.
4. After constructing the signature tree, all the canonical labels that appear only once in the graph are to be removed and the repeating labels are to be re-labeled in the order that they appear in the graph.
5. The signature can be written by reading the tree from the atom x . The vertex color must be enclosed in a parenthesis in each level. For a vertex that appears more than once, the vertex labels should also be included in the parenthesis.

One example for the construction of atomic signatures is shown in Figure 2.13. Here the stepwise procedure for obtaining the atomic signature of atom x up to height three in ethyl benzene is illustrated. In the first step, all the atoms at distance 3 from atom x are extracted. In the second step, the subgraph is canonized with atom x having label 1. In the third step, a tree spanning all the edges is constructed up to height 3 from the subgraph. In the fourth step, the labels that appear only once in the tree are removed and the rest of the labels are renumbered in the order of their appearance. In the final step, the signature is generated in the required orders from the tree by starting from atom x (Chemmanattuvalappil, 2008).

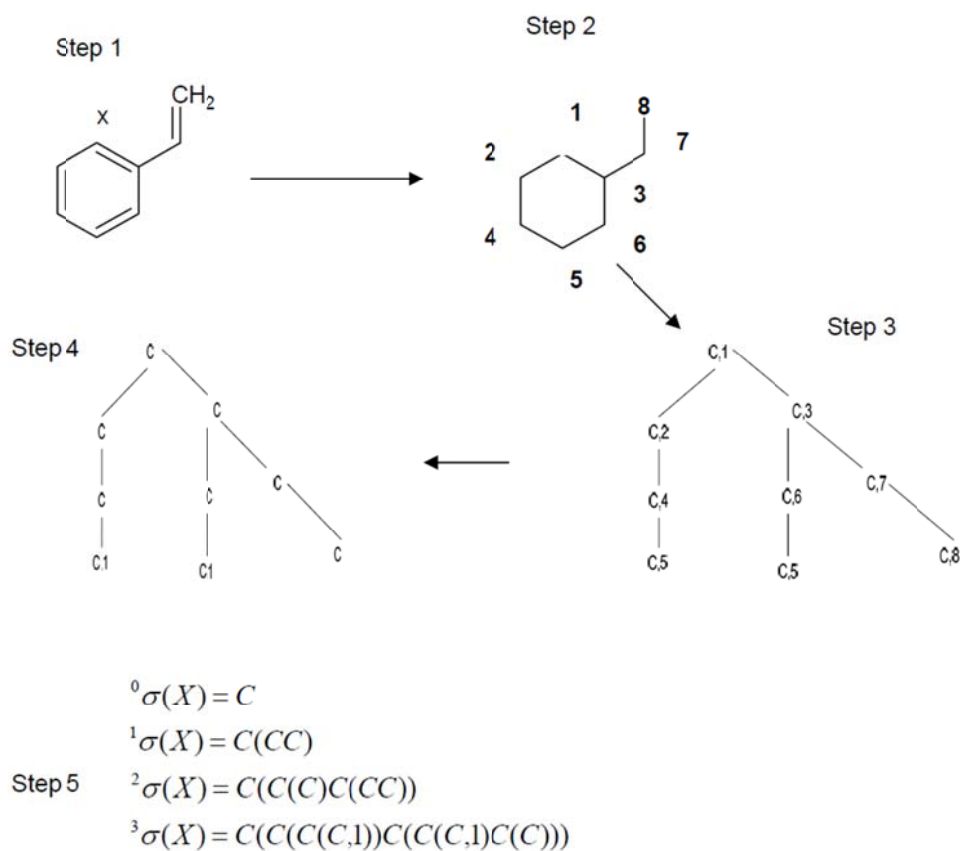


Figure 2.14 - Development of atomic signature up to height 3.

An example for the determination of a molecular signature is illustrated in Figure 2.15 for heights ranging from zero to three. In this case the vertex elements are colored using a Dreiding force field (Mayo et al., 1990). Dreiding is a generic force field with parameters restricted to simple rules which enable the prediction of novel combinations of atoms. Within this force field, atoms are represented by five characters, the first two of which stand for the atom type. The third character represents hybridization, the fourth represents the number of implicit hydrogen atoms and the fifth character represents the alternate characteristics of the atom such as formal oxidation state. After coloring the atoms in the graph, a tree spanning all edges is generated.

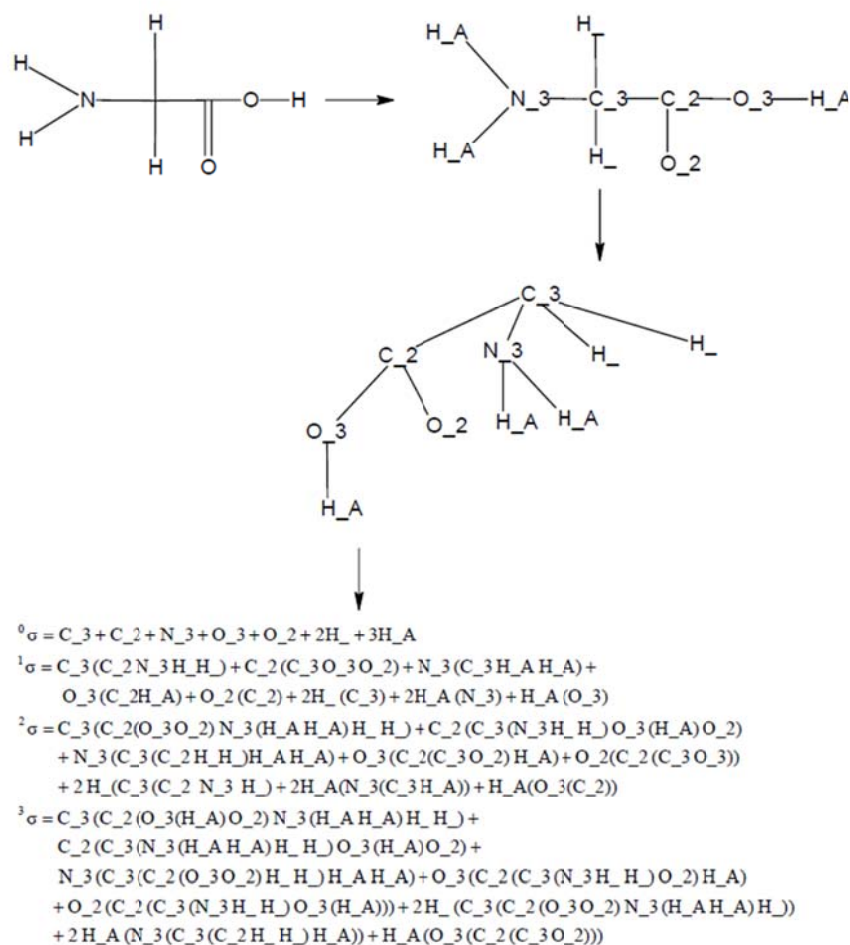


Figure 2.15 - Example of height three molecular signature enumeration.

The signature descriptor has proven its usefulness in the development of property models for a variety of properties and activities. The first test conducted was based on the utilization two distinct data sets (Faulon, 2003). One set came from a biological source and represented measurements of activity for 121 compounds used as HIV-1 inhibitors. The second, much larger set, was for the octanol/water partition coefficient, of 12,865 compounds. These two studies utilized height-0, height-1 and height-2 signatures in an attempt to create a model with characteristics comparable to those made using

conventional descriptors, which were available within the Molconn-Z software (Hall et al., 1991). All models were developed to follow a linear form, and parametrization was performed using the forward-stepping regression technique of multiple linear regression. The 121 HIV-1 protease inhibitors had experimental activity reported in units of pIC₅₀, representing the negative logarithm half maximal inhibitory concentration, which spanned 7 orders of magnitude. Nine of these compounds were chosen as test set compounds and were not utilized in the creation of any models, but were used to evaluate the predictive ability of each model. For this case, the model developed using height-2 signatures was best able to capture the phenomena with a training set R² of 1.0. For the log(P), which represents the logarithm of the experimentally measured octanol/water partition coefficient, model development, 123 of the original data set were left of for model validation and the rest were utilized for model training. In this study, the signature descriptor of height-1 was able to outperform all other property models based on other descriptors.

In addition to being useful for developing accurate and predictive property models, the signature descriptor is also ideal for application in the inverse property model approach to molecular design. The ability of the signature descriptor to define most any topological index has been shown (Faulon, 2003). In addition, the signature descriptor has been proven to have low degeneracy when enumerating solutions. The combination of these features with signature's link to structural fragments is what makes it ideal for molecular design applications. Specific methodologies for its use in molecular design studies will be covered in detail in section 2.6.

2.6 Solution Techniques in CAMD

Over the decades there have been many methodologies developed for the solution of molecular design problems. The particular technique applied is strongly dependent on the nature of information available for the problem at hand. For instance, does one of the property models utilized have a non-linear relationship or perhaps it is in the form of a neural network. Often times, an increase in model complexity will bring about an increase in computational complexity which is not commensurate to the increase in accuracy or differentiation of potential solutions. In any manner, techniques have been developed that are capable of handling many of the problems encountered within this field. These approaches have been broadly categorized as database searches, generate-and-test and programming/optimization. Each approach has its own strengths and weaknesses, some of which will be covered in the following sections.

2.6.1 Database Search

The most straightforward technique for computer-aided molecular design is known as a database search. This approach, which is usually quite fast, consists of testing each molecule within a database against certain criteria while identifying structures with the desired characteristics. These criteria can be a wide range of things such as adherence to given property models with varying descriptors or it could be a molecular similarity search, which could also be based on descriptors or molecular fragments. The main limitation to this approach is that it cannot consider molecular which are not available within the database; however, it does not require the often computationally intensive step of structure generation and as such is usually much faster.

Similarity searching has been one of the most widely applied techniques for identifying potential drug candidates with a database of existing molecules. This approach typically describes the molecules encountered with descriptors which capture the underlying nature of the structure. The molecules considered are typically compared to one structure with a known, usually very high, activity such that those with a high measure of similarity are hoped to also have a desirable activity. One such study by Chen and Reynolds (2002) assesses the effectiveness of utilizing various sets of 2D linear fragment descriptors along with varying measures of comparison. They focused their methods on two large public databases, one of which was NCI anti-AIDS (NCI Developmental Therapeutics Program) and the other MDDR (MDL Information Systems Inc). The underlying assumption with these techniques, and most of molecular design in general, is known as the similar property principle, which states that similar chemical structures should lead to similar physicochemical properties and biological activities. The descriptors used as a measure of similarity, and the technique used for comparison of these descriptors, are what make these approaches so different. Four sets of 2D linear fragment descriptors, based on the original definition of atom pairs and atom sequences, were used in this study as were three forms of the Tanimoto coefficient and the Euclidean distance. The Tanimoto coefficient is a distance measure, or more conveniently a measure of similarity, between two molecules structures. When each molecular structure is represented as a vector of k dimensions, with each dimension representing the occurrence of a particular molecular feature, the Tanimoto coefficient between two points, a and b , is shown in Eq.(2.28). The influence of these structural descriptors and similarity coefficients on the effectiveness of retrieving active structural analogues was

systematically studied. The Euclidean distance is another type of measure which is similar to the Tanimoto coefficient and its calculation, between vectors **a** and **b**, is shown in Eq.(2.29), where *n* represents the dimensionality of each vector.

$$\frac{\sum_{j=1}^k a_j \times b_j}{\sum_{j=1}^k a_j^2 + \sum_{j=1}^k b_j^2 - \sum_{j=1}^k a_j \times b_j} \quad (2.28)$$

$$d(a, b) = d(b, a) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2} \quad (2.29)$$

Of the four structural descriptors utilized in this study, the first is known as MACCS keys (Molecular Design Ltd.), which are a set of questions about a chemical structure. These questions determine the nature of the underlying structure and produce a list of binary values by which that structure is described. The second is represented by Daylight fingerprints (Daylight Chemical Information, Inc.), which enumerate all linear chemical substructures of a predefined range of lengths. These fragments are typically hashed/folded into a bit string with the length fixed to save memory space. This, however, often leads to a loss of certain structural information and introduces additional noise into subsequent calculations. The third, exemplified by the work of Carhart et al. (1985) is very similar to the Daylight fingerprints however no information is compressed. The similarity coefficients, which would be derived as a function of the above mentioned descriptors for pairs of atoms, can be divided into two major classes: association and

distance coefficients. The main difference between these two is that distance coefficients consider the absence of certain structural features as evidence of similarity whereas association coefficients do not. The various forms of coefficients utilized in this study are summarized in Table 2.1.

Table 2.1 - Definitions of Tanimoto Coefficient and Euclidean Distance.

	Tanimoto coefficient	Euclidean distance
Binary form	$S_{A,B} = \frac{c}{a + b - c}$	$D_{A,B} = [a + b - 2c]^{1/2}$
Algebraic form	$S_{A,B} = \frac{\sum_{i=1}^m n_{A,i}n_{B,i}}{\sum_{i=1}^m n_{A,i}^2 + \sum_{i=1}^m n_{B,i}^2 - \sum_{i=1}^m n_{A,i}n_{B,i}}$	$D_{A,B} = [\sum_{i=1}^m (n_{A,i} - n_{B,i})^2]^{1/2}$
Set-theoretic form	$S_{A,B} = \frac{\sum_{i=1}^m \min(n_{A,i}, n_{B,i})}{\sum_{i=1}^m n_{A,i} + \sum_{i=1}^m n_{B,i} - \sum_{i=1}^m \min(n_{A,i}, n_{B,i})}$	$D_{A,B} = \sum_{i=1}^m n_{A,i} - n_{B,i} $

In Table 2.1, a represents the number of unique fragments in compound A, b represents the number of unique fragments in compound B and c represents the number of unique fragments shared by both compounds A and B. Also, $n_{A,i}$ is the number of fragment i in compound A, and $n_{B,i}$ is the number of fragment i in compound B.

It was found that the Tanimoto coefficient gave considerably better results than the Euclidean distance for both data sets. This is interesting because it reveals that the presence of certain structural fragments is a better measure of similarity than the combined consideration of presence and absence of these same features. The major difference in these two methods lies in their ability to distinguish moderately similar structures. It was also found that the binary form of storing the structural information did not account for much memory saving when compared to the increase in discrimination

power seen in including information about the number of each fragment, as done in a set-theoretic approach. In terms of the optimal structural descriptor, it was found that a balance between “fuzzy” and “specific” descriptor types allows for an optimal identification of true hits. They noted that descriptors that were too “fuzzy”, or non-specific, tended to produce more false positives and descriptors that were too specific tended to identify more false negatives.

The previous technique was a good introduction to the potential of database searching using 2D descriptors. However, in the domain of drug design, another very important database searching approach is based on the consideration of 3D similarity. This allows for the identification of molecules that match a hypothesis of 3D requirements for bioactivity. The interest in such an approach was fueled by the availability of tools for molecular modeling and pharmacophore mapping and by the increasing numbers of 3D protein structures as targets for new drugs (Martin, 1992). There are several types of 3D structure searching and they can be differentiated by the source of 3D information used for searching, how this information is described, how the 3D requirements are established as well as the results of the search. The source of 3D information can be developed as a pharmacophore from several active molecules, a proposed bioactive conformation of a single ligand, a low-energy conformation of a ligand with desired affinity, or the actual 3D structure of the protein or DNA binding site. This information is usually defined by geometric constraints such as superposition rules of points or spheres, locations of specific functional groups, potential energies of bound ligands and CoMFA coefficients. Pepperrell et al. (1990) have explored the ability of different definition of 3D similarity to detect molecules which might have an activity

similar to the input molecule. They found that a method called atom mapping was the most effective. This method identifies atoms within a database molecule that are identical in atomic number and most similar in the intra-atomic distance profile to the atoms in the query molecule selected. They actually compared the molecules identified through a 3D similarity search with those found in a 2D similarity search and this information is shown in Figure 2.16. In this Figure, the results for a 3D similarity search, on molecule 30, are shown as 31-35, whereas the results from a 2D similarity search are shown as 36-38.

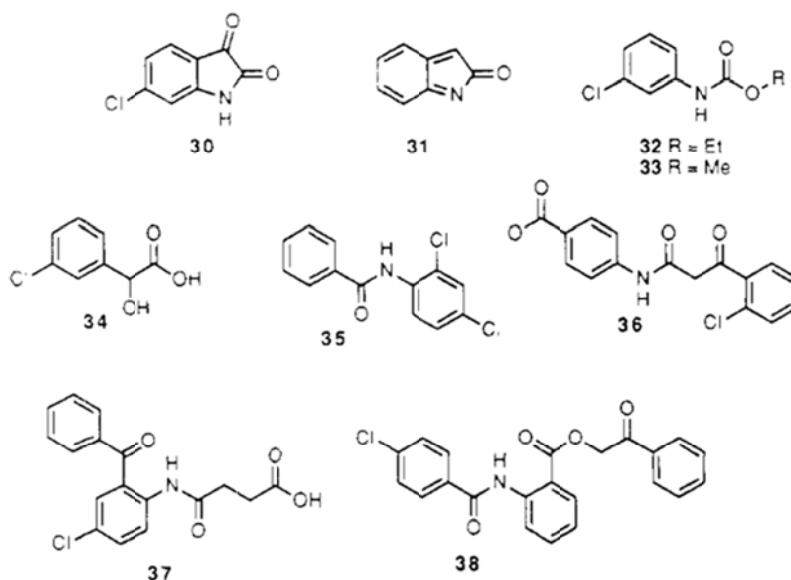


Figure 2.16 - Results of 2D and 3D similarity searches.

2.6.2 Generate and Test

The generate and test approach to molecular design relies on the ability of the algorithm to generate molecular structures for consideration as candidate molecules. This is often done by following a set of rules and combinatorially considering sets of fragments developed from a fragment library. This library can be as variant or focused as desired, however the size of said library will have a huge effect on the computational demand of

generating structures. A situation known as ‘combinatorial explosion’ is well known in combinatorial optimization and refers to the overwhelming consideration of a large number of potential solutions based on a large initial fragment library. In addition, there are deterministic and stochastic techniques for the creation of structures from fragments. Stochastic techniques provide one solution to the combinatorial explosion problem that is often encountered. One interesting observation is that the size of molecules in the fragment library has an effect on the ultimate number of unique structural isomers that are able to be created. This has been seen in the systematic development of structures from signature descriptor fragments. The smaller size fragments, approaching the limit of single atoms, ultimately create more structures, while larger fragments produce fewer structures in a deterministic manner. This approach allows for the consideration novel structures which may not have even been synthesized yet and is attractive because this structure generation can be controlled by the presence of absence of various fragments within the initial fragment library.

One example of the implementation of a generate and test CAMD approach, among many, can be found in Harper et al. (1999). This approach is unique in that it combines the conventional, and very successful, design of molecules based on the group contribution methodology with molecular modeling. Their approach, which aims at avoiding combinatorial explosion, employs a structured generate and test approach, where, every level generates and tests structures with the lower levels using molecular representation and the higher levels using atomic representations. The most time-consuming calculations will be held at the higher levels where the lowest number of acceptable structures has been passed to.

In the first level, groups, or vectors, of first order fragments are considered and the specified properties are tested using the group contribution method. Level two considers the generation of isomers from these fragment vectors and rules are applied so that only feasible chemical compounds are generated. This level allows the consideration of properties which could not be calculated at level one, thus further refining the set of potential solutions. Level three represents a highly diminished set of potential structure with implicitly described connectivity. Here, the connectivity is fully established, and specified through a connectivity matrix, and a detailed microscopic description is recognized. At the beginning of level four, for each molecule surviving the previous steps, a three-dimensional structure is generated using default bond lengths and angles. This information is directly transferred into a molecular modeling program, in which it undergoes an energy minimization calculation to identify a more stable conformation. This process is visually represented in Figure 2.17.

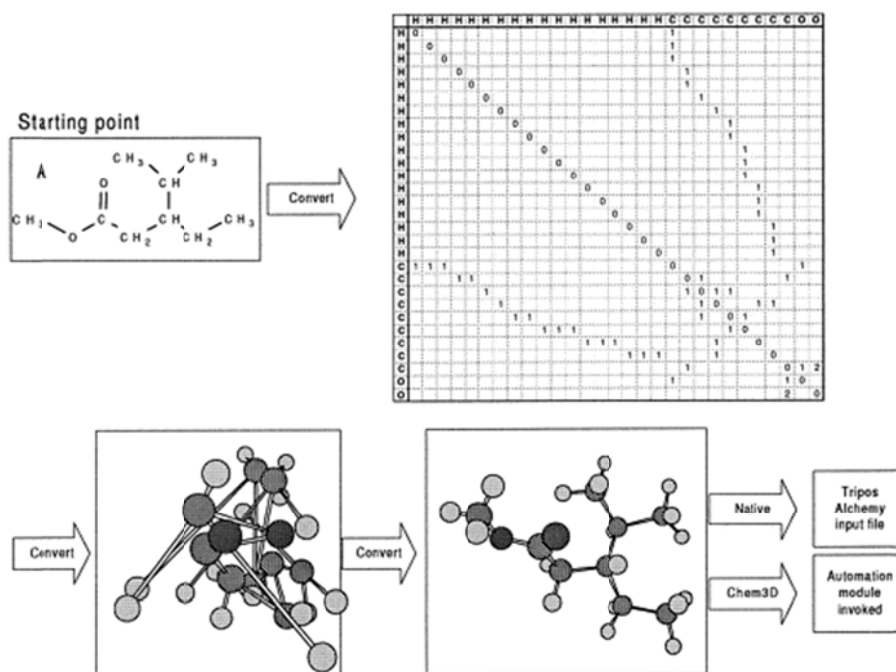


Figure 2.17 - Linkage of CAMD and molecular modeling.

This methodology was tested on two case studies, namely the identification of a replacement solvent for removal of phenol from a waste water stream and the design of an extractive distillation agent. The approach was successful in identifying feasible solutions in both cases and represents an improved methodology for the utilization of both group contribution and molecular modeling in molecular design applications.

2.6.3 Programming and Optimization

The two previous approaches to computer-aided molecular design apply property models in a forward manner. That is to say that they generate a set of descriptors for a known molecule and these descriptors are fed into the property model and the property associated with that molecule is calculated. In comparison, the reverse utilization of a QSAR, or property model, would begin with certain desirable characteristics which are

typically expressed in terms of acceptable property ranges. Using these acceptable property limits, molecules are designed by enumerating an acceptable descriptor space and subsequently calculating which structures fall within this space. The unique characteristic of this inverse approach to molecular design is that it allows for the application of programming and optimization in a very effective manner. One such approach utilizing this concept lies in a study on the application of inverse QSPR using multiobjective optimization (Brown et al., 2006). This study uses a variety of tools which are pulled together to develop a novel workflow for the solution of inverse QSPR problems. The compound generator is a genetic algorithm (GA) that operates directly on graph-based chromosomes that represent molecules in the population. New molecules are optimized by iteratively scoring, sampling and perturbing the current population of molecules, which is common in the GA approach.

The molecular descriptors used in this study are known as Fingal (Fingerprinting Algorithm) (Brown et al., 2005) and represent a molecular hash-key fingerprint which can be rapidly generated and also have proven to be highly applicable to predictive modeling. The Fingal descriptor, while also containing topological information, has geometric information which has been calculated. This geometric information provides immediate estimates on the upper and lower bounds between specific atoms in a molecule. Property models were developed using the PLS regression technique, and these models were utilized to identify properties of the newly generated structures, developed through GA. This model was also continuously updated as the algorithm ran. Pareto ranking, which determines a rank position for each candidate solution according to the number of other solutions that dominate it in all objectives, was utilized to achieve a

balanced optimization of all three objectives considered. A flowchart visualizing the steps associated with this novel approach can be seen in Figure 2.18.

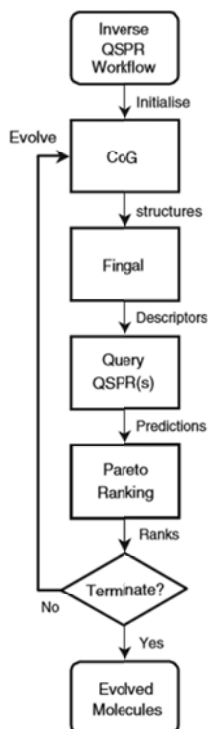


Figure 2.18 - Flowchart for inverse QSPR workflow optimization approach.

Two case-studies were conducted to analyze the effectiveness of this approach, one of which aimed to optimize the property of mean molecular polarizability and the other aqueous solubility. These studies were compared to a model developed in Dragon, using a wide variety of descriptors, as a validation technique. In both cases, the approach was able to identify solutions within the desired property range. This proved that multiple QSPR models can be effectively integrated into an inverse CAMD strategy, and the simultaneous optimization of multiple objectives appears to provide an effective means for evolving multiple novel compounds.

This previous study was a good example of a stochastic optimization approach, utilizing the genetic algorithm, towards the solution of a CAMD problem. However, when the problem can be defined in terms of linear relationships this allows for the deterministic solution of such problems in an efficient manner. This was the situation for a methodology presented by Churchwell et al. (2004) in which the signature molecular descriptor was used to identify novel potent LFA-1/ICAM-1 peptide inhibitors. The nature of signature descriptor is covered in detail in section 2.5.3, but in general it is a fragment based descriptor with low degeneracy which has proven to be useful in the inverse QSAR scheme because of its ability to generate solutions structures in an efficient manner. This study is quite useful in the introduction and development of the methodology presented in this dissertation, which is also based upon an extended version of the signature descriptor. Initially, a QSAR is developed using a forward selection procedure in multilinear regression with signature descriptors representing the independent variables. In addition to the property model developed, there are some constraint equations that are needed, as in the group contribution approach, to ensure the feasibility of structures considered. There are two types of equations within this category and they are the graphicality equation and the consistency equations. The graphicality equation is developed directly from graph theory and ensure that at least one molecular graph can be created form a set of atomic signatures. In order to build a connected graph, it is required that (1) the sum of all the vertex degrees must be even and (2) the number of vertices of odd degree must be even. The resulting equation can be expressed in terms of a degree sequence $N = \{n_1, n_2, \dots, n_k\}$ where n_i is the number of vertices of degree i . In

such a situation, the degree sequence is graphical if and only if there exists an integer $z \geq 0$ such that:

$$\sum_{i=2}^k (i-2)n_i - n_1 + 2 = 2z \quad (2.30)$$

The graphicality equation can be analyzed directly from the height zero molecular signatures, which conveniently offers the degree of each vertex, or atom, in the molecular graph. The consistency equations ensure that, since the atomic signature overlap with each other, the interdependency of each signature upon the others is mathematically feasible. Together, the property model and constraint equations represent a system of equations with unknowns corresponding to the occurrence of various atomic signatures. They represent a set of linear equations which is more specifically labeled a linear Diophantine system since the solutions are limited to positive integer values.

Once a set of solutions has been generated in terms of vectors representing a group of atomic signatures, it remains necessary to establish the various structural isomers which these vectors refer to. This study represents one of the first attempts to enumerate these isomers in a deterministic manner, which means that all of them were identified. There are two primary steps in the identification of structural isomers of a set of isolated nodes within a graph G : (1) determine the orbits or atoms with equivalent atomic signatures of G , and (2) saturate each atom of a chosen orbit. This process is repeated until all vertices have been saturated where a saturated subgraph is not generated in the process. The case study chosen to exemplify this methodology resulted in the identification of 223 compounds matching the established criteria. 14 of these were found

in the original training set and two were within the test set. 77 of these peptides identified were classified as strong inhibitors, and two of these were synthesized and are the strongest inhibiting peptides to date that work in-vivo as well. This shows the strength in such an approach, which is also computationally feasible, being represented as a set of linear equations.

3. Methodology

2D molecular descriptors have predominantly, and very successfully (Katritzky and Gordeeva, 1993), been used in the development of models describing bulk properties such as boiling point, viscosity, density etc. Their applicability in modeling molecular binding interactions, such as ligand receptor affinity, is often limited to data sets of high similarity. Thus, a natural extension to this problem was to include spatial descriptors, which have since proven useful in the design of pharmaceuticals and agrochemicals (Verma et al., 2010). The usefulness seen in this wide variety of descriptors was further applied with the advent of powerful variable selection techniques such as the ones covered in section 2.4.1. These methods have revealed that the optimal subset of descriptors for capturing the structure activity (property) relationships (SARs) for a variety of properties and biological activities includes descriptors with varying dimensionality (Kar and Roy, 2010; Nettles et al., 2006).

The inclusion of spatial, 3D and 4D, descriptors necessitates a method for capturing the conformational capabilities of candidate molecules. Although there are many methods available for conformational sampling, these are typically infeasible for the consideration of a large chemical search space, as is often desired. As a result, often only one local energy minima is supplied to a given model when used in a predictive manner. It has been shown that the actual structure of a bound ligand is often not found in the set of local minima identified through a quick conformational search of the unbound ligand (Perola and Charifson, 2004). This seems especially true when only one conformer is considered. This provides another motivation behind the development of a systematic

algorithm, proposed in this dissertation, towards the solution of molecular design problems with multi-dimensional criteria. The goal is to be able to solve problems of this nature in a computationally efficient manner, while still considering, to a reasonable degree, the conformational capabilities of candidate molecules within the defined chemical search space. The derivation of this spatial information relies on atomic scale simulations, from which the accessible conformations of each atomic signature can be considered. The following sections will reveal how this approach is accomplished.

The use of signature descriptors alone in characterizing properties of interest has proven a successful endeavor (Faulon and Churchwell, 2003), but the power of this technique lies in the ability to reconstruct solution structures, with low degeneracy, for any given set of atomic signatures. This idea was introduced in section 2.5.3, and will be covered in further detail in the following sections. The reason that signature descriptors are so useful during the reconstruction of solutions is that they are, by definition, overlapping fragments. This allows for the systematic application of specific rules to enumerate their connectivity. For a given set of signatures, for which it has been confirmed that there exists at least one structural isomer, there are often multiple isomers that can be created, each of which refers to the same original set of atomic signatures. This feature allows for the systematic consideration of a region of chemical space in which no two structural isomers are considered more than once. In addition, the height of atomic signatures utilized in the study can be controlled such that the resulting solutions generated have a pre-defined degree of similarity to the original data set, from which the signatures were generated. This follows the pattern of smaller signatures resulting in a larger search space while considering more diverse structures and larger signatures doing

just the opposite. This becomes useful when the applicability domain of property models utilized has certain limits, based on similarity to the structures used to develop the model, beyond which the predictivity is less reliable.

Most descriptors between 0D and 2D can be derived from molecular signatures. This allows one to solve existing SARs in signature space, while maintaining the predictability of the original SAR along with the low degeneracy attributed to signature descriptors in enumerating potential solutions. Extension of the signature descriptor to include spatial information maintains the combinatorial efficiencies seen in previous applications while offering the discriminatory power of including descriptors of higher dimensionality in the property models. The general approach, developed within this dissertation, for using these descriptors in a CAMD application has been outlined in Figure 3.1. This approach can be categorized as a generate-and-test CAMD methodology with a very efficient generation step that quickly identifies structural isomers, which are later subjected to conformational analyses.

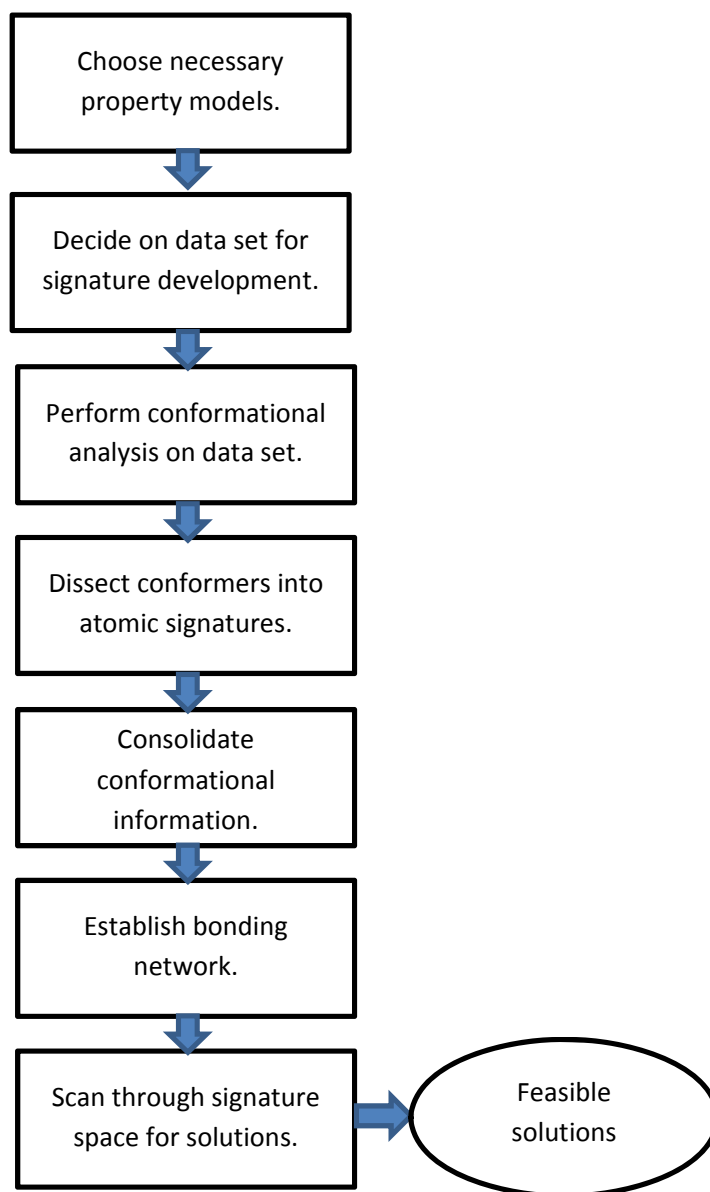


Figure 3.1 - Overview of methodology developed.

3.1 Deterministic Solution Approach

A deterministic algorithm is one in which, given a particular input, the output always remains the same. In addition, the underlying machine always passes through the same sequence of states. These are the most familiar kind of algorithms and are also the most

well studied. Deterministic approaches are suitable to problems which have a small search space or perhaps have highly complex nonlinear equations involved such that linear and nonlinear programming are not feasible techniques. When applied within the field of computer aided molecular design this typically means that every possible solution is visited such that the search space is entirely considered. This offers the benefit of identifying a globally optimal solution, in addition to several other potential candidates, but can suffer from increased running times.

3.1.1 Identification of Property Models

The first step in setting up a problem, within this methodology and most any CAMD approach, involves identification of the necessary property models. There are many types of property models available, which are created using a variety of regression and mapping techniques. The nature of these models, whether they are fragment based or non-linear etc., will help decide the solution approach that must be taken. When a group contribution model is chosen, this information must be written in terms of atomic signatures. Depending on the order of group contributions utilized, varying height signatures will be necessary. Higher order groups will often require larger, higher height, signatures. If all of the models involved are linear and no 3D descriptors are utilized, the problem can be solved using LP (linear programming) techniques. However, all studies considered within this dissertation include 3D descriptors, and as such, various deterministic and stochastic approaches must be applied. Once the necessary property models have been identified, or developed, a data set must be decided upon from which the signatures will be derived.

3.1.2 Data Set Selection

Selection of an appropriate data set for this approach is a very important step because the building blocks for generating candidate molecules are the atomic signatures found in the original set. This means that all solution molecules generated represent a controlled interpolation, or extrapolation, of the chemical space spanned by molecules in the data set. This search is ‘controlled’ by the fact that the atomic signatures generated can have a variable height. Selecting a larger height would result in molecules generated with higher structural similarity to the data set, whereas using a smaller height would allow for more degrees of freedom upon recombination resulting in less similar structures. The tradeoff would be that, in generating more molecules from a smaller height atomic signature, these structures would have an increased likelihood of falling outside the applicability domain for a given property model.

Utilizing the training set on which the property models were developed as a pool for atomic signature development does offer the advantage of generating solutions with increased similarity, however this is no guarantee that these structures will fall within the applicability domain (AD) of the given property models. Sheridan et al. (2004) found that molecules with higher similarity to the training set resulted in the best predicted properties, as measured by root-mean-square difference between observed and predicted activity, for narrow training sets with minimal diversity. However, for more diverse training sets, it becomes unreasonable to define the AD in terms of similarity and alternate methods are necessary. These methods often require projection of the training set into descriptor space with subsequent specification of the AD, now represented by a convex hull in multivariate descriptor space, through various approaches. A review of

these techniques can be found in Jaworska et al. (2005). For these reasons, when using a property model generated from a diverse training set, it is possible and potentially beneficial to utilize an alternate set of molecules for atomic signature development. These molecules could be chosen from a library of compounds with established drug-likeness or perhaps other desirable traits and the generated candidates could be tested against the previously established AD.

3.1.3 Conformational Analysis

The conformational space for each molecule in the data set is explored to identify energetically accessible conformations corresponding to local energy minima on the potential energy surface (PES) for that structure. There are several techniques available for identifying these conformers and they can be generally categorized as being either systematic or stochastic. The choice between these two techniques is typically a function of the anticipated complexity of a given PES. Stochastic techniques, such as those based on Monte Carlo (Metropolis and Ulam, 1949) or molecular dynamics simulations (Karplus and McCammon, 2002), have proven beneficial towards quickly identifying local energy minima for more complicated PES's. Whereas systematic conformational analyses are much more likely to identify a global energy minimum, at the expense of increased computational time (Beusen et al., 1996). Regardless of the chosen method, the goal is to utilize the information obtained through conformational analysis on the data set to estimate spatial characteristics of each molecule considered within the CAMD approach. In doing so, it alleviates the necessity to explore the PES of each molecule considered during the search. This approach will help minimize the time taken to estimate the 3D characteristics of each candidate molecule considered, such that a larger search

space can be explored within reasonable time constraints. Molecules meeting this initial estimation of spatial criteria can be further explored with more accurate simulations.

3.1.4 Spatial Signature Development

As covered earlier, the molecular signature of a given structure is a linear combination of its constituent atomic signatures, which can be seen as atom-centered subgraphs of their respective complete molecular graph. The Signature Translator Program (v. 3.0), developed by Faulon et al. (2003), is implemented in Unix to calculate the set of canonical atomic signatures found in each conformer, which were identified through the previously mentioned conformational search. The program accepts as input a .mol formatted file and the desired signature height and returns an output text file containing the respective atomic signatures. Structural isomers are quickly identified, through matching atomic signature strings, and grouped together along with the cartesian coordinates of each atom found within the structure. The spatial information for each atomic signature was developed in this manner, as opposed to isolated atomic signature simulations, such that the resultant geometry information was derived from the signatures embedded within various molecular structures found within the data set. The goal is to obtain the most realistic, and accessible, conformations for the fragments, which would likely not be seen in isolation.

3.1.5 Compression of Spatial Information

The complete set of atomic signatures, developed from the individual conformers identified through a conformational analysis of the chosen data set, must now be reduced to eliminate conformational redundancies. The technique developed for this task is to compare all pairwise distances between atoms for the given structural isomers. The

absolute difference between pairwise distance values is analyzed subject to a given cutoff value. Two conformers exhibiting a difference in spatial features greater than the selected cutoff value are both maintained as being unique, whereas those with all similar distances are found to be too alike and one is discarded from further use. This is exemplified as shown in Figure 3.2, where one of these two conformers would be discarded at random since the difference in their interatomic distance chosen is less than the cutoff value.

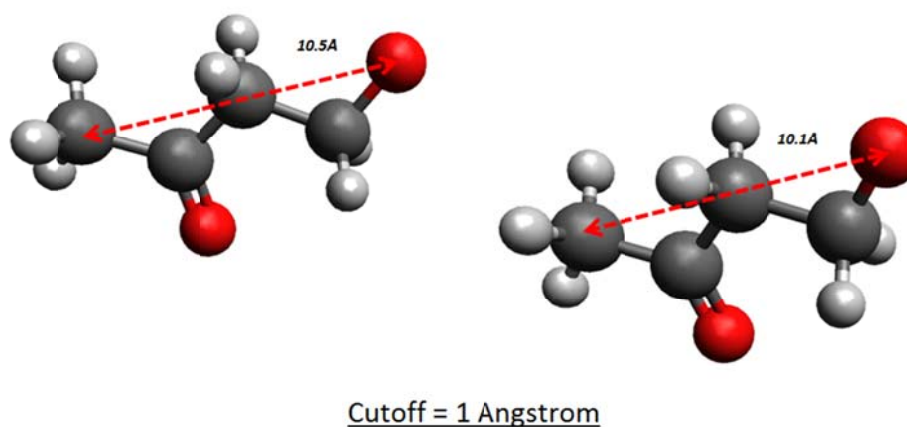


Figure 3.2 – Cutoff criterion example.

The cutoff criterion can be varied to increase or decrease the ‘fineness’ of conformational information ultimately stored. An isomorphism mapping between the conformers is generated to facilitate such a comparison. NetworkX (Hagberg et al., 2008) is a python language software package containing many modules and functions useful for the graph theoretical analysis of molecules, as well as countless other applications. This software is used to turn each atomic signature into a graph object, which represents the molecular fragment encoded within the signature. The isomorphism algorithm within NetworkX is used to quickly generate the mapping between two conformers, which are structural

isomers of each other, such that pairwise distances can be directly compared. It is first necessary to generate a graph occurrence for each unique atomic signature string and this is done through application of an in-house python script which takes advantage of the Python regular expression operations. This script was created to facilitate the conversion of information within a molecular signature into a graph object and links together modules from varying software to facilitate such a task. This approach results in compressing the spatial information calculated in the conformational analysis step, and the end result is a set of canonical atomic signatures, or unique structural isomers, with their accessible conformations stored in the form of cartesian coordinates for each atom involved. Table 3.1 reveals the compression of information of a set of common industrial solvents, which resulted in 73 unique structural isomers, as the cutoff criterion is changed.

Table 3.1 – Cutoff criterion compression example.

Cutoff Criterion (Angstrom)	Conformers Remaining (387 initially)
0.05	282
0.1	271
0.2	257
0.3	240
1.0	199
2.0	182

3.1.6 Creation of Bonding Network

Throughout the isomer enumeration approach, the test of bond compatibility will be made for the same two signatures multiple times. To avoid the unnecessary expenses of repeatedly calculating the same information, this algorithm initially generates a network of bonding capabilities between fragments in the established atomic signature basis set. The NetworkX software is employed to assist in developing and storing this information. Each atomic signature is initiated as a node in what is further referred to as the ‘bonding network’. This network initially contains no edges and signatures, or nodes, are compared pairwise to establish all possible bonds. A bond between two atomic signatures, ${}^h\sigma(x)$ and ${}^h\sigma(y)$, is possible when the ${}^{h-1}\sigma(z)$ signature of atom z , neighboring atom x , matches the ${}^{h-1}\sigma(y)$ signature. This can also be stated, from a graph theoretical viewpoint, as a subgraph, with radius $h-1$ centered at an atom neighboring the x -signature center atom, being isomorphic to the radius $h-1$ subgraph centered at the y -signature center atom. This is visually represented in Figure 3.3 where two nodes in the bonding network, x and y , are represented by their underlying atomic signatures. A bond would be created between these two nodes because the ${}^{h-1}\sigma(z)$ signature of atom z , neighboring atom x , matches the ${}^{h-1}\sigma(y)$ signature.

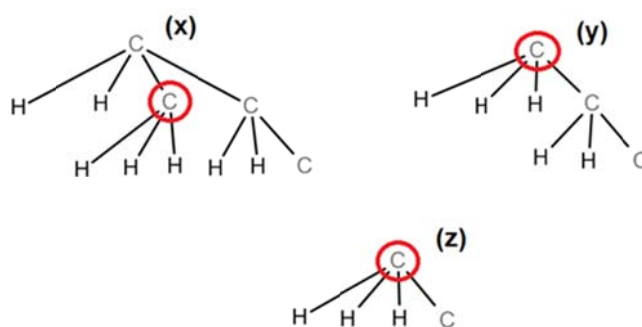


Figure 3.3- Example of compatible ‘bonding network’ edge.

Inclusion of pre-calculated bonding capabilities between fragments generated from the initial data set is one of the novel implementations found within this methodology which has proven useful in speeding up the CAMD approach. Atomic signatures are essentially over-defined, such that they overlap when recombined, and this is taken advantage of to expedite the structural isomer enumeration process. Each set of fragments, or atomic signatures, can typically be used to build several unique structural isomers and as such this network comes in handy.

3.1.7 Generation of Structural Isomers

The calculation of 3D descriptors makes the approach much more complicated as the connectivity of atomic signatures must be established first such that a global geometry can be calculated from the combination of local geometry information provided by each fragment. Atomic signatures, by definition, are overlapping subgraphs of the complete molecular graph, which they collectively reference. For a group of atomic signatures to generate at least one structural isomer there are several feasibility constraints that must be met. There are two types of equations within this category and they are the graphicality equation and the consistency equations. The graphicality equation is developed directly from graph theory and ensure that at least one molecular graph can be created from a set of atomic signatures. In order to build a connected graph, it is required that (1) the sum of all the vertex degrees must be even and (2) the number of vertices of odd degree must be even. The graphicality equation, Eq.(2.30), can be analyzed directly from the height zero molecular signatures, which conveniently offers the degree of each vertex, or atom, in the molecular graph. The consistency equations ensure that, since the atomic signature

overlap with each other, the interdependency of each signature upon the others is mathematically feasible. An example of one such consistency equation is represented in Figure 3.4, where the number of oxygen-carbon single bonds must equal the number of carbon-oxygen single bonds. In the case for nitroglycerine there is only one atomic signature with an oxygen root single-bonded to a carbon atom. However, there are two atomic signatures with a single bonded carbon root. In each case, the carbon root is connected to a single bonded oxygen atom shown with a dashed line. There is only one instance of a carbon-oxygen single bond in both signatures, so the occurrence number is the sum of these, which is 3.

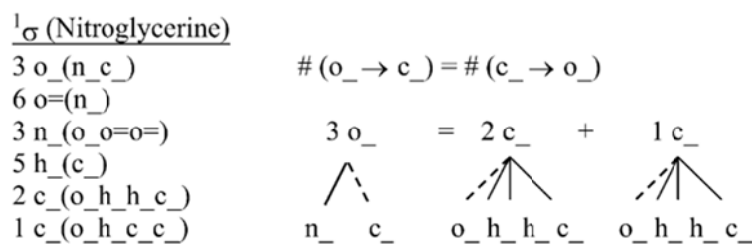


Figure 3.4- Example of consistency equation for nitroglycerine.

Once it has been established that at least one structural isomer is possible, through the previously mentioned structural feasibility constraints in, the task remains to identify these isomers. The approach taken to establish connectivity is based on the orbit saturation algorithm established by Faulon and Churchwell (2003), and has been implemented as a Python script. The algorithm takes advantage of the degeneracy stemming from subgraph isomorphisms, and saturates equivalent subgraphs at the same time and in all possible ways until a complete saturated structure is obtained. The

approach begins by representing each atomic signature as a node within a graph, G , containing no bonds, or edges. The underlying atomic signature represented by each node will contain a central atom with one or more neighbors, all atoms one bond away from this central atom. These neighbors must each be mapped to another node in G , thus creating a bond in G and eliminating this neighbor from further mapping requirements. Once the list of neighbors for a node in G is empty, this node is said to be ‘saturated’ and the same should be done for all remaining unsaturated nodes. The order in which this is done, and the criteria for acceptance of a potential bond in G , have a huge impact on the speed of the algorithm and are explored in further detail below. In addition, this process is visually represented in Figure 3.5.

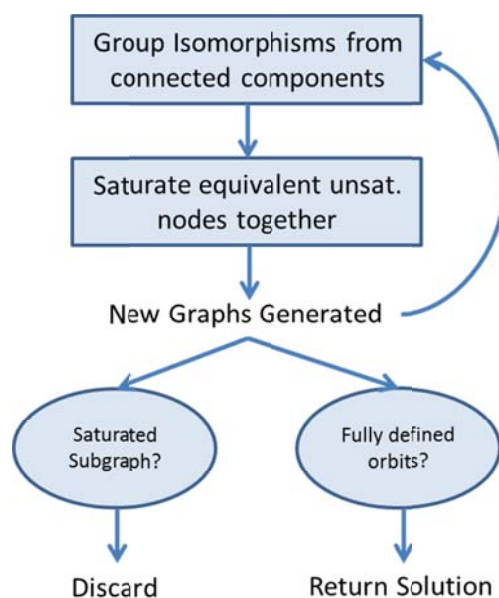


Figure 3.5 – Generation of structural isomers.

The connected subgraphs of G , initially all isolated nodes, are first partitioned into groups containing equivalent isomorphisms and the group containing members with the

least degree of unsaturation is chosen to be saturated. Typically, this first group of nodes would have hydrogen as the root atom, thus containing only one degree of unsaturation (the neighbor of this hydrogen). All nodes in this group are saturated in all possible, and unique, ways such that a new graph is created for each possibility. Each of these new graphs will contain several connected subgraphs, which are partitioned, as before, into groups representing unique isomorphisms. This task is facilitated through the NetworkX `connected_components` function, which takes the graph G as input and returns the various connected subgraphs. The summation of unsaturated neighbor atoms, for each node, or atomic signature, in these new subgraphs would represent the total degree of unsaturation for the given subgraph. Each of these unsaturated neighbors would then be saturated in all possible ways, creating a new set of graphs stemming from the previous set, each containing their own connected subgraphs. This process continues until one of two conditions is met:

- 1) The graph G is completely connected, with the `connected_components` function returning one item, and fully saturated. This graph is returned as a feasible structural isomer.
- 2) The graph G contains a connected subgraph component with a degree of unsaturation equal to zero. This represents a saturated subgraph and G is discarded from further iteration.

3.1.8 Generation of Conformational Isomers

The global geometry of molecular graphs with feasible connectivity must now be established for the calculation of higher dimensional descriptor values. The goal for this

step is to minimize the computational complexity of such an operation by utilizing only those signatures necessary for the establishment of a complete geometry. As mentioned previously, the atomic signatures for a respective molecule overlap with each other and this can be taken advantage of when transforming individual fragment coordinates towards a consistent coordinate space. Not all atomic signatures for a given molecular signature are necessary for this approach, and the steps taken to choose those required are outlined below:

1. The largest signature, based on number of atoms, is selected as a reference frame. If more than one exists with same size, the signature with the least number of unsaturated nodes is chosen. Beyond this, a random signature is chosen among several which have the same size and number of unsaturated nodes.
2. All atomic signatures sharing at least three atoms in common (identified through an isomorphism mapping) with the initial signature are determined. Three atoms are necessary for docking the two fragments together in Cartesian space.
3. From this set, the signature defining the largest amount of new atoms is selected. In the case that several signatures are equivalent in this measure, a random one is selected.
4. The transformation matrix is established and applied to all atoms in the second signature such that consistent coordinate frame is established.
5. Signatures having at least three nodes in common with the graph represented by the combination of the first two signatures are enumerated.

6. From this set, the signature defining the largest amount of new atoms is selected and a transformation matrix is established to dock this atom into the growing molecular graph.
7. Steps 5-6 are continued until every node in the given molecular graph has been defined in the consistent coordinate system established by the initial signature. The resulting set of signatures, with their order fixed, will be referred to as a 'superposition map.'

This procedure can be visualized through the example seen in Figure 3.6 shown below. The steps have been labeled for the chosen example, which begins with selecting the largest atomic signature having the fewest degrees of unsaturation.

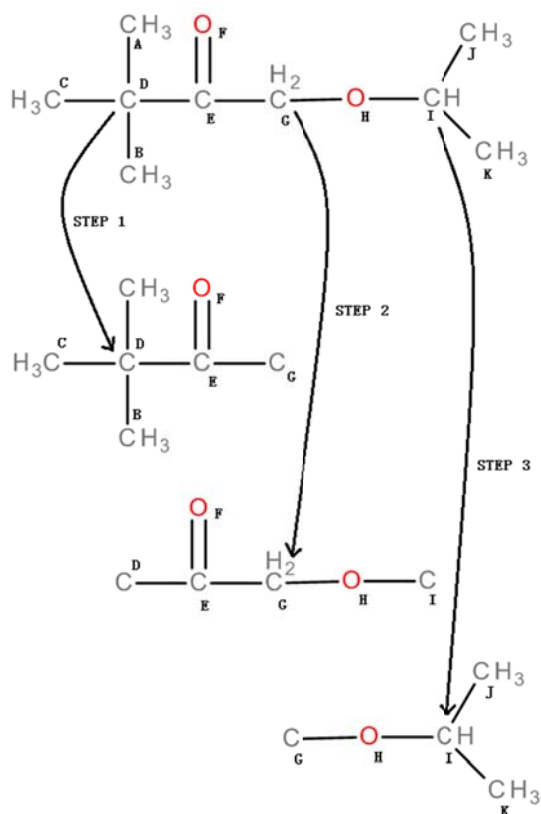


Figure 3.6 – Example of fragment selection for geometry development.

This signature would be the one centered at the carbon at labeled 'D' as it contains 16 atoms, within the fragment established by its atomic signature, with the carbon atom labeled 'G' being the only unsaturated atom. Step two would necessitate identifying another signature overlapping by at least three and identifying the most new atoms. The atoms labelled 'A', 'B', 'C', 'E', 'F', and 'G' all satisfy the three common atoms rule, however, the atomic signature centered at atom 'G' defines the largest number of new atoms, being 2. Within the structure created by linking these first two atomic signatures lies only one unsaturated carbon, which is labeled 'I'. The signatures overlapping this growing fragment, created by linking the first two, by at least three atoms include all atoms labeled 'A' through 'I'. Out of those, 'I' would define the most new atoms, thus completing the structure with zero degrees of unsaturation remaining.

The steps outlined previously chose fragments to be transformed into a consistent coordinate system, and this information was stored in the form of a 'superposition map'. A subgraph isomorphism mapping is stored within each compatible bond, as a dictionary, in the bonding network established previously. It was required that each pair considered for superposition has at least three atoms in common within this mapping. For the first two signatures in the superposition map, an equivalent pair of atoms is randomly chosen from the set of mapped atoms and a translation vector is calculated from the difference in their coordinate vectors. A second pair of mapped nodes is chosen, along with the first pair, to form an axis in each fragment for development of a rotation matrix. Since these two vectors have one point in common, or one overlapping point after the original translation, the Euler rotation theorem can be utilized to develop a rotation matrix which

effectively superpositions the fragments to obey the given mapping. The cross product between these two vectors provides a perpendicular axis around which to rotate by an angle specified by the dot product between the vectors. The next step is to calculate the angle of rotation around the axis, represented by the first two pairs of mapped atoms, which minimizes difference between the remaining mapped atoms. This is done by systematically scanning through angles by a given increment until a minimal deviation in the remaining mapped atoms coordinates has been identified. The result of these calculations is a transformation matrix, represented in homogenous coordinates, which translates and rotates a given atomic signature into the desired reference frame. This matrix \mathbf{R} is exemplified in Equation 3.2. This procedure is performed for each fragment identified within the superposition map such that set of transformation matrices is established which effectively brings the individual fragments into a consistent reference frame.

$$R = \begin{bmatrix} \cos \theta + u_x^2 (1 - \cos \theta) & u_x u_y (1 - \cos \theta) - u_z \sin \theta & u_x u_z (1 - \cos \theta) + u_y \sin \theta \\ u_y u_x (1 - \cos \theta) + u_z \sin \theta & \cos \theta + u_y^2 (1 - \cos \theta) & u_y u_z (1 - \cos \theta) - u_x \sin \theta \\ u_z u_x (1 - \cos \theta) - u_y \sin \theta & u_z u_y (1 - \cos \theta) + u_x \sin \theta & \cos \theta + u_z^2 (1 - \cos \theta) \end{bmatrix}. \quad (3.2)$$

For each molecular signature identified as having at least one unique structural isomer and meeting lower dimensional criteria, the superposition map, along with the respective transformation matrices, is utilized to estimate the energetically accessible conformations this ensemble of fragments could attain. Each atomic signature will have at least one set of coordinates which has been derived from the initial conformational analysis performed on the given data set. These conformers are utilized in a combinatorial manner, along with the respective transformation matrices, to develop a set

of conformers which are tested against the necessary 3D constraints. NumPy, a package for scientific computing within Python, is used to handle the coordinate transformations (Oliphant, 2007).

There are multiple options in using this information and two examples are shown in Figure 3.7. This example follows the notation from Figure 3.6, where the three atomic signature fragments necessary to establish a global geometry were identified. While each atomic signature will have its own set of conformational isomers, only the conformers from signatures labeled 'A', 'G', and 'I' will be used to develop the complete conformational isomers for the chosen molecule.

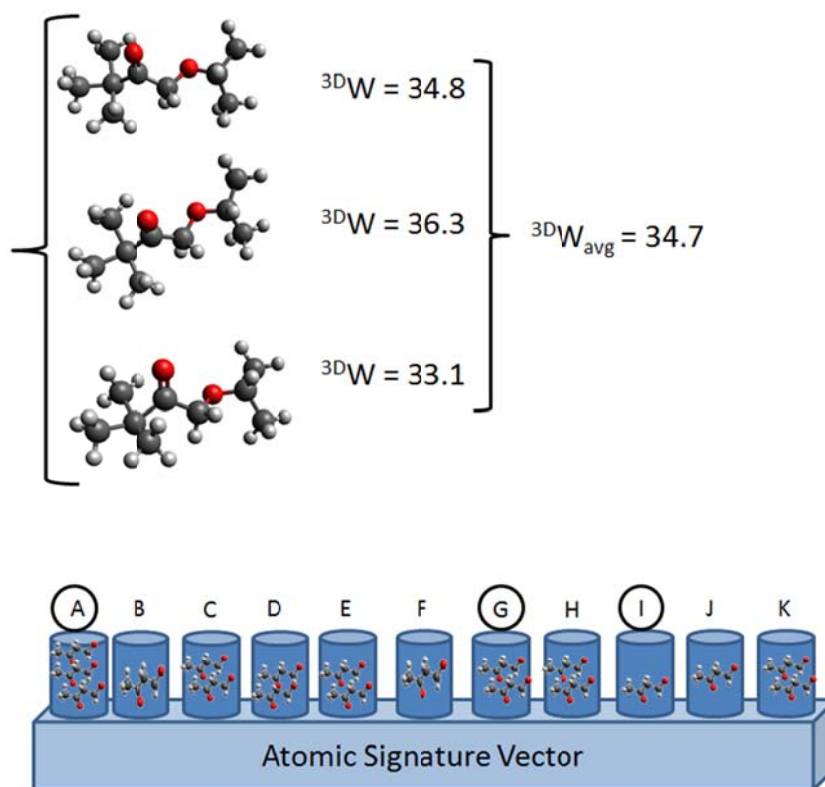


Figure 3.7– Utilization of developed conformational isomers.

The ensemble of conformers generated can be used to calculate an average value for 3D descriptors of a given structural isomer, as shown with the three-dimensional Weiner index. They could also be individually tested against the given criteria since the actual molecule in solution would be represented, to some degree, by a collection of various accessible conformations, which might not be well captured through averaging. In any manner, this approach aims at estimating the accessible conformational space of a given molecular signature while avoiding an extensive conformational analysis for each candidate molecule.

3.1.9 Extension to More Complex Structures

The initial studies, which were used to demonstrate the feasibility and applicability of the proposed methodologies, were based upon simplistic structures. These were relatively simplistic in the fact that they were medium sized organic molecules containing no ring structures. This was acceptable since the main motivation was to exemplify the proposed methodology's ability to solve computer-aided molecular design problems with multi-dimensional constraints. However, once proven, the next step would be to test the limits of this technique against problems of increasing complexity. Extension to consider more complex molecules with advanced characteristics would necessitate the formulation of new structural constraints. In addition to topological considerations, the conformational complexity associated with larger structures is also more evident. This topographic burden would show itself in the form of more complex potential energy surfaces and would require an adapted technique to handle such a case. The steps taken to handle such situations will be covered in the following paragraphs.

To foreshadow the possibilities of applying this methodology to more interesting case studies, it would be worthwhile to present some information on how the signature descriptor itself has performed when applied to complex case studies. Signature descriptors have been used as independent variables in the design of novel polymers with targeted properties (Brown et al., 2006). One of the unique characteristics of this descriptor is the ability to vary its ‘height’, which corresponds to capturing atoms at a further distance from the central atom. The height chosen for generating signature descriptors from a given data set has a direct effect on the resulting chemical search space, which is where novel solutions are identified. The immediate solution to considering case studies with larger molecules might be to use a larger height signature, however, this would have the effect of limiting potential solutions to having a very high similarity to the original data set. In addition, using a smaller height signature would result in combinatorial explosion. This was well recognized within the polymer design study by Brown et al. (2006), and they were able to select an ideal height for the design of novel polymers, which generated solutions as large as 45 atoms while using height one signatures. While this was feasible for a study with only topological constraints, the development of global geometric properties from signature fragments requires at least height two signatures.

The ability of a descriptor to be mapped back to a given number of potential solutions is determined by its degeneracy. Descriptors with high degeneracy will map to a larger number of solutions, which becomes computationally intensive to enumerate all possible structures. The degeneracy associated with the signature descriptor, with respect to varying types of molecules with varying complexity, has been studied (Faulon and

Churchwell, 2003). With respect to alkanes, using height two signatures, 57.5% of structures had no degeneracy, meaning that they referred to only one structural isomer. As for alcohols with the same height two signatures, 99.2% of structures showed no degeneracy at this height. These were relatively simplistic structures, however, when extended to consider fullerene-type structures, 99.7% were uniquely identified at a height of three. In addition, 98.9% of peptides were non-degenerate when a height three signature was applied. This establishes a boundary when applying the signature descriptor in molecular design case studies, beyond which would allow for a more similar set of solutions or a larger search space. In any case, this boundary can be identified for each unique case study even when applied to more complex structures.

Molecular complexity comes not only in the form of larger structures but also more interesting structural features. One such example would be the inclusion of ring like structures, which are found in chemicals across all fields. The signature descriptor is adept at handling such features and is enabled through its foundation in graph theoretical concepts. In addition to the extra topological constraints, there will also be some differences in the topographic characteristics of more complex molecules. The potential energy surface of a given molecule can quickly become very complex based on the size of the structures. The degrees of freedom for such a surface can be calculated as $3N - 6$, with N being the number of atoms in a molecule. Fortunately, many of these features are quite rigid and the only real complexity arises in the torsional bonds found with a given structure. This can still lead to extremely complex potential energy surfaces, which are explored to estimate the spatial capabilities of atomic signatures, and the resultant solution structures, within this methodology. As such, to handle larger structures, it

would become necessary to conduct a more thorough conformational analysis of the original data set. This could also lead to many more structural isomers associated with each atomic signature, representing a unique structural isomer. To avoid the computational complexity associated with such a situation, as well as the potential combinatorial explosion, it would become necessary to increase the cutoff value while compressing the spatial information as discussed in section 3.1.5. This would be a trade off in the quality of each estimated geometry, but with an optimized cutoff value would still generate reasonable geometries. Additionally, it would also be possible to relax topographic constraints associated with a given case study and perform a more thorough conformational analysis of the solutions identified. These techniques could be adopted to approach studies with more complex structures.

3.2 Stochastic Solution Approach

Often times, the nature of CAMD problems involving higher dimensional descriptors is such that linear programming cannot be applied. The transformation from property space to descriptor space can be non-linear as well as that from descriptor space to chemical space, which is where our solutions lie. In addition, the search space for a given CAMD problem is necessarily large, which makes the implementation of deterministic search methods infeasible due to combinatorial explosion. Combinatorial explosion arises within this field because of the utilization of molecular fragments as building blocks. There are so many potential arrangements of these building blocks which can lead to novel solutions.

A stochastic solution approach can circumvent this problem by searching a smaller subset of the original space, with guidance by an appropriate algorithm. This

allows for a much faster search algorithm, albeit at the expense of not considering the entire chemical search space. A stochastic approach can be completely random, or more beneficially, guided towards certain regions of the search space by a given convention. One such approach, as applied in this document, is known as a genetic algorithm (Holland, 1975). Genetic algorithms, GAs, apply a search heuristic which mimics nature by evolving a population of candidates towards an improved 'fitness' by means of several operators. This fitness is often measured as a candidate's closeness to a given set of properties. Those within the desired property range have a higher fitness, whereas those falling outside this range are penalized and have a lower fitness. This satisfies the 'natural selection' part of the algorithm while candidates are altered through various operations which can be categorized as mutation, acting on a single candidate, or crossover, typically involving two candidates. These operations allow for the population to evolve towards an improved overall fitness. One of the benefits of GA applications is that multiple solutions are often found even if the algorithm does not converge. For this reason, GA has been applied very successfully in several CAMD applications (Venkatasubramanian et al., 1995; Douguet et al., 2000; Pegg et al., 2001; Kamphausen et al., 2002). The following Figure 3.8, exemplifies the typical structure of a GA approach.

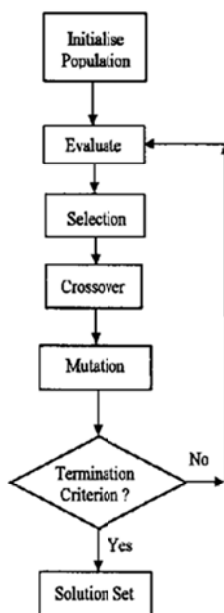


Figure 3.8 – Common Genetic Algorithm Methodology

Within this generalized methodology it can be seen that an initial population is first developed and evaluated by the criteria established by the molecular design problem. Most often this criteria comes in the form of one or more property models as well as various structural constraints. A value is associated with how close each candidate structure matches the criteria, and this is facilitated through a fitness function. The process of selection is based on this fitness value previously established, which provides the component of pressure as paralleled by survival from the environment in natural selection. The more ‘fit’ population members then undergo various operations, which can be categorized as mutation or crossover. The crossover technique takes two structures and breaks them at one or more points, then recombines them to form a new molecule containing features from each of its parents. The mutation operator changes part of the chosen molecular structure by substitution, deletion, or addition of molecular fragments

ranging from simple atoms to larger molecular groups. Termination of the algorithm is decided upon when the overall fitness of the resulting population has reached a certain threshold or when a specific number of iterations have been implemented and the final solution set is what remains.

This generalized methodology has been implemented within the previously covered CAMD algorithm such that a stochastic solution approach could be applied. The initial steps for this new methodology are the same as before in that the desired properties along with their respective models must first be identified. In addition to this, a conformational analysis must be performed in the same manner such that the spatial capabilities of the respective region in chemical space can be captured. Following this, the generated conformational isomers are dissected into spatial signature descriptor fragments for use in creating solution structures in a combinatorial manner. The following steps include application of a GA which has been adapted to satisfy the use of spatial signature descriptors. New operators have been developed which facilitate the use of spatial signature descriptors since they are overlapping in nature. A single point crossover, in addition to insertion, deletion, and random mutations, has been developed. This methodology is useful for solving complex non-linear CAMD problems utilizing molecular descriptors of varying dimensionality while searching a large region of chemical space in an efficient manner.

3.2.1 Overall Genetic Algorithm Methodology

Figure 3.6 provides a depiction of the proposed GA algorithm for the solution of multi-dimensional CAMD problems utilizing spatial signature descriptors.

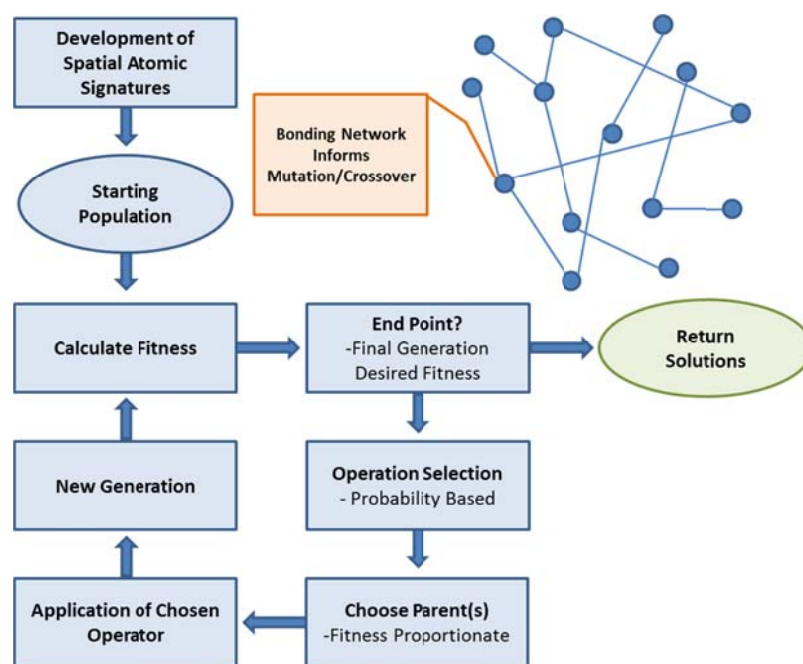


Figure 3.9 – Proposed Genetic Algorithm Utilizing Spatial Signature Descriptors

From Figure 3.9, it can be seen that the spatial capabilities of the desired search space have been characterized using the methodology covered in sections 3.1.3 – 3.1.5. In addition, the algorithm is informed by the bonding network, covered in section 3.1.6, which enumerates all possible bonds between each pair of atomic signatures. This proceeds into the creation of an initial population, which is then evolved by means of various operators into successive generations with potentially improved overall fitness values. An end point is reached when the desired number of iterations has passed or the desired average fitness has been reached within the final population. The conventional representation of an individual within the GA scheme is typically represented as a string of binary values, each associated with the presence or absence of a specific feature. However, since molecules often contain individual features multiple times, such a representation would be quite limiting. Extension to include integer occurrences of

structural fragments would likely lead to the consideration of many collections of fragments which do not represent a feasible structure when combined. For these reasons, the individual members of this GA approach are represented as a type of chemical graph where each node within this graph represents an underlying atomic signature. This can be visualized in Figure 3.10 where the height two atomic signatures are represented for two nodes within a chemical graph.

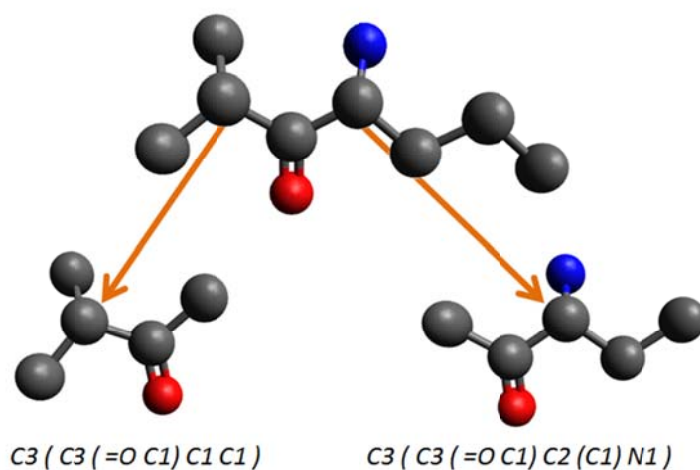


Figure 3.10 – Atomic signatures for nodes in molecular graph

Just as these two atomic signatures, along with the fragments they represent, are shown for the example nodes, each node within the graph would have its own respective atomic signature. Nodes are added in a systematic manner along with bonds, which are informed by the bonding network mentioned previously. This ensures that only structurally feasible molecular graphs are considered, which helps alleviate the combinatorial complexity associated with problems of this nature.

3.2.2 Generation of Starting Population

An initial generation of molecules is selected at random to develop a population, which encompasses a variety of structural features spanning the chosen chemical space, on which to begin reproduction. The input required for this step consists of an upper and lower limit on atom count as well as a population size. For each member of the chosen population size, a target atom count is selected at random from the acceptable size range. Then a node is selected at random from the bonding network and the possible neighbor list is developed, from which the subsequent signature fragment is chosen. This process continues for a growing graph while the selection of signature fragments is a function of the current graph size. Equation 3.3 is utilized as a probability function for the selection of a signature which would either maintain graph unsaturation or effectively cap the growing molecule when nearing the desired size. In Equation 3.3, P_{cap} is the probability of selecting a signature fragment which would saturate the growing graph, N_T is the target number of atoms and N_C is the current number of atoms.

$$P_{cap} = 1 - \left[\frac{(N_T - N_C)}{N_T} \right] \quad (3.3)$$

This constraint helps adhere to the desired range of graph sizes for the starting population. It is possible for the probability to exceed a value of one, in which case graph growth would be terminated as soon as possible. The following pseudo-code outlines the algorithmic approach to generating an initial population:

```
pop_size = 100
size_lower = 10
size_upper = 30
size_list = [x for x in range(size_lower,size_upper+1)]
```

```

for x in range pop_size:
    select random node
    select random desired size
for growing_graph in population:
    while unsaturated == true:
        identify cap probability
        if cap == true:
            identify nodes facilitating saturation
        else:
            identify nodes facilitating growth
    else:
        saturated graph created

```

The desired population size is first chosen, along with the desired upper and lower limit on the size of molecular graphs generated. For each member of the population an initial signature is chosen at random along with the desired size, which is chosen from the developed size list. The molecular graphs are then ‘grown’ by selecting adjacent nodes, which have signatures capable of overlapping with the previous nodes signature. During the growing process, the probability of choosing a node which effectively ‘caps’ the graph by generating a fully saturated molecular structure, is calculated as shown in Equation 3.3. The value of P_{cap} can range from zero to one, and the closer this value is to zero the more likely the algorithm is to ‘cap’ the graph. This leaves some degree of randomness in the selection of whether or not to terminate growth of the molecular graph. Once the operation of cap or grow is chosen, all nodes facilitating this operation are identified and one is chosen at random from this list. This continues until a population of fully saturated molecular graphs has been created.

The chosen methodology of generating a starting population has an inherent degree of randomness such that the diversity of the chosen chemical space has an equal

opportunity of representation. However, control over the size of these individuals has been maintained. This has been done by choosing a desired size distribution within which the population should be maintained such that runaway growth does not occur. In addition, it is also undesirable for the generated population to have members with a graph size that is too small. While this does require some knowledge on the desired size distribution for a given problem, this can be circumvented by selecting a wider size range for situations in which this information is unknown. The graphs generated by this method will provide the population on which subsequent mutation and crossover operations are applied to create potentially improved solutions. As such, these graphs can be seen as points in chemical space which serve as starting points for the further interpolation type search throughout the remaining regions of chemical space.

3.2.3 Fitness Calculation

The fitness function plays a vital role in the guidance of a genetic algorithm towards a set of improved solutions, especially within a large search space. The best fitness functions will help a GA explore the search space more efficiently and effectively. However, a bad fitness function could result in the GA being trapped within a local minimum, lacking the power to explore other regions. The fitness function chosen for this application is expressed as shown in Equation 3.4.

$$f_i = \exp\left(-\alpha \left[\sum_i^n \frac{(P_i - \bar{P}_i)^2}{(P_{imax} - P_{imin})^2}\right]\right) \quad (3.4)$$

This format was chosen as suitable for a molecular design application because the problem is most often expressed in terms of a set of desired property ranges. In this case,

the fitness of each molecular graph is calculated with a Gaussian-like function over the range of desired property constraints. Within this equation, P_i is the calculated property value, P_{imax} and P_{imin} are the upper and lower property range values, and \bar{P}_i is simply the average of these two values. The constant, α , is known as the Gaussian fitness decay rate and characterizes how quickly the fitness falls as it leaves the desired property range. This effect is exemplified in Figure 3.11, where the value of α has been varied with its effect on the fitness distribution shown graphically.

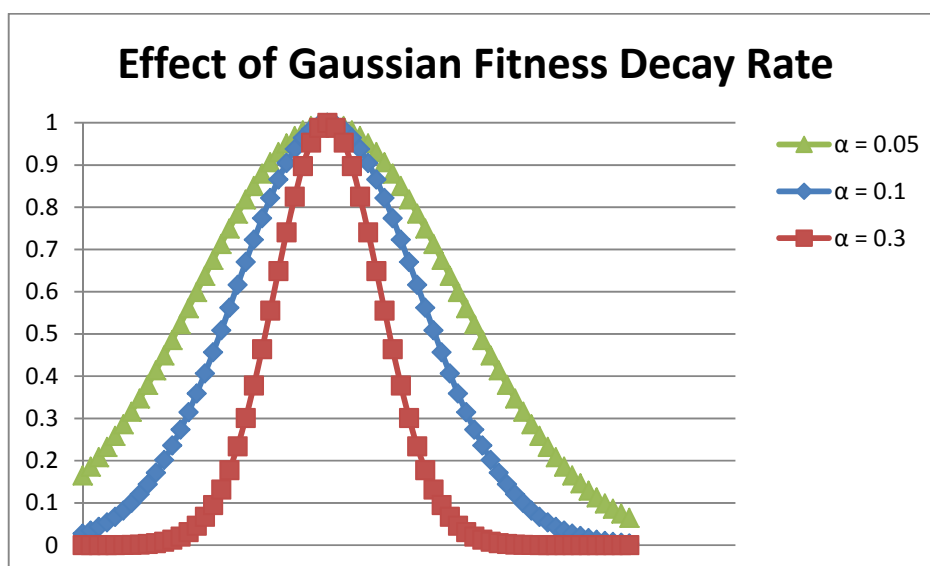


Figure 3.11 – Effect of α on Fitness Distribution

The values produced by this fitness function essentially determine how desirable a given molecular graph is with respect to the chosen property ranges. Structures falling outside these ranges in one or more property category are penalized, resulting in a lower fitness value. Fitness values range from zero to one, with one being considered an optimal solution. Functionally, the way fitness values are used to select optimal candidates is that larger values have a higher likelihood of being chosen for mating or

mutation, and this is a probability based approach. While ‘more fit’ solutions do have an increased likelihood of being chosen, less optimal solutions can also be represented to a varying degree by altering the alpha value mentioned earlier. A smaller alpha value would allow for the increased likelihood of selecting less optimal solutions, while a larger value would be very strict in only selecting the best structures. Some degree of sensitivity analysis is necessary to identify a value that’s optimal for a given case study and this depends on the nature of chemical space considered, as well as its relationship to the chosen properties of interest.

3.2.4 Genetic Operators

Various genetic operators are applied to maintain diversity within the population and effectively explore the available chemical space. New techniques were developed to tackle the unique format of the problem addressed here, namely the stochastic graph based approach to a multi-dimensional molecular design problem. The two main types of operators are mutation and crossover. Within the category of mutation operators, three types were created: reduction, insertion, and fragment mutation. The decision between selecting either a mutation or crossover operation is based upon a split value, which is to be decided upon before the algorithm runs. This split provides a probability based selection between a mutation or crossover operator. While the mutation operations have been designed to control the size of a resultant graph, by either inserting or removing fragments if necessary, the crossover is unbiased towards the size of a child graph. Because of this, problems with a higher probability of selecting a crossover operation often have a wider size range of solutions whereas those with a higher probability of selecting a mutation operator have a more narrow size distribution. This concept is well

exemplified in Figure 3.12, which plots the size distribution as a function of the crossover probability.

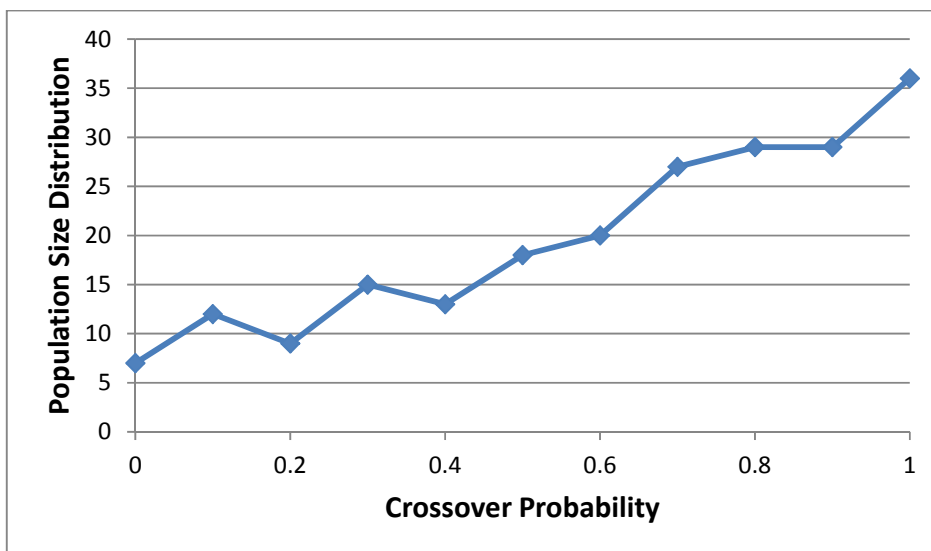


Figure 3.12 – Effect of Crossover on Size Distribution

Once a mutation has been decided upon, the probability based decision of which mutation operator to utilize is a function of the current graph size, in addition to a parameter, N_{opt} , which represents the current optimal graph size derived from previous data. N_{opt} is continuously updated as generations are analyzed and is calculated as shown in Equation 3.5, where f_i is the fitness for a previously created molecular graph and N_i is its size. This technique allows for dynamic size control, which adjusts itself to the information gained previously throughout the stochastic algorithm.

$$N_{opt} = \frac{\sum_i (f_i \cdot N_i)}{\sum_i f_i} \quad (3.5)$$

This format was chosen to maintain strict control over the size of graphs generated by the molecular design algorithm. Previous studies have revealed that physico-chemical properties often have a correlation to molecular size; for example, the boiling point of alkanes can be linearly correlated to the mass of the underlying molecular structure (Needham et al., 1987). However, it is realized that this is often not the case and the possibility to loosen control over the size of molecular graphs generated has been allowed. This can be done by controlling the β value, which characterizes the acceptable size range within which a molecular graph can be considered optimal. Once a mutation operation has been decided upon, the remaining decision between insertion, deletion, or mutation is based upon this β value by the convention shown in Figure 3.13.

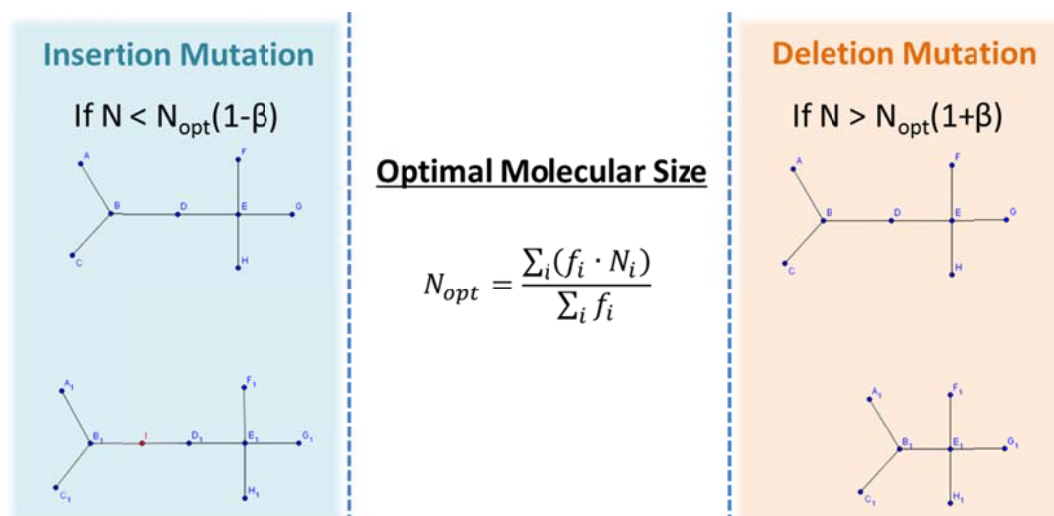


Figure 3.13– Selection of Type of Mutation Operator

Graphs within a certain percentage, β , of the optimal size undergo a signature mutation such that the size of the graph is preserved, while graphs above and below this range undergo reduction and insertion, respectively. Signature mutation involves the selection of a random node, which is then exchanged based on bond compatibility information available in the bonding network. Graph reduction involves the random selection of two or more bonded signatures, which is then replaced by a shorter path identified within the bonding network. Similarly, graph insertion involves the selection of a random bond between two signatures, in which a compatible signature is inserted to increase the size of the resultant molecular graph.

Upon selection of the crossover operator, two parent graphs are selected from the population and are cut/recombined at compatible points just as in conventional crossover operations. For a pair of bonds, one from each graph, to be compatible, the bonding network must establish that each node involved in these bonds can also be bonded to its counterpart in the other graph. Each of these operators rely heavily on information stored within the bonding network, which is why this information is pre-calculated for the signature space developed from the initial data set. The concept of a single-point crossover can be seen in Figure 3.14, where two graphs are selected and cut at a compatible bond. One fragment from each parent is then recombined in a compatible manner to produce the resultant child graph. It is worth noting that, with a single point crossover, there are two possible child graphs that can be produced and one is chosen at random with no bias to the resultant sizes.

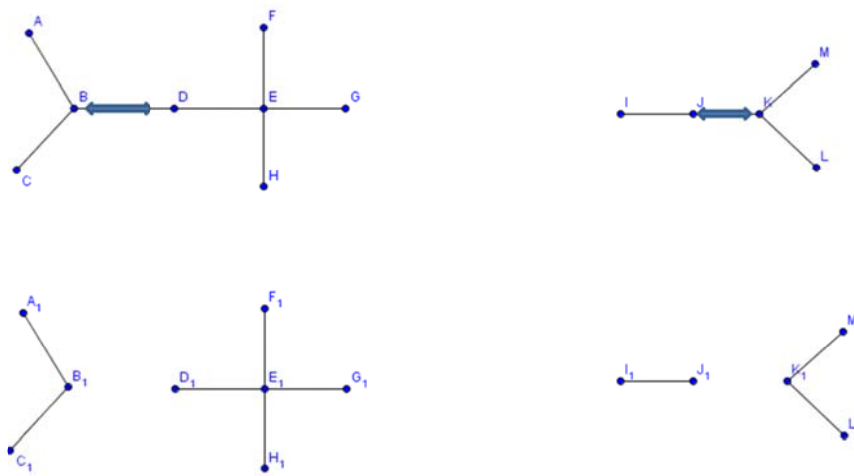


Figure 3.14 – Implementation of Crossover Operator

4. Case Studies

The following case studies are meant to exemplify the usefulness of the above proposed algorithm. The first two case studies included the design of a solvent and a fungicide, for which the substituents were designed. These examples represent an initial attempt at including spatial information within the signature descriptor and molecular design methodology presented here. The search space for both problems was small enough to be solved in a deterministic manner, however, the nature of the problem definition allowed for a linear optimization approach to be taken. Utilizing optimization would allow for a much larger search space to be considered. In addition, the nature of the resulting solutions, being smaller in size on a molecular level, allowed for the spatial information for each potential solution to be derived from a single atomic signature, often the largest one. This is an inherent limitation in the solution of CAMD problems, and the technique for solving studies with larger solutions has been described earlier. This necessitates the development of a global geometry from fragment geometries. Unfortunately, the nature of these calculations does not allow for a linear approximation of the geometry. In addition, most any spatial descriptor utilized is not a linear function of the atomic coordinates or interatomic distances. This being said, what remains is to find an approach whereby the potential energy surface of candidate molecules can be estimated without conducting time consuming scans of this space. This would allow for the consideration of much larger search spaces. The third case study presented analyzes the ability of the proposed algorithm to estimate the PES for a given molecule. This verification was necessary for further studies utilizing this methodology. Once verified, the next step was to consider a study with a much larger search space. This is where the fourth case study,

which included the design of non-peptide mimetics matching a pharmacophore model, came in to play. The development of this dissertation can be seen in the consideration of case studies with increasing complexity, as they have been presented in this section.

4.1 Solvent Design Study

The initial case study is simplistic in nature and represents the design of a solvent with desired properties. The target in this design is to identify an alkane molecule with minimum soil sorption coefficient ($\log K_{oc}$). The property constraints to be satisfied by the solvents are boiling point (BP) and toxic limit concentration (TLC). The constraints are listed in Table 4.1 and the property models are described in Table 4.2.

Table 4.1 - Property constraints for solvent design study.

Property	Upper Bound	Lower Bound
Boiling Point, BP (°C)	85	55
Log(TLC) (ppm)	-	1
Log(K_{oc})	Minimum	

Table 4.2 - Property models for solvent design study.

Property	Property Model
$\log(K_{oc})$	$\log(K_{oc}) = 0.53(\chi^1) - 1.25(\Delta^1 \chi^v) - 0.72(\Delta^0 \chi^v) + 0.66$
Boiling Point, <i>BP</i>	$BP = 393(3D W)^{0.0986} - 682$
$\log(TLC)$	$\log(TLC) = 4.066 - 0.9915(\chi^1)$

For boiling point, a QSPR (Mihalic and Trinajstić, 1991) that makes use of the 3D Wiener index has been used. The three-dimensional Wiener index (${}^3D W$) is computed directly from the geometric distance matrix as follows, where $d_{i,j}$ is the geometric distance between atoms i and j :

$${}^3D W = \frac{1}{2} \sum_{i,j} d_{i,j} \quad (4.1)$$

The property model correlating the first order valence connectivity index to toxicity was developed by Koch (1982). The objective constraint in this study was to design a molecule with a minimal soil sorption coefficient. K_{oc} represents the soil sorption coefficient, which is a strong indicator of the fate of an organic chemical introduced into the environment. A lower value here would represent a decrease in the potential harm this solvent could do if released into the soil. The model (Bahnick and Doucette, 1988) utilized to estimate this property has been correlated to various molecular connectivity indices and the optimal model is shown in Table 4.2. For this study, since only alkane structures were considered, the delta valence connectivity indices within this equation are equal to zero.

To form molecular building blocks, signatures of height 2 are used. There are 65 unique molecular signatures of height 2 that can be developed from linear alkane structures. These signatures have been listed in Table 4.3 below.

Table 4.3 - All height-2 atomic signatures for linear alkanes.

C1(C2(C))	C4(C4(CCC)C4(CCC)C4(CCC)C1)
C1(C3(CC))	C4(C4(CCC)C4(CCC)C3(CC)C3(CC))
C1(C4(CCC))	C4(C4(CCC)C4(CCC)C3(CC)C2(C))
C2(C2(C)C1)	C4(C4(CCC)C4(CCC)C3(CC)C1)
C2(C3(CC)C1)	C4(C4(CCC)C4(CCC)C2(C)C2(C))
C2(C4(CCC)C1)	C4(C4(CCC)C4(CCC)C2(C)C1)
C2(C2(C)C2(C))	C4(C4(CCC)C4(CCC)C1C1)
C2(C3(CC)C2(C))	C4(C4(CCC)C3(CC)C3(CC)C3(CC))
C2(C4(CCC)C2(C))	C4(C4(CCC)C3(CC)C3(CC)C2(C))
C2(C4(CCC)C4(CCC))	C4(C4(CCC)C3(CC)C3(CC)C1)
C2(C3(CC)C3(CC))	C4(C4(CCC)C3(CC)C2(C)C2(C))
C2(C4(CCC)C3(CC))	C4(C4(CCC)C3(CC)C2(C)C1)
C3(C4(CCC)C4(CCC)C4(CCC))	C4(C4(CCC)C3(CC)C1C1)
C3(C4(CCC)C4(CCC)C3(CC))	C4(C4(CCC)C2(C)C2(C)C2(C))
C3(C4(CCC)C4(CCC)C2(C))	C4(C4(CCC)C2(C)C2(C)C1)
C3(C4(CCC)C4(CCC)C1)	C4(C4(CCC)C2(C)C1C1)
C3(C4(CCC)C3(CC)C3(CC))	C4(C4(CCC)C1C1C1)
C3(C4(CCC)C3(CC)C2(C))	C4(C3(CC)C3(CC)C3(CC)C3(CC))
C3(C4(CCC)C3(CC)C1)	C4(C3(CC)C3(CC)C3(CC)C2(C))
C3(C4(CCC)C2(C)C2(C))	C4(C3(CC)C3(CC)C3(CC)C1)
C3(C4(CCC)C2(C)C1)	C4(C3(CC)C3(CC)C2(C)C2(C))
C3(C4(CCC)C1C1)	C4(C3(CC)C3(CC)C2(C)C1)
C3(C3(CC)C3(CC)C3(CC))	C4(C3(CC)C3(CC)C1C1)
C3(C3(CC)C3(CC)C2(C))	C4(C3(CC)C2(C)C2(C)C2(C))

C3(C3(CC)C3(CC)C1)	C4(C3(CC)C2(C)C2(C)C1)
C3(C3(CC)C2(C)C2(C))	C4(C3(CC)C2(C)C1C1)
C3(C3(CC)C2(C)C1)	C4(C3(CC)C1C1C1)
C3(C3(CC)C1C1)	C4(C2(C)C2(C)C2(C)C2(C))
C3(C2(C)C2(C)C2(C))	C4(C2(C)C2(C)C2(C)C1)
C3(C2(C)C2(C)C1)	C4(C2(C)C2(C)C1C1)
C3(C2(C)C1C1)	C4(C2(C)C1C1C1)
C4(C4(CCC)C4(CCC)C4(CCC)C4(CCC))	
C4(C4(CCC)C4(CCC)C4(CCC)C3(CC))	
C4(C4(CCC)C4(CCC)C4(CCC)C2(C))	

The signatures used as building blocks to develop candidate molecular structures were optimized at the AM1 (Austin Model 1) (Dewar et al., 1985) quantum mechanical level, as it was shown to generate reasonable bond lengths and angles for the chosen data set. In the context of chemical graph theory, these signatures are described as sub-graphs and the optimized Cartesian coordinates of each atom were used to derive interatomic Euclidean distances. Together, these interatomic distances represent individual entries in the symmetric geometric distance matrix. An optimization problem has been set up using the given property models with relevant structural constraints and solved for the minimum value of soil sorption coefficient. The best three candidates and their respective atomic signatures have been listed in Table 4.4.

Table 4.4 - Solutions for solvent design study.

Molecule	Atomic Signatures	BP (°C)	log(TLC) (ppm)	log(K _{oc})
CH ₃ CH ₂ CH(CH ₃)CH ₂ CH ₃	2x C1(C2(C3)) 2x C2(C1C3(C1C2)) 1x C1(C3(C2C2)) 1x C3(C1C2(C1)C2(C1))	62.3	3.4	1.54
(CH ₃) ₂ CH(CH ₂) ₂ (CH ₃)	2x C1(C3(C2)) 1x C3(C1C1C2(C2)) 1x C2(C3(C1C1)C2(C1)) 1x C2(C1C2(C3)) 1x C1(C2(C2))	61.0	2.3	1.72
(CH ₃) ₂ CHCH(CH ₃) ₂	4x C1(C3(C1C3)) 2x C3(C1C1C3(C1C1))	58.1	1.9	1.76

4.1.1 Conclusions

The solvent design case study presented here represents an initial attempt at utilizing signature descriptors containing geometry information for the solution of molecular design problems with descriptors of varying dimensionality. The signatures themselves were optimized as isolated fragments without consideration of surrounding or attached atoms. In this study, only the minimum energy conformers were accepted for consideration within the optimization problem. The nature of the solution space allowed for the complete molecular geometry of structures considered to be determined from the largest atomic signature within the molecular signature vector. While the approach may be feasible within this study, where the three-dimensional descriptor was not a strong function of the finer details of the conformational information available, improved methodologies are necessary for more complex applications. Studies with larger

structures existing within the chemical search space will require the development of global molecular geometry from several fragments. In addition, there may be many local conformational minima possible with spatial descriptors that are very sensitive to these varying atomic arrangements. For these reasons, the following studies will consider such cases along with the methodologies useful for tackling problems with increased complexity.

4.2 Design of Alkyl Substituent for Rice Plant Fungicide

Application of the molecular signature descriptor in accounting for topological, topographical and information indices is illustrated through the optimal substituent selection for dialkyldithiolanylidenemalonate (DD). DD has been shown to have eradicant and protectant activity against rice blast disease. Uchida (1980) enumerated the effectiveness of this fungicide in terms of affinity ($\log(VE)$), mobility ($\log(\mu)$) and retention ($\log(R/(100-R))$). These three attributes have been linearly related to the lipophilicity ($\log(P_{oct/wat})$) of the chosen substituents. A QSPR was developed (Basak et al., 1996) to model $\log(P)$ as a function of several different descriptors. Index values are calculated only for the substituent regions of the fungicide. The structure of this fungicide is shown in Figure 4.1.

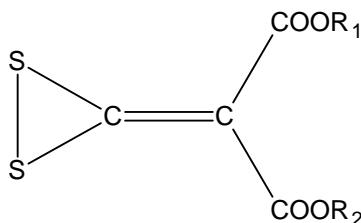


Figure 4.1 - Fungicide structure.

Raman and Maranas (1998) previously visited this problem while correlating $\log(P)$ values to the first order molecular connectivity index. The same upper and lower bounds on mobility and retention are applied in this study, while the objective function is to maximize substituent affinity to the rice plant. These property constraints can be found in Table 4.5. Property models utilized in this study are shown in Table 4.6. The information content (IC) indices infer a measure of molecular symmetry, and this formulation was originally introduced by Shannon (1948). Another information theoretical invariant utilized in the $\log(P)$ QSPR is the complementary information content (CIC) index (Magnuson et al., 1983).

Table 4.5 - Fungicide substituent property constraints.

Property	Upper Bound	Lower Bound
Retention, $\log(R/100-R)$	-2.04	-2.48
Mobility, $\log(\mu)$	0.3	-0.3
Affinity, $\log(V_E)$	Maximum	

Table 4.6 - Fungicide study property models.

Property	Property Model
Retention	$\log(R/100-R)=0.72*\log(P)-1.93$
Mobility	$\log(\mu)=-0.64*\log(P)+1.95$
Affinity	$\log(V_E)=0.53*\log(P)-0.24$
Hydrophobicity	$\log(P)=-5.60+0.19(P_{10})-1.46(IC_0)+1.09(CIC_2)-0.77(CIC_3)-1.36({}^6\chi^b)+5.34({}^0\chi^v)-3.41({}^1\chi^v)+0.55({}^4\chi^v)-0.41({}^3\chi^v_C)+1.10(V_W)-0.17({}^3DW)$

The atomic signatures used in this study were hydrogen suppressed and were allowed to exhibit one degree of unsaturation such that a bond to the existing fungicide structure could be allowed. Given the above property constraints, each estimated through property models utilizing varying descriptor types, the top three solutions were identified as shown in Table 4.7, which shows the respective molecular signature vectors.

Table 4.7 - Molecular signature solutions to fungicide problem.

Atomic Signatures	Occurrence #'s
C1(C2)	2
C1(C3)	1
C2(C2C1)	1
C2(C3C1)	1
C2(C3C2)	1
C3(C2C2C1)	1
C1(C2)	1
C1(C3)	3
C2(C3C1)	1
C3(C3C2C1)	1
C3(C3C1C1)	1
C1(C2)	2
C1(C4)	2
C2(C4C1)	2
C4(C2C2C1C1)	1

As mentioned previously, each set of atomic signatures can refer to more than one molecular signature. This is the case for solutions in the fungicide substituent design study since several solutions can actually satisfy more than one resulting structure. The

structural isomers have been enumerated for each solution, along with their respective estimated properties in Table 4.8.

Table 4.8 - Substituent solution isomers.

Properties			R1	R2
Affinity	Mobility	Retention		
1.480	-0.292	-2.094	methyl methyl ethyl	3-methyl-butyl 2-pentyl <i>sec</i> -butyl
1.521	-0.193	-2.326	methyl	2-methyl-2-butyl
1.704	-0.138	-2.163	methyl	<i>tert</i> -pentyl

4.2.1 Conclusions

This study was an extension of the previous solvent design study in that it considered more complex property models. Several of the solutions identified within the first study (Chemmannattuvalippil, 2008) were also found in this case study. However, the inclusion of spatial properties allowed for a different perspective on each property relationship, which was not captured in the previous application. In addition to spatial descriptors, information theoretic descriptors were also included in this analysis. While the solution was deterministic in nature, the problem could have been potentially solved utilizing linear programming techniques. This problem also addressed the issue of structural isomers arising from degeneracy seen in the use of atomic signatures as a platform deriving all other descriptors. The next step will be to solve problems in which the

solutions are large enough to require geometry development from multiple fragments. However, the proposed technique will need to be verified first to ensure that reasonable molecular geometries are produced.

4.3 Geometry Estimation Technique Analysis

The solution of problems relying on spatial, or three-dimensional, descriptors for structure characterization necessitates the consideration of molecular geometries. The most straight-forward and thorough approach is to carefully evaluate the potential energy surface (PES) of each structure considered within the search space for a given molecular design problem. However, this approach is computationally demanding and limits the region of chemical space that can be searched. While this technique is feasible for lead optimization stages in drug design studies, where the potential candidates have been substantially narrowed down to just a few structures, its application in many virtual screening situations becomes limited. As such, techniques for quickly estimating the PES of candidate molecules, or perhaps identifying likely conformational minima, become beneficial in the solution of these problems. The simplest approach, and most limited in producing realistic geometries, would be to utilize a database of conventional bond lengths and angles etc. to estimate the resulting geometry. Another technique, as discussed in the first case study on solvent design, could be to use fragment geometries to develop a global molecular geometry. The limitations of this approach have been addressed. To overcome these limitations, an approach has been developed which applies the same concept, however, with more detail. The fragment based approach for developing an estimation of the PES for structures considered within a CAMD problem has been explained in the background section of this dissertation. However, what remains

is verification of this technique. The approach taken to verify the aforementioned methodology will be as follows:

1. Choose a data set on which to develop spatial atomic signatures.
2. Leave out structures within the chemical space of the test set from step one for comparison.
3. Develop an estimated set of conformers from the initially obtained atomic signatures using the proposed methodology.
4. Independently develop a set of local energy minima, utilizing the same level of theory for geometry optimization, with which to compare against the estimated PES.
5. Compare the two sets of conformational isomers for structures in the test set to see what percentage of this information was captured in the PES analysis.

4.3.1 Analysis of Methodology in Organic Space

The initial verification of the geometry development methodology was chosen for a data set of organic structures. Within the data set exists linear alkane structures, as well as structures containing double bonds and branched alkanes. This data set is chosen such that the effect of the proposed technique could be studied on structures with varying potential energy surfaces, with some structures having less conformational flexibility than others. The data used to develop the respective atomic signature basis set can be seen in Figure 4.2, with the hydrogen-suppressed structures shown. In addition, the test set is shown in Figure 4.3.

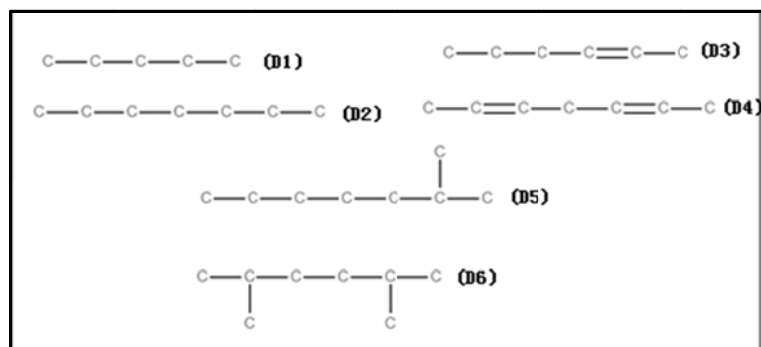


Figure 4.2 – Geometry verification data set.

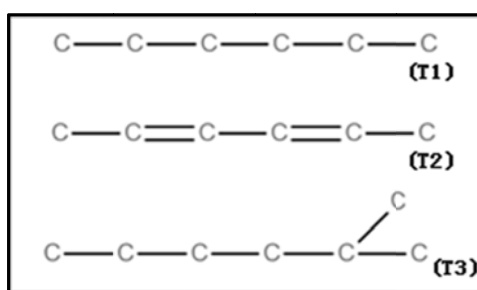


Figure 4.3 – Geometry verification test set.

Both sets of structures were initially drawn into Avogadro and optimized using molecular mechanics with the MMFF94 force field (Halgren, 1996), which was chosen for its ability to reproduce accurate geometries for alkanes and alkenes. The steepest descent algorithm was utilized with 10000 steps and a convergence criterion of 10^{-7} . This was the technique used to develop a consistent initial geometry, which was utilized as input to the conformational analysis step. This initial geometry, in the form of a .mol file, was used as input to create a z-matrix file for which a subsequent Monte Carlo conformational search could be performed within the BOSS program (Jorgensen and Tirado-Rives, 2005). The Monte Carlo method has been shown to quickly and effectively

explore the conformation space for a range of molecular structures (Chang et al., 1989). The conformational search began with 100 starting structures and those with a conformational strain energy within 5 kcal/mol of the lowest conformer identified were kept. Table 4.9 lists the number of conformers identified in this range for each molecule in the data and test sets.

Table 4.9 - Conformers identified with MC search.

Structure (type)	Conformers Identified (within 5 kcal/mol)
T1 (alkane)	7
T2 (alkene)	5
T3 (branched)	9
D1 (alkane)	4
D2 (alkane)	9
D3 (alkene)	5
D4 (alkene)	5
D5 (branched)	16
D6 (branched)	9

As can be seen in Table 4.9, the branched structures typically have a more complicated potential energy surface. This could partially be a result of the fact that the branched alkanes have more atoms, on average, than the linear structures used in this study. However, the trend of alkenes having a less complex potential energy surface, identified by the lower number of conformers found during a comparable PES scan, is consistent here. This is likely a result of the double bonds decreasing the rotational

degree of freedom for a given alkene structure when compared to a similar structure with single bonds. A visualization of the potential energy surface for butane and butene, for one degree of freedom, can be seen in Figure 4.4. This reveals the increased complexity associated with the higher degrees of freedom, resulting from more rotatable bonds. This can be extended, when considering that a branched alkane would have more bonds to rotate around, to hypothesize that their potential energy surface would be relatively more complex than similar alkenes or alkanes.

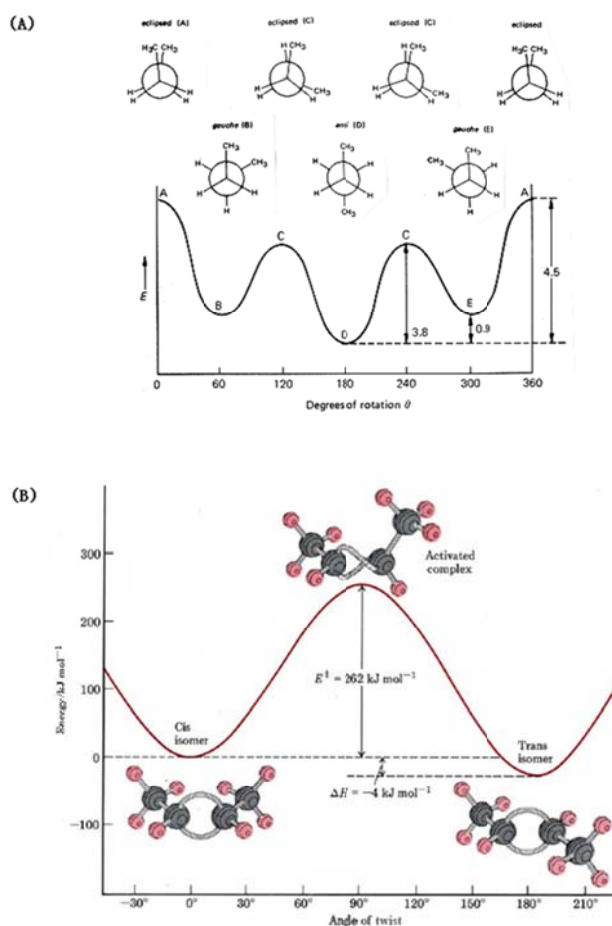


Figure 4.4- Example potential energy diagrams for (A) butane and (B) butene

This geometry estimation methodology was shown to be successful in identifying the likely local energy minima associated with a set of alkanes and alkenes. At a cutoff criterion of 5.5 angstrom and a conformer comparison criterion of 3.2 angstrom, the method was able to reproduce all 21 conformers identified within the previous study. In addition, it is helpful to consider the results in Figure 4.5. This chart shows the number of spatial atomic signatures left after compression using various cutoff values (which discards conformers for being too similar). A larger cutoff value would result in more conformers being discarded and a smaller value would maintain more of them for a more complete description of the PES of each atomic signature.

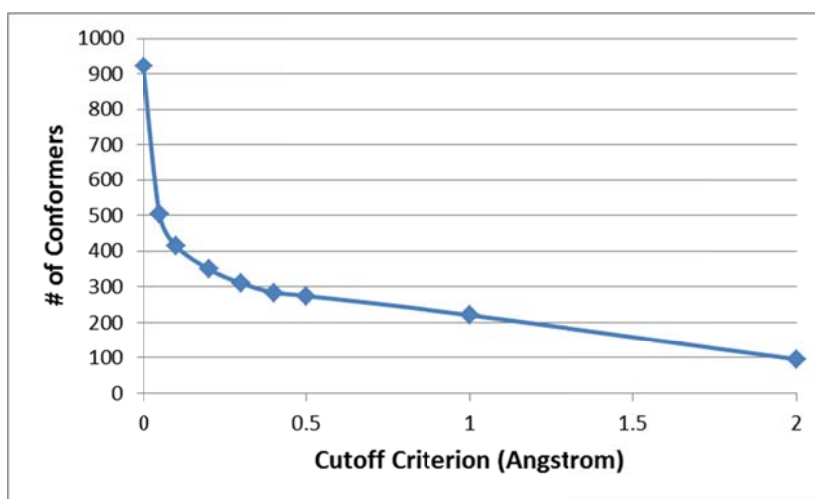


Figure 4.5 - Conformers after compression.

This approach represents a convenient estimation of the PES for various molecules in an expedited manner. Such a technique will prove useful in the solution of CAMD problems requiring spatial information, such as those using property models with three-dimensional descriptors. However, it is necessary to perform some degree of sensitivity analysis based on the nature of a chosen data set. Larger, more complex structures could

have a much more complicated potential energy surface and a smaller cutoff criterion for developing spatial atomic signatures would be required. This can be done by selecting a few structures, representative of the chosen data set, and running the algorithm discussed above, which tests how many conformers can be recreated for a given cutoff. There is an inherent ‘sweet spot’ that is specific to each data set chosen and must be identified such that the algorithm is most effective. This could be done by repeating the previously covered geometry verification methodology until a desired number of conformational isomers are captured with the chosen cutoff value.

4.4 Structure Based Design of Non-Peptide Mimetics

This contribution outlines an algorithm for the design of mimetics based on information from existing pharmacophore models. Ideally, these techniques could be implemented alongside conventional high-throughput screening (HTS) efforts to alleviate the time and costs required to develop new therapeutic drugs with improved processability. This study represents a first attempt at solving problems necessitating geometry estimation from fragment geometries.

4.4.1 Pharmacophore and Non-Peptide Mimetics

The IUPAC definition for a pharmacophore is given as ‘... *the ensemble of steric and electronic features that is necessary to ensure the optimal supramolecular interactions with a specific biological target structure and to trigger (or to block) its biological response*’ (Wermuth, 1998). These models can be developed with or without geometric characterization of the targeted receptor. If the receptor structure is known, several computer-assisted docking techniques can be utilized to develop the pharmacophore model. When the receptor structure is unavailable, there are various superpositioning

techniques available for comparing the spatial features found in ligands and using this information for model development. These alignment techniques are often limited by their inability to capture the conformational flexibility of ligands under consideration. Several algorithms have been developed to overcome this limitation and most are based on pre-calculation of ligand conformations, and/or distance geometry (e.g. the geometric distance between important structural/electrostatic features) (Wolber et al., 2008). Many attempts at utilizing pharmacophore models to develop novel mimetics have focused on replacing the peptide backbone with a non-peptidic framework, or identifying cyclic peptide derivatives (Olson et al., 1993). The limitation of these solutions in regards to oral-bioavailability, drug-likeness, and stability leads to the consideration of alternative approaches. A trend towards models built with more generalized features and a thorough conformational analysis, with rotational and translational invariance, could allow for the extension of in-silico screening to include potential non-peptide candidate mimetics. Properties related to drug-likeness, such as those established in Lipinski's rule of 5, provide a quick estimation as to the potential pharmacokinetics in the human body as well as other important properties like absorption, distribution, metabolism and excretion (ADME) (Lipinski et al., 2001). These rules can place limits on something as simple as the molecular weight or as complex as the molar refractivity or molecular polar surface area.

4.4.2 Model Information

The case study chosen to test this algorithm is based on a 3D pharmacophore model developed for 5-Hydroxytryptamine₆ (5-HT₆) receptor antagonists (Lopez-Rodriguez et al., 2005). Evidence suggests that this receptor may be involved in memory impairment,

psychosis, convulsive disorders, appetite control, and other related central nervous system diseases. This pharmacophore model was developed with the pharmacophore development software called Catalyst (Kurogi et al., 2001) from a training set of 45 structurally diverse antagonists and the optimal model is represented Fig. 4.4. Four regions were identified (Lopez-Rodriguez et al., 2005) to be necessary for optimal receptor blocking, and these include: a hydrophobic site (HYD), an aromatic ring hydrophobic site (AR), a positive ionizable atom (PI), and a hydrogen bond acceptor (HBA). Respective euclidean distances (Å) between model features are also shown.

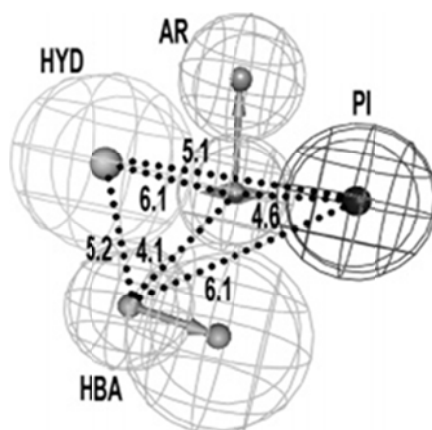


Figure 4.6 - Pharmacophore Model for 5-HT6.

4.4.3 Atomic Signature Development

A sub-set of 22 structures, chosen to represent diversity found in the original pharmacophore training set, was chosen as the basis set for the development of spatial atomic signatures to be utilized in the in-silico design of a potential 5-HT6 antagonist with improved bioavailability. These structures were initially drawn into the Accelrys Draw 4.0 program (Accelrys Inc., 2001) and the resulting 2-Dimensional MDL mol file was imported into Avogadro for estimation of an initial geometry, for which the Merck

Molecular Force Field (Halgren, 1996) was utilized as a suitable candidate to develop a rough initial geometry. A Monte Carlo conformational search, with 200 starting geometries, was performed for each structure and all isomers within 20 kcal/mol of the lowest energy conformer were accepted for further calculations. This conformational analysis was performed in the Biochemical and Organic Simulation System (BOSS) software (Jorgensen and Tirado-Rives, 2005) through application of the xCS200 script, which performs a conformational analysis with 200 starting structures. From the original 22 structures, 508 conformational isomers were ultimately accepted. In house software, facilitated through the use of Faulon's signature software (Faulon, 2014), was developed for the conversion of input mol files, for each conformational isomer, to their respective height 3 atomic signatures. This software was written in Python and the results for the signature software were used as input to the program. From the 23,368 atomic signatures derived from the data set, 254 of these were identified as being unique structural isomers. Upon bonding network generation, about 2.96% of the pairwise signatures were identified as being capable of bond formation. Conformational isomerism data was stored in the respective structural isomers as a dictionary of Cartesian coordinates representing the relative positions of all atoms in each conformer. The measure of similarity used for comparing conformational isomers, or graph isomorphisms, was the root mean square difference in inter-atomic distance values. By setting a limit of 0.2 Å as the maximum acceptable difference between equivalent inter-atomic distances in conformational isomers, the number of signature fragment conformers was reduced by 73%, from around 23,000 to 6,340. This reveals a high degree of spatial similarity between conformational isomers for each type of signature. Since these atomic signatures were derived from a set

of structurally diverse molecules, this provides some verification for the reasonability of using fragment geometries to build global molecular geometries.

4.4.4 Combinatorial Optimization

The applicability domain for this study was defined in terms upper and lower bounds placed on the signature occurrence vector. These bounds were set between zero and the maximal number of repeats for a specific atomic signature as identified in the basis set. The Python itertools module was utilized to generate all possible combinations of atomic signatures within the AD defined. Each molecular signature vector encountered was initially tested against structural feasibility constraints and subsequently against the multi-dimensional criteria covered in section 3.3. Some example of structural fragments identified as meeting the various pharmacophore sites are shown in Figure 4.5.

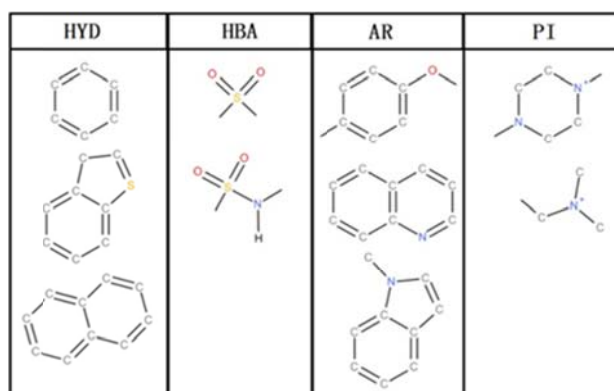


Figure 4.7- Example pharmacophore groups.

A five percent relaxation was placed on spatial criteria in the pharmacophore model to allow for potentially strained geometries to be considered. For each conformational isomer identified as meeting the required constraints a single point energy calculation was performed on the estimated geometry. In addition, this initial geometry underwent a

molecular mechanics energy minimization, utilizing the Austin Model 1 (AM1) (Dewar et al., 1985) semi-empirical force field, after which another single point energy calculation was made. A ratio of minimized strain energy to the strain energy calculated from the initially developed geometry was utilized as a metric with which to rank the candidate solutions. The idea here is to award candidate structures for lying near a local energy minimum, which would indicate an increased likelihood of achieving this geometry in solution. The final number of structures meeting the multidimensional criteria placed on this study was 22. Several of the highest ranked candidates were conformational isomers of structures used in the test set. The top three isomers exhibiting acceptable spatial characteristics, as well as a favorable measure of single-point energy with respect to the nearest local conformational minima have been shown in Figure 4.8. All of the solutions identified for this case study, along with the metric utilized to rank them, can be found within

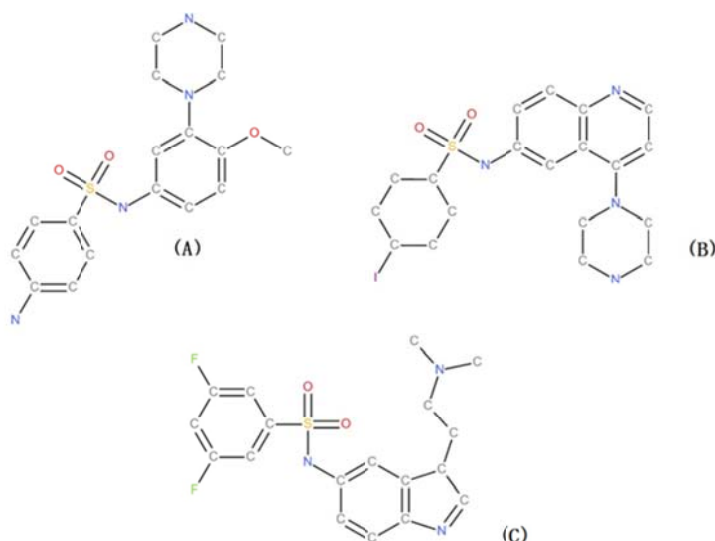


Figure 4.8 - Solutions to antagonist design case study.

4.4.5 Conclusions

The following methodology has been developed for the solution of multi-dimensional inverse molecular design problems while quickly estimating the conformational space of each structural isomer presented. The use of a fragment-based, spatial Signature descriptor was chosen for its compatibility with such a combinatorial algorithm aimed at scanning a large region of chemical space. The algorithm takes the approach of satisfying lower dimensional criteria first so that the computational expenses associated with developing/analyzing more complex criteria, such as that associated with geometric analysis, is minimized. Flexibility in the use of varying data sets for spatial signature development could allow for the application of this methodology at various stages of molecular design, or more specifically drug development.

The case study chosen to exemplify the benefits of this approach was the design of a receptor antagonist with potential therapeutic benefits. Several candidate solutions were identified that were not part of the initial training set. Based on the signature height used in this study, and the nature of the data set with many cyclic structures, the structures tested were fairly similar to the original set.

4.5 Solvent Design with Genetic Algorithm

The case study chosen to exemplify this stochastic approach involves the design of a molecule with a specified boiling point temperature. Basak et al. (1996) developed a structure activity relationship correlating various 2D and 3D molecular descriptors to the normal boiling point for a data set of 1023 chemicals from the Toxic Substances Control Act (TSCA) Inventory. Only those molecules with a listed normal boiling point value and where the hydrogen bonding potential was estimated to be equal to zero were chosen.

The goal of this study was to determine the optimal combination of descriptors between 2D, 3D, and 2D+3D. Most of the topological indices for each chemical within the chosen dataset were calculated utilizing the computer program POLLY (Basak et al., 1988). Because of the varying nature of descriptor values, topological indices were transformed by the natural logarithm of the index plus one. One was added since many of the indices were found to be zero. Geometric parameters were transformed by the natural logarithm of the parameter. Two different regression techniques were utilized. When the number of independent variables was large, stepwise regression was chosen. However, when the number of independent variables was small, all possible subsets regression was used. To include both sets of descriptors, each capturing varying dimensionality, the following model development procedure was chosen. First, only topological indices were utilized to identify the best model. The topological indices utilized within this model were then added to a set of topochemical indices and the best model from this combined set of indices was developed. Finally, the best topological/topochemical indices were then added to the set of geometrical descriptors, from which an optimal model was once again created using a subset of these.

The model containing only topological indices utilized 11 parameters and resulted in an explained variance (R^2) of 80.8% and standard error (s) of 40.9°C. With the addition of topochemical parameters, a model using two topological and seven topochemical parameters was identified as best, having an R^2 of 96.5% and s of 17.4°C. The best fit model, shown in Equation 4.2, resulted from a combination of 2D topological, 2D topochemical, and 3D descriptors with an R^2 of 0.967 and an s of 16.8°C.

$$\begin{aligned}
 BP & \\
 &= -285.7 + 125.3(^6\chi) + 10.9(P_{10}) + 74.5(IC_0) \\
 &- 125.0(^6\chi^b) - 86.3(^3\chi^b_c) + 175.3(^0\chi^v) + 49.1(^2\chi^v) + 18.7(^5\chi^v_{pc}) - 9.1(^{3D}W_H) + 8.1(^{3D}W)
 \end{aligned}
 \tag{4.2}$$

This model contains descriptors of varying nature, each capturing a unique aspect of molecular architecture. As far as topological indices, the sixth order path connectivity index (${}^6\chi$) and number of paths of length 10 (P_{10}) are included. For topochemical indices, the information content based on the zeroth order neighborhood (IC_0), the sixth order bond path connectivity index (${}^6\chi^b$), the third order bond cluster connectivity index (${}^3\chi^b_c$), the zeroth order valence path connectivity index (${}^0\chi^v$), the second order valence path connectivity index (${}^2\chi^v$), and the fifth order valence path-cluster connectivity index were utilized. As for geometric indices, the three-dimensional Wiener index (${}^{3D}W$) was used in both its hydrogen suppressed and hydrogen inclusive format. The summation of this information was found to be optimal in capturing the variance seen within the data set with respect to boiling point. This situation fits the criteria of the established method in that descriptors of varying dimensionality have been utilized for a large set of molecular structures. As such, it was chosen as being suitable for the description of boiling point and used within the stochastic molecular design framework previously established.

4.5.1 Development of Spatial Signatures

A subset of 245 molecules was chosen from the initial 1023 utilized to develop the boiling point property model with the aim of maintaining the original variance in structural features. These structures were initially drawn into the Avogadro molecular modelling program and were quickly optimized to provide a starting point geometry. The force field utilized for this initial optimization was MMFF94, which was chosen by its

ability to produce reasonable geometries for the chosen data set, and the steepest descent algorithm was applied for 10,000 steps or until a convergence of 10^{-7} was met. This step was fairly quick with an average optimization time of about 4-6 seconds. The method utilized for conformational analysis, within the BOSS molecular modelling software, was the AM1 molecular mechanics force field. A conformational analysis was performed for these molecules with an acceptance criterion of each conformer being within 15 kcal/mol of the identified conformational minimum. These conformers were dissected into 194 unique height-2 atomic signatures, or structural isomers. The conformational information for each signature was further compressed by removing conformers exhibiting a similarity limited to 0.2 angstrom for each pairwise inter-atomic distance comparison.

4.5.2 Parameters Utilized

The parameters necessary for this design problem include a lower (10) and upper (25) bound placed on the number of atoms allowed in candidate molecular graphs. In addition the lower and upper limits on acceptable boiling points were placed at 75 and 80 °C, respectively. The steady state population size was chosen to be 100 graphs and each run, of which there were 10 total, continued for 100 generations. The probability of mutation and crossover operators were both set to 0.5. The β variable discussed in section 2.3.2 was set to 0.15 and the gaussian fitness decay rate, α , was set to 0.1. The β variable was chosen such that the size of solutions could be preserved within a reasonable range, which was chosen to be within 15 percent of the identified optimal size. The gaussian fitness decay rate, α , was chosen with consideration of the expected distribution in potential solutions. The variable essentially determines how quickly a solution's fitness value drops off as it leaves the desired property range. In this case, the desired property

range was very small, being only a difference of 5 °C. As such, a larger α value created a fitness function which was ‘harsh’ enough to discriminate between very close property values for such a short property range. This effect can be seen in Figure 3.11. The probability of selection between mutation and crossover operators was chosen to be 0.5 based upon an initial size control study. Within this study, the probability was varied between 0.1 and 0.9 and the criterion for the algorithm was to approach a desired preset molecular size, from which fitness was derived based upon closeness to this size. It was found that larger probability values, which favored crossover operations more heavily, were less effective at guiding the algorithm towards this desired size. The reason for this is that the crossover operator has no discrimination towards the size of the resultant molecular graph. As such, it was also found that a value of 0.5 was acceptable at driving the algorithm towards the optimal size while still visiting a reasonable size range around the optimal size.

4.5.3 Results and Conclusions

The exhaustive combinatorial search for solutions in a chemical space of this size would have been much more time consuming, whereas the genetic algorithm applied here was able to identify satisfactory solutions more quickly. This approach could benefit from inclusion of new types of mutation operators; however the techniques utilized here were effective in controlling the size and diversity of solutions generated. The following graph, shown in Figure 4.9, exemplifies the typical progression of average population fitness as a function of the generation number. A trend of steadily increasing fitness can be seen as the progression of mutation and mating between previous generations is facilitated through the various operators.

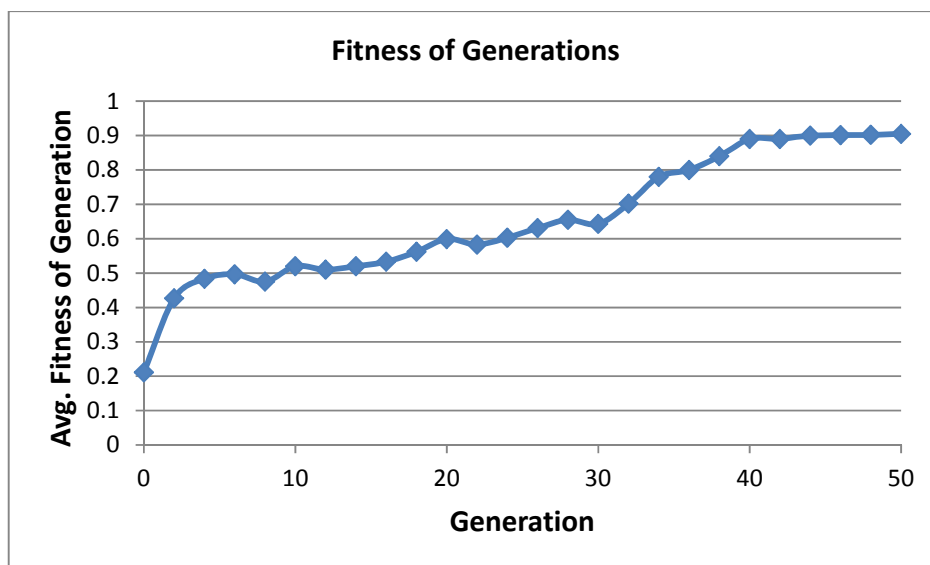


Figure 4.9 – Fitness as a Function of Generation

Many of the solutions were repeated throughout the algorithm, leading to a smaller number of solution structures than initially estimated, and the average number for the ten trials conducted was around 20. Some of the more common solutions have been shown below in Table 4.10.

Table 4.10 – Common Solutions Identified During Solvent Design Case Study

Chemical	Estimated Boiling Point (°C)	Experimental Boiling Point (°C)	Relative Difference (%)
carbon tetrachloride	75.9	76.8	1.2
ethanol	78.7	78.5	0.3
2-methyl-1,3-pentadiene	76.4	79.0	3.3
2,2,3-trimethylbutane	80.0	80.9	1.1
2-butanone	79.5	79.6	0.1
1,3,5-hexatriene	78.8	78.0	1.0

1,2-hexadiene	75.0	76.0	1.3
ethyl acetate	77.3	77.0	0.4
2,2-dimethylpentane	77.8	79.0	1.5
2,4-dimethylpentane	80.1	81.0	1.1
2,2,2-trifluoroethanol	79.3	78.0	1.7
2,3,3-trimethyl-1-butene	78.3	79.0	0.9
acetonitrile	79.7	81.6	2.4
2-methyl-2-propanol	79.5	82.2	3.3
isopropyl alcohol	78.0	82.5	5.6
4,4-dimethyl-2-pentene	79.4	79.0	0.5
3,4-dimethyl-1-pentene	78.1	80.0	2.4
1-butanal	75.5	74.8	0.9
2,3-dimethyl-2-butene	76.7	73.0	4.9

A stochastic molecular design algorithm, in the form of a genetic algorithm, was chosen for solving problems containing large search spaces with complicated multi-dimensional descriptor sets. The technique was successful in identifying several unique solutions for the problem at hand. Several parameters, including α , β , and the Gaussian decay rate variable, must be optimized before allowing the algorithm to run to completion. This would ensure that the search is optimized for the specific property models utilized as well as the search space chosen. Overall, this method was found to be promising at solving such complicated CAMD problems and the development of more complex operators,

including a double point crossover, could prove to be beneficial at tackling more diverse data sets.

5. Conclusions and Future Direction

A methodology for the solution of computer-aided molecular design problems with multi-dimensional characterization has been presented within this dissertation. This technique has allowed for the solution of such problems in a more efficient manner when compared to many of the techniques previously utilized. The reason for this is that molecular design problems utilizing varying property models with descriptors of different dimensionalities can now be solved on a common platform. This provides the freedom to choose the optimal property models for each problem without limitation to the types of descriptors utilized. The signature descriptor, while initially developed (Faulon, 2003) to solve problems containing a maximum dimensionality of two, was extended to include spatial information. The use of this fragment based descriptor facilitated reconstruction of potential solutions, being rooted in graph theoretical concepts, which allowed quick identification of all possible structural isomers relating to a group of molecular fragments. Further, conformational isomers could be estimated conveniently from fragment geometries since the signature descriptors overlap with each other by design, which eliminated the need for excessive conformational analysis of all potential solutions.

The initial methodology was developed to solve the problems presented in a deterministic manner, which considers the entire search space and guarantees identification of a global optimal solution. While thorough, this was seen as a limitation in solving larger molecular design problems with a bigger search space. As such, a stochastic evolutionary methodology was developed, along with the operators necessary to guide a population of solutions towards a set of desired property values. These operators included a single point crossover, deletion mutation, insertion mutation, and

fragment mutation. This technique was found to be effective at quickly identifying solutions meeting the desired criteria. In addition, it was able to identify several likely candidates instead of just searching for one global optimal solution, which is often the case in the solution of non-linear optimization problems such as these.

Within these two approaches at solving multi-dimensional molecular design problems it was necessary to develop many new methodologies for handling problems of this nature. The extension of signature descriptors to include spatial information necessitated molecular modelling techniques, from which the information had to be compressed for efficient usage within the algorithm. Additionally, both the stochastic and deterministic approaches described within this dissertation were coded and can be found within the attached appendix. Several different sources of software were utilized in a unique manner to generate the necessary information and new modules had to be developed to transform this information into a suitable format. The programming written to complete these tasks was written in Python, and occasionally shell scripts were written to transport this information into Python and automate molecular modelling tasks.

Several case studies have been presented to exemplify the applicability of this technique ranging from solvent design to the design of non-peptide mimetics. While a successful approach to molecular design, there are several limitations which need to be addressed. These limitations ultimately lead to the proposed future direction of the project, which can be found in the following sections.

5.1 Improved Simulation Techniques

The conformational analysis of molecular structures in all studies presented within this dissertation was done in a vacuum using varying levels of theory within the molecular and quantum mechanical approaches. In reality, this is not an optimal representation of molecular structures which are typically in solution or bound to form molecular complexes. Extension of this technique to generate molecular conformations which would more likely be found in these situations could be beneficial to the solution of more complex CAMD problems. This could perhaps be done by developing the initial spatial atomic signature data set from simulations of structure in solution or interacting with other molecules. One such approach could take advantage of the data generated through molecular dynamics simulations, to provide more realistic estimations of the geometry of potential solutions.

Molecular dynamics (MD) is a computer simulation of the physical movements of atoms and molecules within the simulation. The trajectories of atoms and molecules are determined by numerically solving Newton's equations of motion for a system of particles. The forces between atoms and molecules are determined by application of a molecular mechanics force field. Since it is currently impossible to solve for the properties of complex systems through analytical means, MD simulations afford the possibility of numerical analysis. These simulations have been applied successfully to the modelling of very large systems including: simulation of the complete satellite tobacco mosaic virus (Freddolino et al., 2006), which allowed researchers to probe the mechanisms of viral assembly; simulation of protein folding events such as that of the Villin Headpiece (Yong et al., 1998), which is an actin-binding protein; simulation of

nano-scale events such as the exfoliation of grapheme layers (Buddhika and Subbiah, 2011).

What these simulations provide with respect to the proposed methodology is a more accurate simulation/estimation of the spatial properties for a collection of molecules. This information is much closer to reality when compared to the conventional gas phase simulations utilized to gain insight into a molecule's preferred spatial conformations. This is because gas phase simulations often consider a single molecule isolated in vacuum space, whereas MD simulations consider the environmental interactions and how they affect preferred conformations.

There are several notable examples within the CAMD community where MD simulations have been successfully applied in the characterization and prediction of properties/attributes. One such example involves the prediction of ionic liquid properties including density, viscosity, diffusivity, melting point, enthalpy of vaporization and surface tension. Ionic liquids are salts which are liquid at room temperature and they possess at least one asymmetric unit comprised of a large organic cation and an organic or inorganic anion. The structural asymmetry makes their crystallization difficult and because of this they have an extremely low vapor pressure, high chemical stability, and good solvating capacity for organic and inorganic compounds and even biopolymers. Some examples of the most common cations and anions found in literature are shown in Figure 5.1.

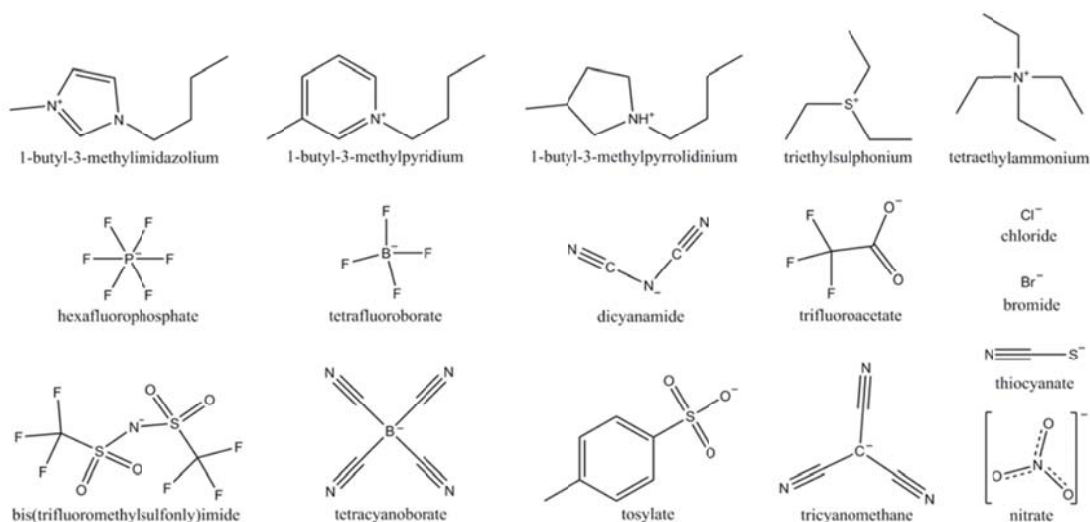


Figure 5.1 - Examples of Common Ionic Liquid Cations and Anions

While atomic scale descriptors, such as topological and topographic indices, have provided excellent correlations in the past, larger scale simulations, such as MD, provide a wealth of information which is not captured by studies at a smaller length scale. For example, the radial distribution function can be estimated through MD simulations and is quite useful for estimating the density of a given solvent, pure or even a mixture. Because of this, MD simulations are able to provide improved estimates for bulk properties when the force field utilized has been optimized for the given case. The interaction effects associated with molecules in a condensed phase are considered and this provides a more realistic estimation of the spatial properties seen for a given structure.

5.2 Consideration of Proteins

The signature descriptor has already been used to successfully explore the chemical space of proteins (Churchwell et al., 2004). The inverse quantitative structure-activity relationship (QSAR) approach was applied to a small set of inhibitory peptides directed

against leukocyte trafficking and localization. The forward approach of developing a QSAR was first conducted to develop a property model relating the occurrence of atomic signatures to the potency of a given peptide as measured by IC_{50} , which is the concentration that leads to half-maximal inhibition of receptor to ligand. A forward stepping algorithm was applied to select the most statistically significant signatures, which were then correlated to the logarithm of IC_{50} values. A Diophantine solver was applied to solve this equation in terms of the optimal set of atomic signatures. The next step was to generate all possible molecular structures containing these chosen signatures. The goal was to identify a novel inhibitory peptide possessing a lower IC_{50} value than any other structure within the training set. The group was able to identify and synthesize two peptides which were found to be the strongest inhibiting peptides to date, and they were measured as being very close to their estimated values. This study exemplifies the applicability of using atomic signatures towards exploring the chemical space of a set of proteins and identifying novel solutions with improved properties. While the descriptors utilized were the atomic signatures themselves, which simply contain two-dimensional information, a strong correlation was identified nonetheless.

It is well known that the association between ligands and receptors are strongly influenced by the spatial properties of both molecules involved. As such it would be useful to include such information in the CAMD efforts associated with identifying inhibitors with improved binding affinity. The problem with this approach is that, especially for protein structures, the conformational space associated with these potential inhibitors is very complex and requires much computational effort. In addition to having local geometry information such as bond lengths and angles, proteins also contain higher

order structural features known as secondary and tertiary structures. This information is difficult to capture at a local fragment-based level, however there are techniques available for estimating protein tertiary structures from fragment geometries. One such approach by Simons et al. (1997) develops this higher order structural information from fragments using simulated annealing and Bayesian scoring functions. In this technique, they are able to assemble native-like structures from fragments of unrelated protein structures with similar local sequences using these Bayesian scoring functions.

Previous applications, within this dissertation, have shown that the global geometry of a given molecules can be successfully estimated by the individual contributions from local fragment geometries. However, this was studied for only smaller organic structures. The secondary and tertiary structures associated with proteins are much more difficult to estimate and applying the methods utilized before would lead to protein structures which are considerably more expanded than native proteins. To account for this, one could initially assemble a set of protein conformers using fragments, and this would provide a reasonable starting point for further exploration of the protein's torsional space. Exploration of the torsional space could be guided by the process of conformational space annealing, such as that which was successfully applied in the Simons et al. (1997) study. This allows the algorithm to search a wide area within the conformational space of each protein, and ultimately/quickly converge on a set of likely local energy minima. Conformational space annealing has been applied in other optimization studies (Lee et al., 1997) and was successful in exploring the conformational space of reasonably complex protein structures.

5.3 Simulated Annealing

Simulated annealing (Van Laarhoven and Aarts, 1987) represents a class of solution methods applied for combinatorial optimization problems with analogies applied from the physical process of annealing. Annealing is the physical thermal process of melting a solid by heating it, followed by slow cooling and crystallization into a stable state. This approach has been applied to CAMD studies (Ourique and Telles, 1998) and conformational analysis studies (Kleber and Tsallis, 1996) with much success. Starting at a higher “temperature” the search algorithm is allowed to explore less favorable solution spaces, denoted by a higher conformational energy or being further away from the desired properties with respect to a CAMD problem, such that it does not get stuck in a local metastable state. The temperature is slowly dropped, or cooled, so that the structures generated have an increased likelihood of exhibiting the desired properties.

It has been established within this dissertation that the nature of many CAMD problems is highly nonlinear such that linear programming techniques cannot be applied. In addition, the complexity of many descriptors and property models utilized make application of an MINLP technique very difficult. As such, the application of stochastic optimization techniques was necessitated because of these reasons, in addition to the large search space considered by most ambitious CAMD problems. One technique presented within this dissertation is known as genetic algorithm, and it was shown to be successful towards solving problems of this nature. However, it would be beneficial to explore additional stochastic optimization approaches, such as simulated annealing, because of their success in similar applications.

One application of simulated annealing (SA) within the CAMD community can be found in study by Ourique and Telles (1998). This work solved problems identifying pure substances or mixtures that satisfied a set of chemical or physical properties by representing molecules as chemical graphs and applying a simulated annealing algorithm. Molecules were represented as hydrogen-suppressed graphs with bonding and atom type information stored within a structure-composition matrix. Within this square matrix, group identities were stored in the main diagonal with zeroes and ones in the remaining entries corresponding to bonds between these groups. In this case, the representation of molecules was provided in the matrix; however, the defined search space would be vast utilizing an atom based description of molecules. The utilization of signature fragments as molecular building blocks would further contain the search space such that structures would be more likely to fall within the applicability domain of the chosen property models. In addition, extension to consider conformational information would fall in line with the simulated annealing approach. While the SA algorithm could act on the initial graph itself, estimation of likely conformational minima could be processed in the same manner. This would provide a common algorithm with which to analyze the feasibility of various molecules for a given CAMD problem, while also exploring the structural and conformational capabilities of a chosen search space. Such an approach could prove to be beneficial in identified novel chemical solutions unattainable within the GA approach because of the initial exploration of unfavorable regions in chemical space.

References

- Achenie, L., Gani, R., & Venkatasubramanian, V. (2003). *Computer Aided Molecular Design: Theory and Practice*. Amsterdam, The Netherlands: Elsevier Science.
- Allen, L., & Karo, A. (1960). Basis Functions for Ab Initio Calculations. *Revs. Mod. Phys.*, 275-285.
- Balaban, A. (1976). *Chemical application of graph theory*. Academic Press.
- Basak, S., Gute, B., & Grunwald, G. (1996). A comparative study of topological and geometrical parameters in estimating normal boiling point and octanol/water partition coefficient. *Journal of Chemical Information and Computer Sciences*, 1054-1060.
- Basak, S., Gute, B., & Grunwald, G. (1996). A Comparative Study of Topological and Geometrical Parameters in Estimating Normal Boiling Point and Octanol/Water Partition Coefficient. *J. Chem. Inf. Comput. Sci.*, 1054-1060.
- Basak, S., Roy, A., & Ghosh, J. (1988). Proceedings of the IInd International Conference on Mathematical Modeling. Minnesota: University of Minnesota.
- Becke, A. (1988). Density-functional exchange-energy approximation with correct asymptotic behavior. *Physical Reviews A*, 3098-3100.
- Beusen, D., Shands, E., & Karasek, S. (1996). Systematic search in conformational analysis. *J. Mol. Struc.*, 157-171.
- Bogdanov, B., Nikolic, S., & Trinajstic, N. (1989). On the Three-Dimensional Wiener Number. *J. Math. Chem.*, 299-309.
- Born, M., & Huang, K. (1988). *Dynamical Theory of Crystal Lattices*. Oxford: Clarendon Press.
- Breiman, L., Friedman, J., Stone, C., & Olshen, R. (1984). *Classification and regression trees*. CRC press.
- Brown, N., McKat, B., & Gasteiger, J. (2006). A novel workflow for the inverse QSPR problem using multiobjective optimization. *J. Comput Aided Mol Des*, 333-341.
- Brown, N., McKay, B., & Gasteiger, J. (2005). Fingal: A Novel Approach to Geometric Fingerprinting and a Comparative Study of Its Application to 3D-QSAR Modelling. *QSAR & Combinatorial Science*, 480-484.

- Buddhika, J., & Subbiah, S. (2011). A novel mechanical cleavage method for synthesizing few-layer graphenes. *Nanoscale Res. Lett.*, 95-101.
- Carhard, R., Smith, D., & Venkataraghavan, R. (1985). Atom Pairs as Molecular Features in Structure-Activity Studies: Definition and Applications. *J. Chem. Inf. Comp. Sci.*, 64-73.
- Castro, E., Gutman, I., Marino, D., & Peruzzo, P. (2002). Upgrading the Wiener index. *J. Serb. Chem. Soc.*, 647-651.
- Chang, G., Guida, W., & Still, W. (1989). An internal-coordinate Monte Carlo method for searching conformational space. *Journal of the American Chemical Society*, 4379-4386.
- Chemangattuvalippil, N. (2008). *A Systematic Property Based Approach for Molecular Synthesis using Higher Order Molecular Groups and Molecular Descriptors*. Auburn: Auburn University.
- Chen, X., & Reynolds, C. (2002). Performance of Similarity Measures in 2D Fragment-Based Similarity Searching: Comparison of Structural Descriptors and Similarity Coefficients. *J. Chem. Inf. Comput. Sci.*, 1407-1414.
- Churchwell, C., Rintoul, M., Martin, S., Visco Jr., D., Kotu, A., Larson, R., . . . Faulon, J.-L. (2004). The signature molecular descriptor 3. Inverse-quantitative structure-activity relationship of ICAM-1 inhibitory peptides. *Journal of Molecular Graphics and Modelling*, 263-273.
- Clark, T., & Koch, R. (1999). *Linear Combination of Atomic Orbitals*. Berlin: Springer.
- Constantinou, L., & Gani, R. (1994). New group contribution method for estimating properties of pure compounds. *AIChE Journal*, 1697-1710.
- Constantinou, L., Bagherpour, K., Gani, R., Klein, J., & Wu, D. (1996). Computer aided product design: problem formulations, methodology and applications. *Computers Chem. Eng.*, 685-702.
- Cortes, C., & Vladimir, V. (1995). Support-vector networks. *Machine learning*, 273-297.
- Cramer, R., Patterson, D., & Bunce, J. (1988). Comparative Molecular Field Analysis (CoMFA) 1. Effect of Shape on Binding of Steroids to Carrier Proteins. *Journal of the American Chemical Society*, 5959-5967.
- Cramer, R., Patterson, D., & Bunce, J. (1988). Comparative Molecular Field Analysis (CoMFA). 1. Effect of Shape on Binding of Steroids to Carrier Proteins. *J. AM. Chem. Soc.*, 5959-5967.

- Cruciani, G., Vrivori, P., Carrupt, P.-A., & Testa, B. (2000). Molecular fields in quantitative structure-permeation relationships: the VolSurf approach. *Journal of Molecular Structure: THEOCHEM*, 17-30.
- Curry, H. (1944). The method of steepest descent for nonlinear minimization problems. *Quarterly of Applied Mathematics*, 250-261.
- Cussler, E., & Moggridge, G. (2001). *Chemical Product Design*. Cambridge, UK: The Press Syndicate of the University of Cambridge.
- Davis, S., Hada, S., Herring, R., & Eden, M. (2014). Characterization Based Reverse Design of Ionic Liquids. *Computer Aided Chemical Engineering*, 285-290.
- Daylight Chemical Information, I. (n.d.). MedChem ver 3.54. Mission Viejo, CA, USA: Daylight Chemical Information, Inc.
- Dewar, M., Zoebisch, E., Healy, E., & Stewart, J. (1985). Development and use of quantum mechanical molecular models. 76. AM1: a new general purpose quantum mechanics molecular model. *Journal of the American Chemical Society*, 3902-3909.
- Dewar, M., Zoebisch, E., Healy, E., & Stewart, J. (1985). Development and use of quantum molecular models. 75. Comparative tests of theoretical procedures for studying chemical reactions. *The Journal of the American Chemical Society*, 3902-3909.
- Diudea, M., Horvath, D., & Graovac, A. (1995). Molecular Topology. 15. 3D Distance Matrices and Related Topological Indices. *J. Chem. Inf. Comput. Sci.*, 129-135.
- Douguet, D., Throeu, E., & Grassy, G. (2000). A genetic algorithm for the automated generation of small organic molecules: Drug design using an evolutionary algorithm. *Journal of computer-aided molecular design*, 449-466.
- Dudek, A., Arodz, T., & Galvez, J. (2006). Computational Methods in Developing Quantitative Structure-Activity Relationships (QSAR): A Review. *Combinatorial Chemistry & High Throughput Screening*, 213-228.
- Eriksson, L., Johansson, E., Kettaneh-Wold, N., Trygg, J., Wikstron, C., & Wold, S. (2006). *Multi- and Megavariate Data Analysis: Basic Principles and Applications (Part I)*. Umea: Umetrics Academy.
- Faulon, J.-L. (2003). The Signature Molecular Descriptor. 1. Using Extended Valence Sequences in QSAR and QSPR studies. *J. Chem. Inf. Comput. Sci.*, 707-720.

- Faulon, J.-L. (2014, 8 26). *Molecular Signature Software*. Retrieved from Jean-Loup Faulon WikiSpace:
<http://jfaulon.wikispaces.com/Signature+publications+%26+downloads>
- Faulon, J.-L., & Churchwell, C. (2003). The Signature Molecular Descriptor 2. Enumerating Molecules from Their Extended Valence Sequences. *J. Chem. Inf. Comput. Sci.*, 721-734.
- Fletcher, R. (1987). *Practical methods of optimization*. New York: John Wiley & Sons.
- Freddolino, P., Arkhipov, S., Larson, S., McPherson, A., & Schulten, K. (2006). Molecular dynamics simulations of the complete satellite tobacco mosaic virus. *Structure*, 437-449.
- Fredenslund, A., Jones, R., & Prausnitz, J. (1975). Group-Contribution Estimation of Activity Coefficients in Nonideal Liquid Mixtures. *AIChE Journal*, 1086.
- Geladi, P., & Kowalski, B. (1986). Partial least-squares regression: a tutorial. *Analytica chimica acta*, 1-17.
- Gibbs, J. (1902). *Elementary Principles in Statistical Mechanics*. New York: Charles Scribner's Sons.
- Guyon, I., & Elisseeff, A. (2003). An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 1157-1182.
- Hada, S., Herring, R., & Eden, M. (2013). Design of Ionic Liquids Using Property Clustering and Decomposition Techniques. *Computer Aided Chemical Engineering*, 955-960.
- Halgren, T. (1996). Merck molecular force field. I. Basis, form, scope, parameterization, and performance of MMFF94. *Journal of Computational Chemistry*, 490-519.
- Hall, L., Kellogg, G., & Haney, D. (1991). *MOLCONN-Z*. Quincy, MA: Hall Associates Consulting.
- Hall, L., Kier, L., & Murray, W. (1975). Molecular connectivity II: Relationship to water solubility and boiling point. *Journal of pharmaceutical sciences*, 1974-1977.
- Hansch, C. (1969). A Quantitative Approach to Biochemical Structure-Activity Relationships. *Acc. Chem. Res.*, 232-239.
- Hansch, C., Maloney, P., Fujita, T., & Muir, R. (1962). Correlation of biological activity of phenoxyacetic acids with Hammett substituent constants and partition coefficients. *Nature*, 178-180.

- Harper, P. M., Gani, R., Kolar, P., & Ishikawa, T. (1999). Computer-aided molecular design with combined molecular modeling and group contribution. *Fluid Phase Equilibria*, 337-347.
- Haser, J., Herring, R., Datta, S., & Eden, M. (2014). Development of QSPR Model Relating Solvent Structure to Crystal Morphology. *Computer Aided Chemical Engineering*, 321-326.
- Herring, R., & Eden, M. (2014b). Graph-Based Genetic Algorithm for De Novo Molecular Design. *Computer Aided Chemical Engineering*, 327-332.
- Herring, R., & Eden, R. (2014). De Novo Molecular Design using a Graph-Based Genetic Algorithm Approach. *Computer Aided Chemical Engineering*, 7-12.
- Herring, R., Haser, J., Hada, S., & Eden, M. (2013). Structure Based Design of Non-Peptide Mimetics. *Computer Aided Chemical Engineering*, 175-180.
- Herring, R., Namikis, R., Chemmangattuvalappil, N., Roberts, C., & Eden, M. (2012). Incorporating Topographical Characteristics in Molecular Signature Descriptors. *Computer Aided Chemical Engineering*, 497-501.
- Herring, R., Namikis, R., Chemmangattuvalappil, N., Roberts, C., & Eden, M. (2012). Molecular Design using Three-Dimensional Descriptors. *Computer Aided Chemical Engineering*, 225-229.
- Holland, J. (1975). Adaptation in Natural and Artificial Systems. *The University of Michigan Press*.
- Insight, I. (2001). Accelrys Software. San Diego, CA, USA.
- Itskowitz, P., & Tropsha, A. (2005). k Nearest Neighbors QSAR Modeling as a Variational Problem: Theory and Applications. *J. Chem. Inf. Model.*, 777-785.
- Jaworska, J., Nikolova-Jeliazkova, N., & Aldenberg, T. (2005). QSAR Applicability Domain Estimation by Projection of the Training Set in Descriptor Space: A review. *ATLA*, 445-459.
- Joback, K., & Reid, R. (1983). Estimation of Pure-Component Properties from Group Contributions. *Chemical Engineering Communication*, 233.
- Jorgensen, W., & Tirado-Rives, J. (2005). Molecular modeling of organic and biomolecular systems using BOSS and MCPRO. *Journal of Computational Chemistry*, 1689-1700.

- Kar, S., & Roy, K. (2010). QSAR modeling of toxicity of diverse organic chemicals to *Daphnia magna* using 2D and 3D descriptors. *Journal of Hazardous Materials*, 344-351.
- Karplus, M., & McCammon, A. (2002). Molecular dynamics simulations of biomolecules. *Nature*, 646-652.
- Katritzky, A., & Gordeeva, E. (1993). Traditional Topological Indices vs. Electronic, Geometrical, and Combined Molecular Descriptors in QSAR/QSPR Research. *J. Chem. Inf. Comput. Sci.*, 835-857.
- Kier, L., & Hall, L. (1999). *Molecular Structure Description: The Electrotopological State*. New York: Academic Press.
- Kier, L., & Murray, W. (1975). Molecular Connectivity. 4. Relationships to Biological Activity. *Journal of Medicinal Chemistry*, 1272-1274.
- Kier, L., Murray, W., Randic, M., & Hall, L. (1975). Molecular connectivity. I. Relationship to nonspecific local anesthesia. *Journal of Pharmaceutical Sciences*, 1971-1974.
- Kirkpatrick, P., & Ellis, C. (2004). Chemical Space. *nature insight*, 823-865.
- Kittler, J. (1978). Feature set search algorithms. *Pattern recognition and signal processing*, 41-60.
- Klebe, G., Abraham, U., & Mietzner, T. (1994). Molecular Similarity Indices in a Comparative Analysis (CoMSIA) of Drug Molecules To Correlate and Predict Their Biological Activity. *J. Med. Chem.*, 4130-4146.
- Kleber, M., & Tsallis, C. (1996). Geometry optimization and conformational analysis through generalised simulated annealing. *International Journal of Quantum Chemistry*, 373-381.
- Koch, R. (1982). Molecular Connectivity and Acute Toxicity of Environmental Pollutants. *Chemosphere*, 925-931.
- Kurogi, Y., & Guner, O. (2001). Pharmacophore modeling and three-dimensional database searching for drug design using catalyst. *Current medicinal chemistry*, 1035-1055.
- Leach, A. (2001). Empirical Force Field Models: Molecular Mechanics. In A. Leach, *Molecular Modelling: principles and applications* (pp. 165-252). Harlow: Pearson Education Limited.

- Lee, J., Scheraga, H., & Rackovsky, S. (1997). New Optimization Method for Conformational Energy Calculations on Polypeptides: Conformational Space Annealing. *Journal of Computational Chemistry*, 1222-1232.
- Lipinski, C., Lombardo, F., Dominy, B., & Feeney, P. (2001). Experimental and Computational Approaches to Estimate Solubility and Permeability in Drug Discovery and Development Settings. *Advanced Drug Delivery Reviews*, 3-26.
- Lopez-Rodriguez, M., Benhamu, B., de la Fuente, T., Sanz, A., Pardo, L., & Campillo, M. (2005). A Three-Dimensional Pharmacophore Model for 5-Hydroxytryptamine (5-HT₆) Receptor Antagonists. *Journal of Medicinal Chemistry*, 4216-4219.
- Ltd., M. D. (n.d.). Maccs II. San Leandro, CA, USA.
- MacGregor, J., & Kourti, T. (1995). Statistical process control of multivariate process. *Control Engineering Practice*, 403-414.
- Magnuson, V., Harriss, D., & Basak, S. (1983). Information indices. *Studies in Physical and Theoretical Chemistry*, 178-191.
- Martin, Y. (1992). 3D Database Searching in Drug Design. *Journal of Medicinal Chemistry*, 2145-2154.
- Mayo, S., Olafson, B., & Goddard, W. (1990). DREIDING: a generic force field for molecular simulations. *Journal of Physical Chemistry*, 8897-8909.
- Merkwirth, C., Mauser, H., Schulz-Gasch, T., Roche, O., Stahl, M., & Lengauer, T. (2004). Ensemble Methods for Classification in Cheminformatics. *J. Chem. Inf. Comput. Sci.*, 1971-1978.
- Metropolis, N., & Ulam, S. (1949). The Monte Carlo Method. *J. Am. Stat. Assoc.*, 335-341.
- Mihalic, Z., & Trinajstic, N. (1991). The Algebraic Modelling of Chemical Structures: On the Development of Three-Dimensional Molecular Descriptors. *Journal of Molecular Structure*, 65-78.
- Moller, C., & Plesset, M. (1934). Note on an approximation treatment for many-electron systems. *Physical Reviews*, 618-622.
- NCI AIDS Data Set*. (2002). Retrieved from NCI Developmental: (<http://dtp.nci.nih.gov/docs/aids/aids->
- Needham, D., Wei, I., & Seybold, P. (1988). Molecular Modeling of the Physical Properties of the Alkanes. *Journal of the American Chemical Society*, 4186-4194.

- Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, 308-313.
- Nelder, J., & Mead, R. (1965). A simplex method for function minimization. *The computer journal*, 308-313.
- Nettles, J., Jenkins, J., Bender, A., Deng, Z., Davies, J., & Glick, M. (2006). Bridging Chemical and Biological Space: "Target Fishing" Using 2D and 3D Molecular Descriptors. *J. Med. Chem.*, 6802-6810.
- Oliphant, T. (2007). Python for Scientific Computing. *Comp. in Sci. and Eng.*, 10-20.
- Olson, G., Bolin, D., Bonner, M., Bos, M., Cook, C., Fry, D., . . . Hill, D. (1993). Concepts and Progress in the Developmetn of Peptide Mimetics. *Journal of Medicinal Chemistry*, 3039-3049.
- Ourique, J., & Telles, A. (1998). Computer-aided molecular design with simulated annealing and molecular graphs. *Computers and Chemical Engineering*, 615-618.
- Pastor, M., Cruciani, G., McLay, I., Pickett, S., & Clementi, S. (2000). GRid-INdependent Descriptors (GRIND): A Novel Class of Alighment-Independent Three-Dimensional Molecular Descriptors. *J. Med. Chem.*, 3233-3243.
- Pegg, S., Haresco, J., & Kuntz, I. (2001). A genetic algorithm for structure-based de novo design. *Journal of computer-aided molecular design*, 911-933.
- Pepperrell, C., Taylor, R., & Willett, P. (1990). Implementation and use of an atom-mapping procedure for similarity searching in databases of 3-D chemical structures. *Tetrahedron Computer Methodology*, 575-593.
- Perola, E., & Charifson, P. (2004). Conformational Analysis of Drug-Like Molecules Bound to Proteins: An Extensive Study of Ligand Reorganization upon Binding. *J. Med. Chem.*, 2499-2510.
- Pople, J., Beveridge, D., & Dobosh, P. (1967). Approximate Self-Consistent Molecular-Orbital Theory. V. Intermediate Neglect of Differential Ocerlap. *The Journal of Chemical Physics*, 2026-2033.
- Quinlan, J. (1986). Induction of decision trees. *Machine learning*, 81-106.
- Raman, V., & Maranas, C. (1998). Optimization in product design with properties correlated with topological indices. *Computers and Chemical Engineering*, 747-763.
- Randic, M. (1987). In R. Lacher. Amsterdam: Elsevier.

- Reeves, F., & Reeves, C. (1964). Function minimization by conjugate gradients. *The computer journal*, 149-154.
- Rish, I. (2001). An empirical study of the naive Bayes classifier. *IJCAI 2001 workshop of empirical methods in artificial intelligence*, 41-46.
- Roothaan, C. (1951). New Developments in Molecular Orbital Theory. *Reviews of Modern Physics*, 69-89.
- Rouvray, D. (1971). Graph theory in chemistry. *Royal Inst. Chem. Revs.*, 173-195.
- Shannon, C. (1948). A note on the concept of entropy. *Bell System Tech*, 379-423.
- Sheridan, R., Feuston, R., Maiorov, V., & Kearsley, S. (2004). Similarity to molecule sin the training set is a good discriminator for prediction accuracy in QSAR. *J. Chem. Inf. Comput. Sci.*, 1912-1928.
- Sherrill, D., & Schaefer III, H. (1999). The Configuration Interaction Method: Advances in Highly Correlated Approaches. *Advances in Quantum Chemistry*, 143-269.
- Siedlecki, W., & Sklansky, J. (1988). On Automatic Feature Selection. *Internatinoal Journal of Pattern Recognition and Artificial Intelligence*, 197-220.
- Silverman, B., & Platt, D. (1996). Comparative Molecular Moment Analysis (CoMMA): 3D-QSAR without Molecular Superposition. *J. Med. Chem.*, 2129-2140.
- Simons, K., Kooperberg, C., Huang, E., & Baker, D. (1997). Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and Bayesian scoring functions. *Journal of Molecular Biology*, 209-225.
- Slater, J. (1930). Atomic Shielding constants. *Physical reviews*, 57.
- Smola, A., & Scholkopf, B. (2004). A tutorial on support vector regression. *Statistics and computing*, 199-222.
- Stewart, J. (1989). Optimization of parameters for semiempirical methods I. Method. *The Journal of Computational Chemistry*, 209-220.
- Stewart, J. (1991). Optimization and Application of Magnesium Parameters for PM3. *Journal of Computational Chemistry*, 320-328.
- Sutter, M., Dixon, S., & Jurs, P. (1995). Automated Descriptor Selection for Quantitative Structure-Activity Relationships Using Generalized Simulated Annealing. *J. Chem. Inf. Comput. Sci.*, 77-84.

- Svetnik, V., Wang, T., Tong, C., Liaw, A., Sheridan, R., & Song, Q. (2005). Boosting: An ensemble learning tool for compound classification and QSAR modeling. *Journal of Chemical Information and Modeling*, 786-799.
- Systems, M. I. (2002). MDDR Data Set. San Leandro, CA, USA.
- Testa, B., & Salvesen, B. (1980). Quantitative Structure-Activity Relationships in Drug Metabolism and Disposition: Pharmacokinetics of N-substituted Amphetamines in Humans. *J. Pharm. Sci.*, 497-501.
- Todeschini, R., & Consonni, V. (2009). *Molecular Descriptors for Chemoinformatics*. Weinheim: Wiley-VCH.
- Toropov, A., Toropova, A., Ismailov, T., & Bonchev, D. (1998). 3D weighting of molecular descriptors for QSPR/QSAR by the method of ideal symmetry (MIS). 1. Application to boiling points of alkanes. *Journal of Molecular Structure*, 237-247.
- Trinajstić, N. (1983). *Chemical graph theory*. Boca Raton, FL: CRC Press.
- Uchida, M. (1980). Affinity and Mobility of Fungicidal Dialkyldithiolanylidene malonates in rice plants. *Pesticide Biochemistry and Physiology*, 249-255.
- Van Laarhoven, P., & Aarts, E. (1987). *Simulated annealing*. Netherlands: Springer.
- Venkatasubramanian, V., Chan, K., & Caruthers, J. (1995). Evolutionary Design of Molecules with Desired Properties Using the Genetic Algorithm. *J. Chem. Inf. Comp. Sci.*, 188-195.
- Verma, J., & Khedkar, V. (2010). 3D-QSAR in Drug Design- A Review. *Curr. Top. in Med. Chem.*, 95-115.
- Visco Jr., D., Pophale, R., & Faulon, J.-L. (2003). The signature molecular descriptor. 1. Using extended valence sequences in QSAR and QSPR studies. *J. Chem. Inf. Comp. Sci.*, 707-720.
- Visco Jr., D., Pophale, R., Rintoul, M., & Faulon, J.-L. (2002). Developing a methodology for an inverse quantitative structure-activity relationship using the signature molecular descriptor. *Journal of Computer Graphics and Modelling*, 429-438.
- Weiner, H. (1947). Structural Determination of paraffin boiling points. *Journal of American Chemical Society*, 17-20.

- Weiner, H. (1947a). Structural determination of paraffin boiling points. *Journal of American Chemical Society*, 17-20.
- Weiner, H. (1947b). Correlation of heats of isomerization and differences in heats of vaporization of isomers among the paraffin hydrocarbons. *Journal of American Chemical Society*, 2636-2638.
- Wentzell, P., & Vega Montoto, L. (2003). Comparison of principal components regression and partial least squares regression through generic simulations of complex mixtures. *Chemometrics and Intelligent Laboratory Systems*, 257-279.
- Wermuth. (1998). Glossary of terms used in medicinal chemistry (IUPAC Recommendations 1997). *Annual Reports in Medicinal Chemistry*, 385-395.
- Wermuth, C., Ganellin, C., Lindberg, P., & Mitscher, L. (1998). Glossary of terms used in medicinal chemistry (IUPAC Recommendations). *Pure and Applied Chemistry*, 1129-1143.
- Wolber, G., Seidel, T., Bendix, F., & Langer, T. (2008). Molecule-pharmacophore superpositioning and pattern matching in computational drug design. *Drug discovery today*, 23-29.
- Wold, S., Kettaneh, N., & Tjessem, K. (1996). Hierarchical multiblock PLS and PC models for Easier Model interpretation and as an alternative to variable selection. *Journal of Chemometrics*, 463-482.
- Yong, D., Wang, L., & Killman, P. (1998). The early stage of folding of villin headpiece subdomain observed in a 200-nanosecond fully solvated molecular dynamics simulation. *Proceedings of the National Academy of Sciences*, 9897-9902.

Appendix A – Python Code for Proposed Methodology

A.1 – Creation of Spatial Atomic Signatures from Directory

The format chosen for developing atomic signatures from a given data set is to specify a directory containing the mol files of the data set and turn each of these molecules into a graph and subsequently their respective atomic signature graphs. A flowchart explaining the flow of information can be seen in Figure A.1.1.

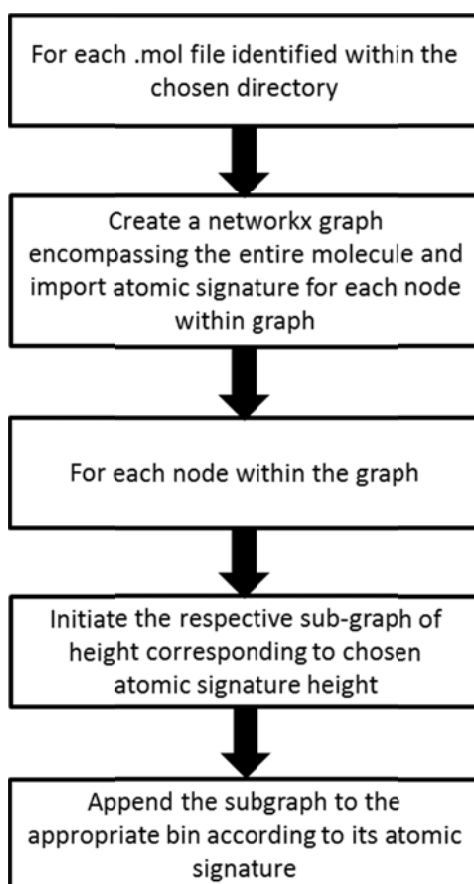


Figure A.1.1 – Flowchart for the generation of atomic signatures from mol files.

The following code exemplifies how this is achieved within a python framework. The technique for generating hydrogen suppressed molecular graphs is also shown.


```

import networkx as nx
import networkx.algorithms.isomorphism as iso
import math
from math import fabs
import itertools
import re
import glob
import subprocess
import linecache
from collections import defaultdict
from operator import eq
from pprint import pprint
import numpy
from operator import itemgetter
import random
valence_dictionary = {'C':4,'O':2,'H':1,'N':3,'S':2,'Cl':1,'F':1,'Br':1}
bond_dictionary = {1:1,2:2,4:1.5}
class Directory:
    def __init__(self,full_parent_directory):
        self.full_parent_directory = full_parent_directory
    def create_signatures(self,height):
        """This function canonizes the mol files in directory and stores signatures in self.signatures dictionary"""
        self.height = height
        self.signatures = defaultdict(lambda:defaultdict(list))
        self.subgraphs = []
        self.compressed_subgraphs = []
        self.working_signature = ""
        if self.full_parent_directory[-1] == '/':
            self.full_parent_directory = self.full_parent_directory[:len(self.full_parent_directory)-1]
        mol_files = self.full_parent_directory + '/*.mol'
        file_iter = glob.iglob(mol_files)
        total_subgraphs = 0
        signatures_stored = 0
        graph_count = 0
        for fn in file_iter:

```

```

# Creating a nx.Graph from the mol file
G = nx.Graph()
if 'OpenBabel' in str(linecache.getline(fn,2)):
    atom_bond_number_line = 4
else:
    atom_bond_number_line = 5
atomn = int((linecache.getline(fn,atom_bond_number_line)[1:4]).strip())
bondn = int((linecache.getline(fn,atom_bond_number_line)[4:8]).strip())
coordinate_begin_line = atom_bond_number_line + 1
coordinate_end_line = atom_bond_number_line + atomn
bond_begin_line = coordinate_end_line + 1
bond_end_line = coordinate_end_line + bondn
for i in range(bond_begin_line, bond_end_line + 1):
    atom1 = int((linecache.getline(fn,i)[0:4]).strip())
    atom2 = int((linecache.getline(fn,i)[4:8]).strip())
    bondtype = int((linecache.getline(fn,i)[8:10]).strip())
    G.add_edge(atom1, atom2, type=bondtype)
for i in range(coordinate_begin_line,coordinate_end_line + 1):
    j = i - (atom_bond_number_line)
    G.node[j]['xyz'] = numpy.array([float((linecache.getline(fn,i)[4:11]).strip()),
                                   float((linecache.getline(fn,i)[14:21]).strip()),float((linecache.getline(fn,i)[24:31]).strip())])
    G.node[j]['atom'] = ((linecache.getline(fn,i)[31:33]).strip())
# Now breaking this graph into subgraphs
nodes = G.nodes_iter()
for node in nodes:
    subgraph=nx.ego_graph(G, node, radius=self.height)
    subgraph.graph['degree'] = subgraph.degree(node)
    subgraph.graph['center'] = node
    subgraph.graph['atom'] = G.node[node]['atom']
    subgraph.graph['rebuild'] = graph_count
    self.subgraphs.append(subgraph)
graph_count += 1
print 'Subgraphs created:',len(self.subgraphs)
for subgraph in self.subgraphs:
    match_status = 0

```

```

if len(self.compressed_subgraphs) > 0:
    for subgraph_list in self.compressed_subgraphs:
        nm = iso.categorical_node_match('atom', 'C')
        em = iso.numerical_edge_match('type', 4)
        GM = iso.GraphMatcher(subgraph, subgraph_list[0], node_match = nm, edge_match = em)
        if GM.is_isomorphic():
            match_status = 1
            new_graph = nx.relabel_nodes(subgraph, GM.mapping, copy=True)
            subgraph_list.append(new_graph)
            break
    if match_status == 0:
        new_list = []
        new_list.append(subgraph)
        self.compressed_subgraphs.append(new_list)
    else:
        new_list = []
        new_list.append(subgraph)
        self.compressed_subgraphs.append(new_list)

conformers = 0
signatures = 0
for each_list in self.compressed_subgraphs:
    signatures += 1
    conformers += len(each_list)
print 'Subgraphs stored:', conformers
print 'Overall there were:', signatures, 'unique signatures identified.'

def create_suppressed_signatures(self, height):
    """This function canonizes the mol files in directory and stores hydrogen suppressed signatures in self.signatures
    dictionary"""
    self.height = height
    self.signatures = defaultdict(lambda: defaultdict(list))
    self.subgraphs = []
    self.compressed_subgraphs = []
    self.working_signature = ""
    if self.full_parent_directory[-1] == '/':
        self.full_parent_directory = self.full_parent_directory[:len(self.full_parent_directory)-1]
    mol_files = self.full_parent_directory + '/*.mol'

```

```

file_iter = glob.iglob(mol_files)

total_subgraphs = 0

signatures_stored = 0

graph_count = 0

for fn in file_iter:

    # Creating a nx.Graph from the mol file

    G = nx.Graph()

    if 'OpenBabel' in str(linecache.getline(fn,2)):

        atom_bond_number_line = 4

    else:

        atom_bond_number_line = 5

    atomn = int((linecache.getline(fn,atom_bond_number_line)[1:4]).strip())

    bondn = int((linecache.getline(fn,atom_bond_number_line)[4:8]).strip())

    coordinate_begin_line = atom_bond_number_line + 1

    coordinate_end_line = atom_bond_number_line + atomn

    bond_begin_line = coordinate_end_line + 1

    bond_end_line = coordinate_end_line + bondn

    for i in range(bond_begin_line, bond_end_line + 1):

        atom1 = int((linecache.getline(fn,i)[0:4]).strip())

        atom2 = int((linecache.getline(fn,i)[4:8]).strip())

        bondtype = int((linecache.getline(fn,i)[8:10]).strip())

        G.add_edge(atom1, atom2, type=bondtype)

    for i in range(coordinate_begin_line,coordinate_end_line + 1):

        j = i - (atom_bond_number_line)

        G.node[j]['xyz'] = numpy.array([float((linecache.getline(fn,i)[4:11]).strip()),
                                       float((linecache.getline(fn,i)[14:21]).strip()),float((linecache.getline(fn,i)[24:31]).strip()), 1])

        G.node[j]['atom'] = ((linecache.getline(fn,i)[31:33]).strip())

    # Now breaking this graph into subgraphs

    edges_to_remove = []

    for edge in G.edges_iter():

        if G.node[edge[0]]['atom']=='H' or G.node[edge[1]]['atom']=='H':

            edges_to_remove.append(edge)

    for edge in edges_to_remove:

        G.remove_edge(edge[0],edge[1])

    nodes_to_remove = []

```

```

for node in G.nodes_iter():
    if G.node[node]['atom'] == 'H':
        nodes_to_remove.append(node)
for node in nodes_to_remove:
    G.remove_node(node)
for node in G.nodes_iter():
    subgraph=nx.ego_graph(G, node, radius=self.height)
    subgraph_graph['degree'] = subgraph.degree(node)
    subgraph_graph['center'] = node
    subgraph_graph['atom'] = G.node[node]['atom']
    subgraph_graph['rebuild'] = graph_count
    self.subgraphs.append(subgraph)

graph_count += 1
print 'Subgraphs created:',len(self.subgraphs)
for subgraph in self.subgraphs:
    match_status = 0
    if len(self.compressed_subgraphs) > 0:
        for subgraph_list in self.compressed_subgraphs:
            nm = iso.categorical_node_match('atom', 'C')
            em = iso.numerical_edge_match('type', 4)
            GM = iso.GraphMatcher(subgraph,subgraph_list[0],node_match = nm, edge_match = em)
            if GM.is_isomorphic():
                match_status = 1
                new_graph=nx.relabel_nodes(subgraph, GM.mapping, copy=True)
                subgraph_list.append(new_graph)
                break
    if match_status == 0:
        new_list = []
        new_list.append(subgraph)
        self.compressed_subgraphs.append(new_list)
    else:
        new_list = []
        new_list.append(subgraph)
        self.compressed_subgraphs.append(new_list)
conformers = 0

```

```

signatures = 0
for each_list in self.compressed_subgraphs:
    signatures += 1
    conformers += len(each_list)
print 'Subgraphs stored:', conformers
print 'Overall there were:', signatures, 'unique signatures identified.'

```

A.2 – Network Generation

The bonding network is discussed extensively throughout the dissertation, and the generation of this network is established here. In addition, the algorithm responsible for network compression is also shown. The steps taken to first create the network can be visualized as shown in Figure A.2.1. Additionally, the steps necessary to compress the bonding network are shown in Figure A.2.2.

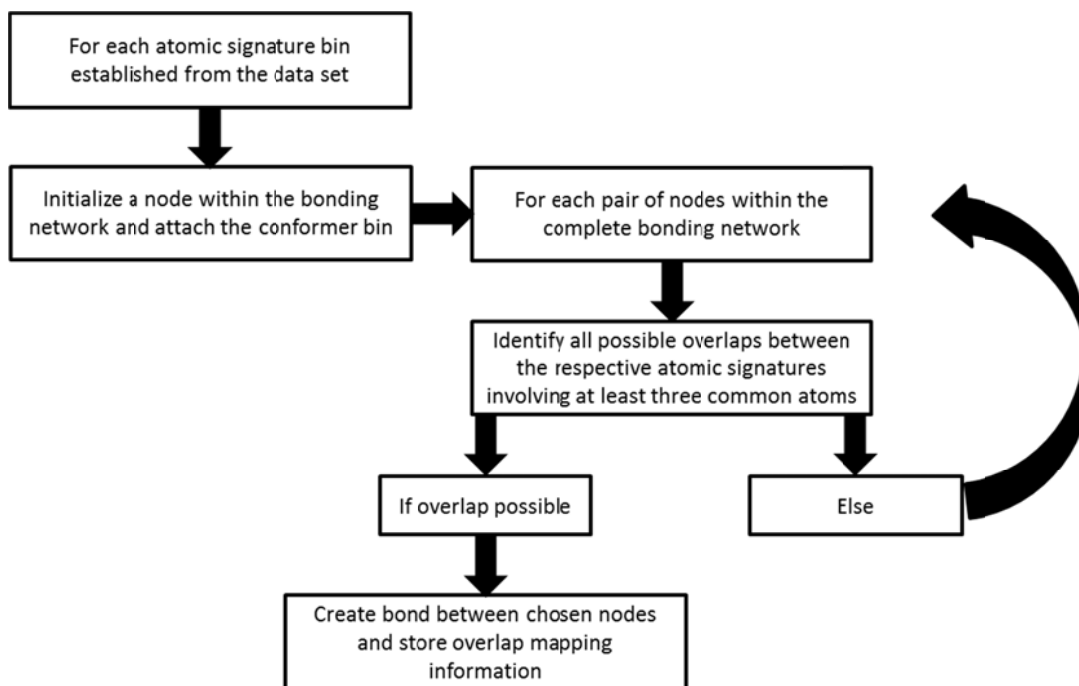


Figure A.2.1 – Visualization of bonding network creation steps.

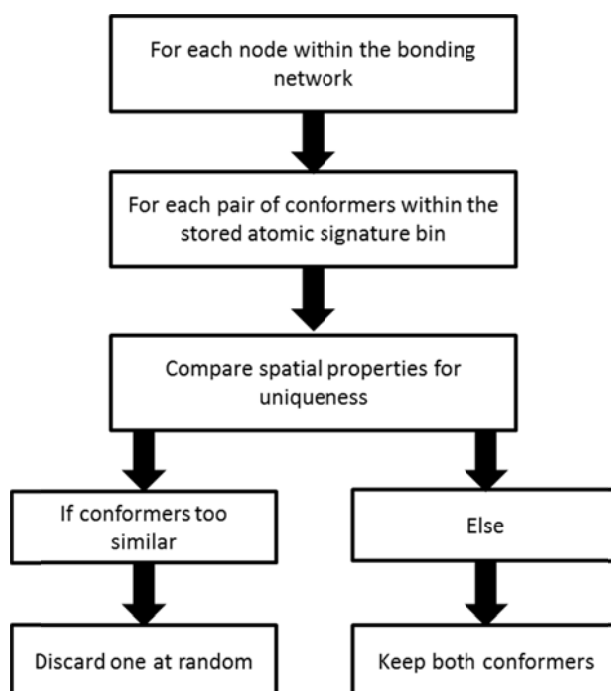


Figure A.2.2 – Flowchart depicting steps involved in bonding network compression.

The code written to achieve the creation of a bonding network, along with its compression, is shown below.

```

def create_network(self):
    self.network = nx.Graph()
    n = 0
    for subgraph_list in self.compressed_subgraphs:
        graph = subgraph_list[0]
        rebuild_list = []
        for subgraph in subgraph_list:
            rebuild_list.append(subgraph.graph['rebuild'])
        self.network.add_node(n)
        self.network.node[n]['conformers'] = subgraph_list
        self.network.node[n]['atom'] = graph.graph['atom']
        self.network.node[n]['rebuild_list'] = rebuild_list
        self.network.node[n]['canonical'] = n
        conformer_center = graph.graph['center']
  
```

```

self.network.node[n]['degree'] = graph.degree(conformer_center)
self.network.node[n]['conformer_center'] = conformer_center
self.network.node[n]['conformer_neighbors'] = [x for x in graph.neighbors(conformer_center)]
n += 1
print 'All signatures stored in network...'
print 'Establishing connectivity information...'
bonds_created = 0
possible = 0
for pair in itertools.combinations_with_replacement(self.network.nodes(data=False), 2):
    possible += 1
    bond_status = 0
    G1 = self.network.node[pair[0]]['conformers'][0]
    G1_center = self.network.node[pair[0]]['conformers'][0].graph['center']
    G2 = self.network.node[pair[1]]['conformers'][0]
    G2_center = self.network.node[pair[1]]['conformers'][0].graph['center']
    G2_ego = nx.ego_graph(G2,G2_center,radius = self.height-1)
    map_dict = defaultdict(list)
    bond_type_dict = defaultdict(int)
    for adjacent_node in G1.neighbors(G1_center):
        G1_neighbor_ego = nx.ego_graph(G1,adjacent_node,radius = self.height-1)
        nm = iso.categorical_node_match('atom', 'C')
        em = iso.numerical_edge_match('type', 1)
        GM2 = iso.GraphMatcher(G1_neighbor_ego,G2_ego,node_match = nm,edge_match = em)
        if GM2.is_isomorphic():
            bond_status = 1
            for dict_j in GM2.isomorphisms_iter():
                bond_type = G1.edge[G1_center][adjacent_node]['type']
                taken_G1_node = adjacent_node
                taken_G2_node = dict_j[G1_center]
                map_dict[(pair[0],taken_G1_node)].append((pair[1],taken_G2_node))
                map_dict[(pair[1],taken_G2_node)].append((pair[0],taken_G1_node))
    if bond_status == 1:
        bonds_created += 1
        self.network.add_edge(pair[0],pair[1])
        self.network.edge[pair[0]][pair[1]]['map_dict'] = map_dict

```



```

        self.network.edge[pair[0]][pair[1]]['type'] = bond_type

    print 'Bonding information established...'

    print possible, 'bonds were analyzed.'

    print (bonds_created)*100.00/possible, 'percent of these were feasible.'

def compress_network(self,cutoff):

    """This function compares the conformers for each node in the network and consolidates them based on a
    pairwise distance based comparison with tolerance of cutoff value specified in function call"""

    original_conformers = 0

    for node in self.network.nodes():

        old_list = self.network.node[node]['conformers'][:]

        original_conformers += len(old_list)

        for pair in itertools.combinations(old_list,2):

            status = 0

            graph_1 = pair[0]

            graph_2 = pair[1]

            nm = iso.categorical_node_match('atom', 'C')

            em = iso.numerical_edge_match('type', 1)

            GM = iso.GraphMatcher(graph_1,graph_2,node_match=nm,edge_match=em)

            if GM.is_isomorphic():

                for atom_pair in itertools.combinations(graph_1.nodes(data=False),2):

                    node0 = atom_pair[0]

                    node1 = atom_pair[1]

                    node2 = GM.mapping[node0]

                    node3 = GM.mapping[node1]

                    d1_squared = (graph_1.node[node0]['xyz'] - graph_1.node[node1]['xyz'])**2

                    d1 = (d1_squared.sum())**0.5

                    d2_squared = (graph_2.node[node2]['xyz']-graph_2.node[node3]['xyz'])**2

                    d2 = (d2_squared.sum())**0.5

                    if fabs(d1-d2) > float(cutoff):

                        status = 1

                        break

            if status == 0:

                if graph_1 in self.network.node[node]['conformers']:

                    if graph_2 in self.network.node[node]['conformers']:

                        unlucky_one = random.randint(1,2)

```

```

        if unlucky_one == 1:
            self.network.node[node]['conformers'].remove(graph_1)
        else:
            self.network.node[node]['conformers'].remove(graph_2)
    elif not graph_1 in self.network.node[node]['conformers']:
        if not graph_2 in self.network.node[node]['conformers']:
            chosen_one = random.randint(1,2)
            if chosen_one == 1:
                self.network.node[node]['conformers'].append(graph_1)
            else:
                self.network.node[node]['conformers'].append(graph_2)

    final_conformers = 0

    for node in self.network.nodes_iter():
        final_conformers += len(self.network.node[node]['conformers'])

    print 'We started with:', original_conformers, 'and consolidated down to:', final_conformers, 'conformer based on a cutoff of:', cutoff, 'Angstroms'

    print 'This means we removed:', (original_conformers - final_conformers)*100/(original_conformers),'percent from the original set.'

```

A.3 – Molecular Signature Class with Feasibility Functions

This section represents how the various molecular signatures are identified, in a deterministic manner, as well as their feasibility verification. Molecular signatures, representing collections of atomic signatures, must pass several tests of feasibility before it is verified that a complete molecule can be recreated. The tests included within this section include the graphicality equation, graph connectivity, the hand-shaking lemma, and a short test used to verify that the maximum number of one type of atomic signature has not been passed for any given molecular signature. An overview of the steps taken to develop an exhaustive list of unique structural isomers matching a given molecular signature can be found in section 3.1.7 of this dissertation.

```

import networkx as nx
import networkx.algorithms.isomorphism as iso
import math

```

```

from math import fabs
import itertools
import re
import glob
import subprocess
import linecache
from collections import defaultdict
from operator import eq
from pprint import pprint
import numpy
from operator import itemgetter
import random
valence_dictionary = {'C':4,'O':2,'H':1,'N':3,'S':2,'Cl':1,'F':1,'Br':1}
bond_dictionary = {1:1,2:2,4:1.5}
class Potential_Graph:
    def __init__(self,signatures):
        self.signatures = signatures
        self.complete_bond_list = []
        self.orbit_dict = defaultdict(list)
class AutoVivification(dict):
    """Implementation of perl's autovivification feature"""
    def __getitem__(self,item):
        try:
            return dict.__getitem__(self,item)
        except KeyError:
            value = self[item] = type(self)()
            return value
class Molecular_Signature:
    def __init__(self,height,mother_graph,atomic_signature_list):
        self.atomic_signatures = atomic_signature_list
        self.mother_graph = mother_graph
        self.height = height
        self.structural_isomers = []
    def is_repeat_satisfied(self,max):
        self.max = max

```

```

self.repeat_status = 0
for each in self.atomic_signatures:
    appearances = self.atomic_signatures.count(each)
    if appearances > self.max:
        self.repeat_status = 1
        return False
    break
else:
    pass
if self.repeat_status == 0:
    return True
def is_graphical(self):
    degree_list = [self.mother_graph.node[n]['degree'] for n in self.atomic_signatures]
    graphicality = (2*degree_list.count(4) + degree_list.count(3) - degree_list.count(1) + 2)%2
    if graphicality==0:
        return True
    else:
        return False
def is_connected(self):
    self.compressed_signatures = []
    for i in self.atomic_signatures:
        if not i in self.compressed_signatures:
            self.compressed_signatures.append(i)
    test = self.mother_graph.subgraph(self.compressed_signatures)
    if nx.is_connected(test):
        return True
    else:
        return False
# Hand Shaking Lemma
def hand_shaking_lemma(self):
    height_one_edge_list = []
    height_two_path_list = []
    height_three_path_list = []
    isomer_status = 0

```

```

for each in self.atomic_signatures:
    graph = self.mother_graph.node[each]['conformers'][0]
    center = graph.graph['center']
    center_atom = graph.node[center]['atom']
    neighbor_list = graph.neighbors(center)
    for neighbor in neighbor_list:
        first_edge = graph[center][neighbor]['type']
        neighbor_atom = graph.node[neighbor]['atom']
        height_one_edge_list.append((center_atom,first_edge,neighbor_atom))
        for third_node in [x for x in graph.neighbors(neighbor) if not x==center]:
            third_atom = graph.node[third_node]['atom']
            second_edge = graph[neighbor][third_node]['type']
            height_two_path_list.append((center_atom,first_edge,neighbor_atom,second_edge,third_atom))
            if self.height == 3:
                for fourth_node in [x for x in graph.neighbors(third_node) if not x == neighbor]:
                    fourth_atom = graph.node[fourth_node]['atom']
                    third_edge = graph[third_node][fourth_node]['type']

height_three_path_list.append((center_atom,first_edge,neighbor_atom,second_edge,third_atom,third_edge,fourth_atom))

height_one_edge_set = set(height_one_edge_list)
for each_bond in height_one_edge_set:
    if each_bond[0] == each_bond[2]:
        if height_one_edge_list.count(each_bond)%2 == 0:
            pass
        else:
            return False
    else:
        matched_bond = (each_bond[2],each_bond[1],each_bond[0])
        if height_one_edge_list.count(each_bond) == height_one_edge_list.count(matched_bond):
            pass
        else:
            return False

height_two_path_set = set(height_two_path_list)
for each_path in height_two_path_set:
    if (each_path[0] == each_path[2]) and (each_path[2] == each_path[4]) and (each_path[1] == each_path[3]):
        if height_two_path_list.count(each_path)%2 == 0:

```

```

        pass
    else:
        return False
    else:
        if (height_two_path_list.count((each_path[4],each_path[3],each_path[2],each_path[1],each_path[0])) ==
            height_two_path_list.count(each_path)):
            pass
        else:
            return False
height_three_path_set = set(height_three_path_list)
for each_path in height_three_path_set:
    if ((each_path[0] == each_path[2]) and (each_path[2] == each_path[4]) and (each_path[4] == each_path[6]) and
        (each_path[1] == each_path[3]) and (each_path[3] == each_path[5])):
        if (height_three_path_list.count(each_path))% 2 == 0:
            pass
        else:
            return False
    else:
        if (height_three_path_list.count((each_path[6],each_path[5],each_path[4],each_path[3],each_path[2],each_path[1],
            each_path[0]))==height_three_path_list.count(each_path)):
            pass
        else:
            return False
return True
def exhaustive_structural_isomers(self):
    def saturation_algorithm(graph):
        connected_components = nx.connected_components(graph)
        length = len(connected_components)
        unsat_deg_list = []
        for x in range(len(connected_components)):
            total_degree_of_unsat = 0
            for node in connected_components[x]:
                desired_sat = valence_dictionary[graph.node[node]['atom']]
                sig_one = graph.node[node]['sig']
                actual_sat = 0

```

```

if not graph.node[node].keys():
    print 'Boolean worked'
    actual_sat = 0
else:
    for neighbor in graph.neighbors(node):
        sig_two = graph.node[neighbor]['sig']
        bond_type = self.mother_graph.edge[sig_one][sig_two]['type']
        actual_sat += bond_dictionary[bond_type]
    degree_of_unsat = desired_sat - actual_sat
    if degree_of_unsat < 0:
        return 'saturated subgraph'
    graph.graph['unsat_dict'][node]=degree_of_unsat
    total_degree_of_unsat += degree_of_unsat
    unsat_deg_list.append(total_degree_of_unsat)
if (0 in unsat_deg_list) and (length == 1):
    return 'solution found'
elif 0 in unsat_deg_list:
    return 'saturated subgraph'
else:
    automorphism_groups = []
    free_atom_lists = []
    smallest_degree = min(unsat_deg_list)
    # Just pulls the first connected component with the min. degree of unsat
    desired_index = unsat_deg_list.index(smallest_degree)
    desired_subgraph = graph.subgraph(connected_components[desired_index])
    unsat_atoms = [x for x in desired_subgraph.nodes() if graph.graph['unsat_dict'][x] > 0]
    if len(unsat_atoms) == 0:
        return 'saturated subgraph'
    unsat_atom_1 = unsat_atoms[0]
    free_atom_lists.append(unsat_atom_1)
    automorphism_groups.append(connected_components[desired_index])
    for component_list in connected_components:
        if connected_components.index(component_list) != desired_index:
            tested_subgraph = graph.subgraph(component_list)
            nm = iso.numerical_node_match('sig', 1)

```

```

GM = iso.GraphMatcher(desired_subgraph, tested_subgraph, node_match = nm)

if GM.is_isomorphic():
    automorphism_groups.append(component_list)
    mapped_atom_1 = GM.mapping[unsat_atom_1]
    free_atom_lists.append(mapped_atom_1)
return (automorphism_groups, free_atom_lists)

def exhaustive_structural_isomers(self):
    def saturation_algorithm(graph):
        connected_components = nx.connected_components(graph)
        length = len(connected_components)
        unsat_deg_list = []
        for x in range(len(connected_components)):
            total_degree_of_unsat = 0
            for node in connected_components[x]:
                desired_sat = valence_dictionary[graph.node[node]['atom']]
                sig_one = graph.node[node]['sig']
                actual_sat = 0
                if not graph.node[node].keys():
                    print 'Boolean worked'
                    actual_sat = 0
                else:
                    for neighbor in graph.neighbors(node):
                        sig_two = graph.node[neighbor]['sig']
                        bond_type = self.mother_graph.edge[sig_one][sig_two]['type']
                        actual_sat += bond_dictionary[bond_type]
            degree_of_unsat = desired_sat - actual_sat
            if degree_of_unsat < 0:
                return 'saturated subgraph'
            graph.graph['unsat_dict'][node] = degree_of_unsat
            total_degree_of_unsat += degree_of_unsat
        unsat_deg_list.append(total_degree_of_unsat)
    if (0 in unsat_deg_list) and (length == 1):
        return 'solution found'
    elif 0 in unsat_deg_list:
        return 'saturated subgraph'

```



```

else:
    automorphism_groups = []
    free_atom_lists = []
    smallest_degree = min(unsat_deg_list)
    # Just pulls the first connected component with the min. degree of unsat
    desired_index = unsat_deg_list.index(smallest_degree)
    desired_subgraph = graph.subgraph(connected_components[desired_index])
    unsat_atoms = [x for x in desired_subgraph.nodes() if graph.graph['unsat_dict'][x] > 0]
    if len(unsat_atoms) == 0:
        return 'saturated subgraph'
    unsat_atom_1 = unsat_atoms[0]
    free_atom_lists.append(unsat_atom_1)
    automorphism_groups.append(connected_components[desired_index])
    for component_list in connected_components:
        if connected_components.index(component_list) != desired_index:
            tested_subgraph = graph.subgraph(component_list)
            nm = iso.numerical_node_match('sig', 1)
            GM = iso.GraphMatcher(desired_subgraph, tested_subgraph, node_match = nm)
            if GM.is_isomorphic():
                automorphism_groups.append(component_list)
                mapped_atom_1 = GM.mapping[unsat_atom_1]
                free_atom_lists.append(mapped_atom_1)
    return (automorphism_groups, free_atom_lists)
def generate_new_graphs(graph, automorphism_list, mapped_unsat_atoms_list):
    new_graph_list = []
    automor_count = len(automorphism_list)
    possible_bonded_to_nodes = []
    node1 = mapped_unsat_atoms_list[0]
    for keyi in graph.graph['m_dict'].keys():
        for seti in graph.graph['m_dict'][keyi].keys():
            if node1 in seti:
                for x in range(graph.graph['m_dict'][keyi][seti]):
                    possible_bonded_to_nodes.append(keyi)
    check_list = []
    for group in itertools.combinations(possible_bonded_to_nodes, automor_count):

```

```

new_graph = graph.copy()
check_group = []
for x in range(len(group)):
    sig = graph.node[group[x]]['sig']
    check_group.append(sig)
check_group.sort()
if not check_group in check_list:
    check_list.append(check_group)
    for x in range(len(group)):
        node1 = group[x]
        node2 = mapped_unsat_atoms_list[x]
        new_graph.add_edge(node1,node2)
        for set_j in new_graph.graph['m_dict'][node1]:
            if node2 in set_j:
                graph.graph['m_dict'][node1][set_j] += -1
        for set_k in new_graph.graph['m_dict'][node2]:
            if node1 in set_k:
                graph.graph['m_dict'][node2][set_k] += -1
    match_status = 0
    for graph_2 in new_graph_list:
        nm = iso.numerical_node_match('sig', 1)
        GM2 = iso.GraphMatcher(new_graph,graph_2,node_match = nm)
        if GM2.is_isomorphic():
            match_status = 1
            break
    if match_status == 0:
        new_graph_list.append(new_graph)
return new_graph_list

# This first step creates a bonding dict with full signatures references
signature_set = set(self.atomic_signatures)
bonding_dict = defaultdict(set)
for pair in itertools.combinations_with_replacement(signature_set,2):
    if self.mother_graph.has_edge(pair[0],pair[1]):
        dict_i = self.mother_graph.edge[pair[0]][pair[1]]['map_dict']
        for key,entry in dict_i.items():

```

```

        for sig in entry:
            bonding_dict[key].add(sig)
            bonding_dict[sig].add(key)
# Now I am creating the n and m occurrence dictionaries
# I need to create the max occurrence dictionary
m_dict = defaultdict(lambda:defaultdict(int))
for entry,list_i in bonding_dict.items():
    second_list = []
    for each in list_i:
        second_list.append(each[0])
    second_set = set(second_list)
    second_tuple = tuple(second_set)
    m_dict[entry[0]][second_tuple] += 1
# I will be creating a graph with no bonds to start with as a 'base graph'
# This graph will maintain the equivalent n12 and m12 dictionaries as referenced in Faulon's paper
sig_to_node_dictionary = defaultdict(list)
base_graph = nx.Graph()
n = 0
for each in signature_set:
    for x in range(self.atomic_signatures.count(each)):
        base_graph.add_node(n)
        base_graph.node[n]['sig'] = each
        base_graph.node[n]['atom'] = self.mother_graph.node[each]['atom']
        sig_to_node_dictionary[each].append(n)
        n += 1
base_graph.graph['unsat_dict'] = defaultdict(int)
# Now I need to translate the previously generated occurrence dictionaries into the given base graph with appropriate node references
base_graph.graph['m_dict'] = defaultdict(lambda:defaultdict(int))
for sig1 in m_dict.keys():
    for node1 in sig_to_node_dictionary[sig1]:
        for sig2_list in m_dict[sig1].keys():
            node2_list = []
            for sig2 in sig2_list:
                for node2 in sig_to_node_dictionary[sig2]:
                    node2_list.append(node2)

```

```

        node2_set = set(node2_list)
        node2_tuple = tuple(node2_set)
        base_graph.graph['m_dict'][node1][node2_tuple] = m_dict[sig1][sig2_list]
# I will create a simple bonding dict here for estimating all possible bonds
simple_bonds = defaultdict(set)
for node1 in base_graph.nodes():
    sig1 = base_graph.node[node1]['sig']
    for neighbor in self.mother_graph.neighbors(sig1):
        for node2 in base_graph.nodes():
            if neighbor == base_graph.node[node2]['sig']:
                simple_bonds[node1].add(node2)
# This is where I'm actually running the while loop to generate all graphs
final_isomers = 0
old_list = []
self.final_isomers = []
self.potential_isomers = []
z = saturation_algorithm(base_graph)
y = generate_new_graphs(base_graph,z[0],z[1])
old_list.extend(y)
iterations = 0
while len(old_list) > 0 and iterations < 150:
    iterations += 1
    for graph in old_list:
        z = saturation_algorithm(graph)
        if z == 'solution found':
            self.final_isomers.append(graph)
        elif z == 'saturated subgraph':
            pass
        elif type(z) is tuple:
            y = generate_new_graphs(graph,z[0],z[1])
            self.potential_isomers.extend(y)
    old_list = self.potential_isomers[:]
    self.potential_isomers[:] = []
    if len(self.final_isomers) > 0:
        break

```

else:

```
print 'Final number of structural isomers:', len(self.final_isomers)
```

A.4 – Genetic Algorithm

The genetic algorithm was proposed to handle larger and more complex CAMD problems and its coding is shown in this section. In addition to generation of a starting population, the mutation operators are also included. A unique bonding network, specific to the GA approach, must also be created and this code is included as well. Finally, the generation of conformational isomers from atomic signatures is shown. The generalized approach to implementing mutation operators is shown in Figure A.4.1.

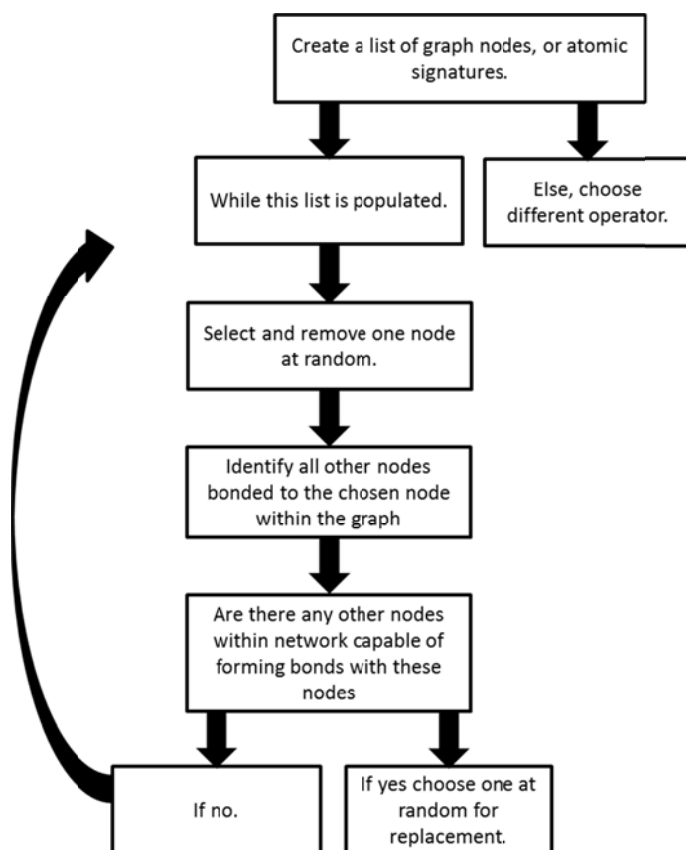


Figure A.4.1 – Generalized approach to implementing mutation operators.

The approach taken to identify candidates feasible for crossover mutation can be found in section 3.2.4 of this dissertation. The code written for each of these operations is shown below.

```
import networkx as nx
import networkx.algorithms.isomorphism as iso
import math
from math import fabs
import itertools
import re
import glob
import subprocess
import linecache
from collections import defaultdict
from operator import eq
from pprint import pprint
import numpy
from operator import itemgetter
from random import *
from random import randint
from bisect import bisect
sin = numpy.sin
cos = numpy.cos
valence_dictionary = {'C':4,'O':2,'H':1,'N':3,'S':2,'Cl':1,'F':1,'Br':1,'P':5}
bond_dictionary = {1:1,2:2,4:1.5}
#####
# Just a couple of algebraic operations
#####
def unit_vector(vector):
    return vector/numpy.linalg.norm(vector)
def angle_between(v1, v2):
    v1_u = unit_vector(v1)
    v2_u = unit_vector(v2)
    angle = numpy.arccos(numpy.dot(v1_u,v2_u))
```

```

if numpy.isnan(angle):
    if (v1_u == v2_u).all():
        return 0.0
    else:
        return numpy.pi
return angle

#####

# These are some operators for the selection process in genetic algorithm

#####

def weighted_choice(choices):
    values, weights = zip(*choices)
    total = 0
    cum_weights = []
    for w in weights:
        total += w
        cum_weights.append(total)
    x = random() * total
    i = bisect(cum_weights, x)
    return values[i]

def choose_operation(network,city,fitness_list,optimal_size,beta,split):
    randy = random()
    if randy > split:
        # This is a mutation and I must choose one based on fitness
        chosen_index = weighted_choice(fitness_list)
        chosen_graph = city[chosen_index]
        chosen_graph_size = len( chosen_graph.graph['conformers'][0].nodes() )
        if chosen_graph_size > optimal_size*(1 + beta):
            return ('mutation','deletion',chosen_graph.copy())
        elif chosen_graph_size < optimal_size*(1 - beta):
            return ('mutation','insertion',chosen_graph.copy())
        else:
            return ('mutation','node',chosen_graph.copy())
    else:
        found_pair = 'no'
        while found_pair == 'no':

```

```

chosen_index_1 = weighted_choice(fitness_list)
chosen_index_2 = weighted_choice(fitness_list)
while chosen_index_2 == chosen_index_1:
    chosen_index_2 = weighted_choice(fitness_list)
chosen_graph_1 = city[chosen_index_1].copy()
chosen_graph_2 = city[chosen_index_2].copy()
# I must verify that crossover is possible first
edges_in_1 = chosen_graph_1.edges()
edges_in_2 = chosen_graph_2.edges()
possible_edge_pairs = list( itertools.product( edges_in_1 , edges_in_2 ) )
for edge_pair in possible_edge_pairs:
    # Try switching node ones
    edge_in_1 = edge_pair[0]
    edge_in_2 = edge_pair[1]
    node_1_2 = edge_in_2[0]
    node_2_2 = edge_in_2[1]
    node_1_1 = edge_in_1[0]
    node_2_1 = edge_in_1[1]
    sig_1_2 = chosen_graph_2.node[node_1_2]['sig']
    sig_2_2 = chosen_graph_2.node[node_2_2]['sig']
    sig_1_1 = chosen_graph_1.node[node_1_1]['sig']
    sig_2_1 = chosen_graph_1.node[node_2_1]['sig']
    if network.has_edge(sig_1_2,sig_2_1):
        node_1_2_unsats = [ x[1] for x in chosen_graph_2.edge[edge_in_2[0]][edge_in_2[1]]['unsat_info'] if x[0] ==
            sig_1_2 ]
        node_2_1_unsats = [ x[1] for x in chosen_graph_1.edge[edge_in_1[0]][edge_in_1[1]]['unsat_info'] if x[0] ==
            sig_2_1 ]
        move_list = [ x for x in chosen_graph_1.graph['move_list'] if node_1_1 in x and node_2_1 in x ]
        if len(move_list) == 0:
            break
        move = [ x for x in chosen_graph_1.graph['move_list'] if node_1_1 in x and node_2_1 in x ][0]
        if move[1] == node_2_1:
            # This means that the maps should be keyed by (sig 2 in 1)
            for map_key in network.edge[sig_1_2][sig_2_1]['map_dict'].keys():
                if map_key[0] == sig_2_1:
                    for map in network.edge[sig_1_2][sig_2_1]['map_dict'][map_key]:

```



```

# If it's keyed by sig2in1 then those will be the entries
map_1_items = [ x[1] for x in map.items() ]
missing_node_2_1_unsats = [ x for x in node_2_1_unsats if x not in map_1_items ]
missing_node_1_2_unsats = [ x for x in node_1_2_unsats if x not in map.keys() ]
if len(missing_node_2_1_unsats) == 0 and len(missing_node_1_2_unsats) == 0:
    found_pair = 'yes'
    return ( 'crossover', chosen_graph_2, edge_in_2, node_1_2,
            chosen_graph_1, edge_in_1, node_2_1, map )
else:
    # This means that the maps should be keyed by (sig 1 in 2)
    for map_key in network.edge[sig_1_2][sig_2_1]['map_dict'].keys():
        if map_key[0] == sig_1_2:
            for map in network.edge[sig_1_2][sig_2_1]['map_dict'][map_key]:
                # If it's keyed by sig1in2 then those will be the entries
                map_1_items = [ x[1] for x in map.items() ]
                missing_node_1_2_unsats = [ x for x in node_1_2_unsats if x not in map_1_items ]
                missing_node_2_1_unsats = [ x for x in node_2_1_unsats if x not in map.keys() ]
                if len(missing_node_2_1_unsats) == 0 and len(missing_node_1_2_unsats) == 0:
                    found_pair = 'yes'
                    return ( 'crossover', chosen_graph_1, edge_in_1, node_2_1, chosen_graph_2,
                            edge_in_2, node_1_2, map )

def create_network(self):
    self.network = nx.Graph()
    n = 0
    for subgraph_list in self.compressed_subgraphs:
        graph = subgraph_list[0]
        self.network.add_node(n)
        self.network.node[n]['conformers'] = subgraph_list
        self.network.node[n]['atom'] = graph.graph['atom']
        self.network.node[n]['atom_count'] = len(graph.nodes())
        conformer_center = graph.graph['center']
        self.network.node[n]['degree'] = graph.degree(conformer_center)
        self.network.node[n]['conformer_center'] = conformer_center
        self.network.node[n]['unsaturated_nodes'] = []
        # This step will determine which neighbors to the central
        for node_1 in graph.nodes():

```

```

neighbors = graph.neighbors(node_1)
required_valence = valence_dictionary[graph.node[node_1]['atom']]
node_1_degree = 0
for node_2 in graph.neighbors(node_1):
    node_1_degree += bond_dictionary[graph.edge[node_1][node_2]['type']]
if node_1_degree < required_valence:
    self.network.node[n]['unsaturated_nodes'].append(node_1)
n += 1
print 'All signatures stored in network...'
print 'Establishing connectivity information...'
bonds_created = 0
possible = 0
for pair in itertools.combinations_with_replacement(self.network.nodes(), 2):
    bond_status = 0
    possible += 1
    map_dict = defaultdict(list)
    map_dict.clear()
    sig_a = self.network.node[pair[0]]['conformers'][0]
    sig_a_center = self.network.node[pair[0]]['conformers'][0].graph['center']
    sig_b = self.network.node[pair[1]]['conformers'][0]
    sig_b_center = self.network.node[pair[1]]['conformers'][0].graph['center']
    # This is for one side where sig_a is G2 and sig_b is G1
    G1 = sig_b
    for unsat_node in self.network.node[pair[0]]['unsaturated_nodes']:
        G2 = nx.ego_graph(sig_a, unsat_node, radius = 2)
        nm = iso.categorical_node_match('atom', 'C')
        em = iso.numerical_edge_match('type', 1)
        GM1 = iso.GraphMatcher(G1, G2, node_match = nm, edge_match = em)
        if GM1.subgraph_is_isomorphic():
            for dict_j in GM1.subgraph_isomorphisms_iter():
                if ( len(dict_j.keys()) >= 3 and sig_b_center in dict_j.keys() and dict_j[sig_b_center] == unsat_node):
                    bond_status = 1
                    map_dict[(pair[0], unsat_node)].append(dict_j)
    # This is for the other side where sig_b is G2 and sig_a is G1
    G1 = sig_a

```

```

for unsat_node in self.network.node[pair[1]]['unsaturated_nodes']:
    G2 = nx.ego_graph(sig_b,unsat_node,radius = 2)
    nm = iso.categorical_node_match('atom', 'C')
    em = iso.numerical_edge_match('type', 1)
    GM2 = iso.GraphMatcher(G1,G2,node_match = nm,edge_match = em)
    if GM2.subgraph_is_isomorphic():
        for dict_k in GM2.subgraph_isomorphisms_iter():
            if (len(dict_k.keys()) >= 3 and sig_a_center in dict_k.keys() and dict_k[sig_a_center] == unsat_node):
                bond_status = 1
                map_dict[(pair[1],unsat_node)].append(dict_k)
if bond_status == 1:
    bonds_created += 1
    self.network.add_edge(pair[0],pair[1])
    self.network.edge[pair[0]][pair[1]]['map_dict'] = map_dict
print 'Bonding information established...'
print possible, 'bonds were analyzed.'
print (bonds_created)*100.00/possible , 'percent of these were feasible.'

#####

def create_starting_population(self,size,atom_count_lower,atom_count_upper):
    self.size = size
    self.city = []
    size_list = [x for x in range(atom_count_lower,atom_count_upper+1)]
    for x in range(self.size):
        first_sig = choice(self.network.nodes())
        # I will use indices in case a signature is used more than once
        index = 0
        target_size = choice(size_list)
        signature_graph = nx.Graph()
        signature_graph.graph['target_size'] = target_size
        signature_graph.add_node(index)
        signature_graph.node[index]['sig'] = first_sig
        signature_graph.node[index]['unsat_nodes'] = self.network.node[first_sig]['unsaturated_nodes'][:]
        signature_graph.graph['current_size'] = self.network.node[first_sig]['atom_count']
        signature_graph.graph['index'] = 1
        signature_graph.graph['conformers'] = []

```

```

signature_graph.graph['move_list'] = []

self.city.append(signature_graph)

print 'Initial population size:', len(self.city)

for graph in self.city:

    total_unsat = sum([len(graph.node[x]['unsat_nodes']) for x in graph.nodes()])

    while total_unsat > 0:

        P_cap = 1 - ((graph.graph['target_size'] - graph.graph['current_size']) / (float(graph.graph['target_size'])))

        if P_cap >= 1.0:

            cap = 'yes'

        else:

            rand = random()

            if rand <= P_cap:

                cap = 'yes'

            else:

                cap = 'no'

        # First I want to identify potential nodes to bond with their new degree of unsat

        unsat_nodes = []

        for node in graph.nodes():

            if len(graph.node[node]['unsat_nodes']) > 0:

                for node2 in graph.node[node]['unsat_nodes']:

                    unsat_nodes.append((node, graph.node[node]['sig'], node2))

        chosen_unsat_node = choice(unsat_nodes)

        graph.node[chosen_unsat_node[0]]['unsat_nodes'].remove(chosen_unsat_node[2])

        chosen_unsat_point = (chosen_unsat_node[1], chosen_unsat_node[2])

        potential_new_node_list = []

        new_unsat_degree_list = []

        for neighbor in self.network.neighbors(chosen_unsat_node[1]):

            if (chosen_unsat_point in self.network.edge[chosen_unsat_node[1]][neighbor]['map_dict'].keys() and
                len(self.network.node[neighbor]['unsaturated_nodes']) > 0):

                potential_new_node_list.append(neighbor)

                new_unsat_degree_list.append(len(self.network.node[neighbor]['unsaturated_nodes']) - 1)

        if cap == 'no' and len([x for x in new_unsat_degree_list if x != 0]) == 0:

            cap = 'yes'

        if cap == 'yes':

            best = min(new_unsat_degree_list)

            best_options = [new_unsat_degree_list.index(x) for x in new_unsat_degree_list if x == best]

```

```

chosen_partner = potential_new_node_list[choice(best_options)]

graph.add_node(graph.graph['index'])

graph.node[graph.graph['index']]['sig'] = chosen_partner

graph.node[graph.graph['index']]['unsat_nodes'] = self.network.node[chosen_partner]['unsaturated_nodes'][:]

graph.add_edge( chosen_unsat_node[0] , graph.graph['index'] )

graph.edge[chosen_unsat_node[0]][graph.graph['index']]['unsat_info'] = []

graph.edge[chosen_unsat_node[0]][graph.graph['index']]['unsat_info'].append(
(chosen_unsat_node[1],chosen_unsat_node[2]) )

# If there are multiple mappings here I could choose one at random or have some other criteria
chosen_map = self.network.edge[chosen_unsat_node[1]][chosen_partner]['map_dict']
                [chosen_unsat_point][0].copy()

unsats_to_remove = [x for x in chosen_map.keys() if x in self.network.node[chosen_partner]
                ['unsaturated_nodes']]

graph.graph['move_list'].append((graph.graph['index'],chosen_unsat_node[0],chosen_map))

for unsat_node in unsats_to_remove:

    graph.node[graph.graph['index']]['unsat_nodes'].remove(unsat_node)

    graph.edge[chosen_unsat_node[0]][graph.graph['index']]['unsat_info'].append(
    (chosen_partner,unsat_node) )

nodes_added = self.network.node[chosen_partner]['atom_count'] - len(chosen_map.keys())

graph.graph['current_size'] += nodes_added

graph.graph['index'] += 1

elif cap == 'no':

    best_options = []

    index = 0

    for x in new_unsat_degree_list:

        if x != 0:

            best_options.append(potential_new_node_list[index])

            index += 1

    chosen_partner = choice(best_options)

    graph.add_node(graph.graph['index'])

    graph.node[graph.graph['index']]['unsat_nodes'] = self.network.node[chosen_partner]['unsaturated_nodes'][:]

    graph.node[graph.graph['index']]['sig'] = chosen_partner

    graph.add_edge( chosen_unsat_node[0] , graph.graph['index'] )

    graph.edge[chosen_unsat_node[0]][graph.graph['index']]['unsat_info'] = []

    graph.edge[chosen_unsat_node[0]][graph.graph['index']]['unsat_info'].append( (chosen_unsat_node[1],
    chosen_unsat_node[2]) )

    # If there are lots of mappings here I could choose one at random or have some other criteria

```

```

chosen_map = self.network.edge[chosen_unsat_node[1]][chosen_partner]['map_dict']
[chosen_unsat_point][0].copy()

unsats_to_remove = [ x for x in chosen_map.keys() if x in self.network.node[chosen_partner]
[unsaturated_nodes] ]

graph.graph['move_list'].append((graph.graph['index'],chosen_unsat_node[0],chosen_map))

for unsat_node in unsats_to_remove:

    graph.node[graph.graph['index']]['unsat_nodes'].remove(unsat_node)

    graph.edge[chosen_unsat_node[0]][graph.graph['index']]['unsat_info'].append(
(chosen_partner,unsat_node) )

nodes_added = self.network.node[chosen_partner]['atom_count'] - len(chosen_map.keys())

graph.graph['current_size'] += nodes_added

graph.graph['index'] += 1

total_unsat = sum([len(graph.node[x]['unsat_nodes']) for x in graph.nodes()])

else:

    print 'Graph created with size:', graph.graph['current_size']

iterations = 0

mapping_condition = 'yes'

zero_list = [ x[0] for x in graph.graph['move_list'] ]

one_list = [ x[1] for x in graph.graph['move_list'] ]

zero_set = set ( zero_list )

for each in zero_set:

    if each in one_list:

        if one_list.index(each) < zero_list.index(each):

            shuffle(graph.graph['move_list'])

            mapping_condition = 'no'

            break

while mapping_condition == 'no':

    iterations += 1

    mapping_condition = 'yes'

    zero_list = [ x[0] for x in graph.graph['move_list'] ]

    one_list = [ x[1] for x in graph.graph['move_list'] ]

    zero_set = set ( zero_list )

    for each in zero_set:

        if each in one_list:

            if one_list.index(each) < zero_list.index(each):

                shuffle(graph.graph['move_list'])

```

```

        mapping_condition = 'no'
        break
def create_new_population(self,new_size,beta,split):
    node_mutation_count = 0
    insertion_mutation_count = 0
    deletion_mutation_count = 0
    crossover_count = 0
    new_pop = 0
    new_city = []
    while new_pop < new_size:
        operation = choose_operation(self.network,self.city,self.fitness_list,self.optimal_size,beta,split)
        if operation[0] == 'mutation' and operation[1] == 'node':
            # This is going to be where I perform a node mutation
            # First I will choose a random node
            original_graph = operation[2]
            nodes = original_graph.nodes()[:]
            found_node = 'no'
            graph_added = 'no'
            while len(nodes) > 0 and graph_added == 'no':
                shuffle(nodes)
                random_node = nodes.pop()
                random_nodes_sig = original_graph.node[random_node]['sig']
                edges_from_random_node = nx.edges(original_graph,random_node)
                unsat_to_satisfy = []
                unsat_to_satisfy_dict = defaultdict(int)
                for edge in edges_from_random_node:
                    for unsat in original_graph.edge[edge[0]][edge[1]]['unsat_info']:
                        if unsat[0] != random_nodes_sig:
                            unsat_to_satisfy.append(unsat)
            # The unsat to satisfy might occur more than once...if so, I must find a node which can accomodate this
            unsat_to_satisfy_set = set(unsat_to_satisfy)
            for unsat in unsat_to_satisfy_set:
                unsat_to_satisfy_dict[unsat] = unsat_to_satisfy.count(unsat)
            list_of_unsat_potential_lists = []
            for unsat in unsat_to_satisfy_dict.keys():

```

```

unsat_potentials = []
for neighbor in self.network.neighbors(unsat[0]):
    if neighbor != random_nodes_sig:
        if unsat in self.network.edge[unsat[0]][neighbor]['map_dict'].keys():
            if len(self.network.edge[unsat[0]][neighbor]['map_dict']) >= unsat_to_satisfy_dict[unsat]:
                unsat_potentials.append(neighbor)
list_of_unsat_potential_lists.append(unsat_potentials)
if len(list_of_unsat_potential_lists) > 1:
    optimal_nodes = []
    for x in list_of_unsat_potential_lists[0]:
        present = 'no'
        for y in range(len(list_of_unsat_potential_lists)):
            if x in list_of_unsat_potential_lists[y]:
                present = 'yes'
            else:
                present = 'no'
                break
        if present == 'yes':
            optimal_nodes.append(x)
    elif len(list_of_unsat_potential_lists) == 1:
        optimal_nodes = set(list_of_unsat_potential_lists[0])
    elif len(list_of_unsat_potential_lists) == 0:
        continue
move_list_tuples = [ x for x in original_graph.graph['move_list'] if x[0] == random_node or x[1] ==
                    random_node]
while len(optimal_nodes) > 0 and graph_added == 'no':
    chosen_replacement_node = optimal_nodes.pop()
    graph = original_graph.copy()
    new_move_list = [ x for x in graph.graph['move_list'] if x not in move_list_tuples ]
    # Depending on the first two entries of the map i need to decide if the map is keyed or entried by the
    new node
    for edge in move_list_tuples:
        if edge[0] == random_node:
            # This means the dictionary list is keyed by the other node
            other_node = edge[1]
            other_sig = graph.node[other_node]['sig']

```



```

other_unsats = [ x[1] for x in graph.edge[edge[0]][edge[1]]['unsat_info'] if x[0] == other_sig ]
map_found = 'no'
for other_unsat in other_unsats:
    for map in self.network.edge[chosen_replacement_node][other_sig]
        ['map_dict'][(other_sig,other_unsat)]:
        map_items_1 = [ x[1] for x in map.items() ]
        missing_other_unsats = [ x for x in other_unsats if x not in map_items_1 ]
        if len(missing_other_unsats) == 0:
            map_found = 'yes'
            graph.edge[edge[0]][edge[1]]['unsat_info'][:] = []
            new_randy_unsats = [ x for x in map.keys() if x in self.network.node
                [chosen_replacement_node]['unsaturated_nodes']]
            for randy_unsat in new_randy_unsats:
                graph.edge[edge[0]][edge[1]]['unsat_info'].append(
                    ( chosen_replacement_node , randy_unsat ))
            break
        if map_found == 'yes':
            break
if map_found == 'yes':
    for other_unsat in other_unsats:
        graph.edge[edge[0]][edge[1]]['unsat_info'].append( (other_sig,other_unsat) )
    to_remove = [ x for x in graph.graph['move_list'] if x[0] == edge[0] and x[1] ==
        edge[1] ][0]
    new_move_list.append( ( random_node, other_node, map.copy() ) )
else:
    # This means the dictionary list is keyed by the random node
    other_node = edge[0]
    other_sig = graph.node[other_node]['sig']
    other_unsats = [ x[1] for x in graph.edge[edge[0]][edge[1]]['unsat_info'] if x[0] == other_sig ]
    # Now I'll search through all map keys and maps to search for one which covers all other
    unsats
    map_found = 'no'
    for random_unsat in self.network.node[chosen_replacement_node]['unsaturated_nodes']:
        if self.network.has_edge(chosen_replacement_node,other_sig):
            for map in self.network.edge[chosen_replacement_node][other_sig]['map_dict']
                [(chosen_replacement_node,random_unsat)]:
                uncovered_other_unsats = [ x for x in other_unsats if x not in map.keys() ]
                if len(uncovered_other_unsats) == 0:

```

```

        map_found = 'yes'
        break
    if map_found == 'yes':
        break
if map_found == 'yes':
    graph.edge[edge[0]][edge[1]]['unsat_info'][:] = []
    for other_unsat in other_unsats:
        graph.edge[edge[0]][edge[1]]['unsat_info'].append( (other_sig, other_unsat) )
    random_unsats = [ x for x in map.keys() if x in self.network.node
                     [chosen_replacement_node]['unsaturated_nodes'] ]
    for randy_unsat in random_unsats:
        graph.edge[edge[0]][edge[1]]['unsat_info'].append(
            (chosen_replacement_node, randy_unsat) )
    to_remove = [ x for x in graph.graph['move_list'] if x[1] == random_node and x[0] ==
                 other_node ][0]
    new_move_list.append( ( other_node, random_node, map.copy() ) )
if map_found == 'yes':
    graph.graph['move_list'] = new_move_list
    mapping_condition = 'yes'
    zero_list = [ x[0] for x in graph.graph['move_list'] ]
    one_list = [ x[1] for x in graph.graph['move_list'] ]
    zero_set = set ( zero_list )
    for each in zero_set:
        if each in one_list:
            if one_list.index(each) < zero_list.index(each):
                shuffle(graph.graph['move_list'])
                mapping_condition = 'no'
                break
    while mapping_condition == 'no':
        mapping_condition = 'yes'
        zero_list = [ x[0] for x in graph.graph['move_list'] ]
        one_list = [ x[1] for x in graph.graph['move_list'] ]
        zero_set = set ( zero_list )
        for each in zero_set:
            if each in one_list:
                if one_list.index(each) < zero_list.index(each):
                    shuffle(graph.graph['move_list'])

```

```

        mapping_condition = 'no'
        break

    graph_added = 'yes'
    graph.graph['conformers'][:] = []
    graph.node[random_node]['sig'] = chosen_replacement_node
    move_test = 'good'
    for move in graph.graph['move_list']:
        if move[0] not in graph.nodes() or move[1] not in graph.nodes():
            move_test = 'bad'
            print 'Uh oh there was a move without a bond for a node mutation graph'
            break
        if move_test == 'good':
            print 'New graph added to city by node mutation.'
            new_city.append(graph)
            new_pop += 1
            node_mutation_count += 1
    else:
        continue

```

Deletion Mutation

```

elif operation[0] == 'mutation' and operation[1] == 'deletion':
    original_graph = operation[2]
    nodes = original_graph.nodes()[:]
    found_node = 'no'
    graph_added = 'no'
    while len(nodes) > 0 and graph_added == 'no':
        shuffle(nodes)
        random_node = nodes.pop()
        random_nodes_sig = original_graph.node[random_node]['sig']
        edges_from_random_node = nx.edges(original_graph, random_node)
        unsat_to_satisfy = []
        unsat_to_satisfy_dict = defaultdict(int)
        for edge in edges_from_random_node:
            for unsat in original_graph.edge[edge[0]][edge[1]]['unsat_info']:
                if unsat[0] != random_nodes_sig:
                    unsat_to_satisfy.append(unsat)

```

```

# The unsat to satisfy might occur more than once...if so, I must find a node which can accomodate this
unsat_to_satisfy_set = set(unsat_to_satisfy)

for unsat in unsat_to_satisfy_set:
    unsat_to_satisfy_dict[unsat] = unsat_to_satisfy.count(unsat)

list_of_unsat_potential_lists = []

for unsat in unsat_to_satisfy_dict.keys():
    unsat_potentials = []

    for neighbor in self.network.neighbors(unsat[0]):
        if neighbor != random_nodes_sig:
            if unsat in self.network.edge[unsat[0]][neighbor]['map_dict'].keys():
                if len(self.network.edge[unsat[0]][neighbor]['map_dict']) >= unsat_to_satisfy_dict[unsat]:
                    unsat_potentials.append(neighbor)

    list_of_unsat_potential_lists.append(unsat_potentials)

if len(list_of_unsat_potential_lists) > 1:
    optimal_nodes_original = []

    for x in list_of_unsat_potential_lists[0]:
        present = 'no'

        for y in range(len(list_of_unsat_potential_lists)):
            if x in list_of_unsat_potential_lists[y]:
                present = 'yes'

            else:
                present = 'no'

                break

        if present == 'yes':
            optimal_nodes_original.append(x)

elif len(list_of_unsat_potential_lists) == 1:
    optimal_nodes_original = set(list_of_unsat_potential_lists[0])

elif len(list_of_unsat_potential_lists) == 0:
    continue

optimal_nodes = [ x for x in optimal_nodes_original if self.network.node[x]['atom_count'] <
self.network.node[random_nodes_sig]['atom_count'] ]

move_list_tuples = [ x for x in original_graph.graph['move_list'] if x[0] == random_node
or x[1] == random_node]

while len(optimal_nodes) > 0 and graph_added == 'no':
    new_move_list = [ x for x in original_graph.graph['move_list'] if x not in move_list_tuples ]
    graph = original_graph.copy()

```

```

chosen_replacement_node = optimal_nodes.pop()
for edge in move_list_tuples:
    if edge[0] == random_node:
        # This means the dictionary list is keyed by the other node
        other_node = edge[1]
        other_sig = graph.node[other_node]['sig']
        other_unsats = [ x[1] for x in graph.edge[edge[0]][edge[1]]['unsat_info'] if x[0] == other_sig ]
        map_found = 'no'
        # Might need to keep track of other unsats utilized so that they're only hit once
        for other_unsat in other_unsats:
            if self.network.has_edge(chosen_replacement_node, other_sig):
                for map in self.network.edge[chosen_replacement_node][other_sig]['map_dict']
                    [(other_sig, other_unsat)]:
                    map_items_1 = [ x[1] for x in map.items() ]
                    missing_other_unsats = [ x for x in other_unsats if x not in map_items_1 ]
                    if len(missing_other_unsats) == 0:
                        map_found = 'yes'
                        graph.edge[edge[0]][edge[1]]['unsat_info'][:] = []
                        new_randy_unsats = [x for x in map.keys() if x in self.network.node
                            [chosen_replacement_node]['unsaturated_nodes']]
                        for randy_unsat in new_randy_unsats:
                            graph.edge[edge[0]][edge[1]]['unsat_info'].append(
                                ( chosen_replacement_node , randy_unsat ) )
                        break
                    if map_found == 'yes':
                        break
            if map_found == 'yes':
                for other_unsat in other_unsats:
                    graph.edge[edge[0]][edge[1]]['unsat_info'].append( (other_sig, other_unsat) )
                to_remove = [ x for x in graph.graph['move_list'] if x[0] == edge[0] and
                    x[1] == edge[1] ][0]
                new_move_list.append( ( random_node, other_node, map.copy() ) )
            elif map_found == 'no':
                print 'Map not found'
                break
    else:
        # This means the dictionary list is keyed by the random node

```

```

other_node = edge[0]
other_sig = graph.node[other_node]['sig']
other_unsats = [ x[1] for x in graph.edge[edge[0]][edge[1]]['unsat_info'] if x[0] == other_sig ]
map_found = 'no'
for random_unsat in self.network.node[chosen_replacement_node]['unsaturated_nodes']:
    if self.network.has_edge(chosen_replacement_node,other_sig):
        for map in self.network.edge[chosen_replacement_node][other_sig]['map_dict']
            [(chosen_replacement_node,random_unsat)]:
            uncovered_other_unsats = [ x for x in other_unsats if x not in map.keys() ]
            if len(uncovered_other_unsats) == 0:
                map_found = 'yes'
                break
            if map_found == 'yes':
                break
if map_found == 'yes':
    graph.edge[edge[0]][edge[1]]['unsat_info'][:] = []
    for other_unsat in other_unsats:
        graph.edge[edge[0]][edge[1]]['unsat_info'].append( (other_sig,other_unsat ) )
    random_unsats = [ x for x in map.keys() if x in self.network.node[
        chosen_replacement_nodes['unsaturated_nodes'] ]
    for randy_unsat in random_unsats:
        graph.edge[edge[0]][edge[1]]['unsat_info'].append( (chosen_replacement_node,
            randy_unsat) )
    to_remove = [ x for x in graph.graph['move_list'] if x[1] == random_node and x[0] ==
        other_node ][0]
    new_move_list.append( ( other_node, random_node, map.copy() ) )
else:
    print 'Map not found'
    break
if map_found == 'yes':
    graph.graph['move_list'] = new_move_list
    mapping_condition = 'yes'
    zero_list = [ x[0] for x in graph.graph['move_list'] ]
    one_list = [ x[1] for x in graph.graph['move_list'] ]
    zero_set = set ( zero_list )
    for each in zero_set:
        if each in one_list:

```

```

        if one_list.index(each) < zero_list.index(each):
            shuffle(graph.graph['move_list'])
            mapping_condition = 'no'
            break
    while mapping_condition == 'no':
        mapping_condition = 'yes'
        zero_list = [ x[0] for x in graph.graph['move_list'] ]
        one_list = [ x[1] for x in graph.graph['move_list'] ]
        zero_set = set ( zero_list )
        for each in zero_set:
            if each in one_list:
                if one_list.index(each) < zero_list.index(each):
                    shuffle(graph.graph['move_list'])
                    mapping_condition = 'no'
                    break
    graph_added = 'yes'
    graph.graph['conformers'][:] = []
    graph.node[random_node]['sig'] = chosen_replacement_node
    move_status = 'good'
    for move in graph.graph['move_list']:
        if move[0] not in graph.nodes() or move[1] not in graph.nodes():
            print 'Uh oh there was a move without a bond for a deletion mutation'
            move_status = 'bad'
            break
    if move_status == 'good':
        print 'New graph added to city by deletion mutation.'
        new_city.append(graph)
        new_pop += 1
        deletion_mutation_count += 1
    else:
        continue

# Insertion Mutation
elif operation[0] == 'mutation' and operation[1] == 'insertion':
    original_graph = operation[2]
    nodes = original_graph.nodes()[:]
```

```

found_node = 'no'
graph_added = 'no'
while len(nodes) > 0 and graph_added == 'no':
    shuffle(nodes)
    random_node = nodes.pop()
    random_nodes_sig = original_graph.node[random_node]['sig']
    edges_from_random_node = nx.edges(original_graph,random_node)
    unsat_to_satisfy = []
    unsat_to_satisfy_dict = defaultdict(int)
    for edge in edges_from_random_node:
        for unsat in original_graph.edge[edge[0]][edge[1]]['unsat_info']:
            if unsat[0] != random_nodes_sig:
                unsat_to_satisfy.append(unsat)
# The unsat to satisfy might occur more than once...if so, I must find a node which can accomodate this
    unsat_to_satisfy_set = set(unsat_to_satisfy)
    for unsat in unsat_to_satisfy_set:
        unsat_to_satisfy_dict[unsat] = unsat_to_satisfy.count(unsat)
    list_of_unsat_potential_lists = []
    for unsat in unsat_to_satisfy_dict.keys():
        unsat_potentials = []
        for neighbor in self.network.neighbors(unsat[0]):
            if neighbor != random_nodes_sig:
                if unsat in self.network.edge[unsat[0]][neighbor]['map_dict'].keys():
                    if len(self.network.edge[unsat[0]][neighbor]['map_dict']) >= unsat_to_satisfy_dict[unsat]:
                        unsat_potentials.append(neighbor)
        list_of_unsat_potential_lists.append(unsat_potentials)
    if len(list_of_unsat_potential_lists) > 1:
        optimal_nodes_original = []
        for x in list_of_unsat_potential_lists[0]:
            present = 'no'
            for y in range(len(list_of_unsat_potential_lists)):
                if x in list_of_unsat_potential_lists[y]:
                    present = 'yes'
            else:
                present = 'no'

```



```

        break
    if present == 'yes':
        optimal_nodes_original.append(x)
elif len(list_of_unsat_potential_lists) == 1:
    optimal_nodes_original = set(list_of_unsat_potential_lists[0])
elif len(list_of_unsat_potential_lists) == 0:
    continue
optimal_nodes = [ x for x in optimal_nodes_original if self.network.node[x]['atom_count'] >
                 self.network.node[random_nodes_sig]['atom_count'] ]
move_list_tuples = [ x for x in original_graph.graph['move_list'] if x[0] == random_node
                    or x[1] == random_node ]
while len(optimal_nodes) > 0 and graph_added == 'no':
    new_move_list = [ x for x in original_graph.graph['move_list'] if x not in move_list_tuples ]
    graph = original_graph.copy()
    chosen_replacement_node = optimal_nodes.pop()
# Depending on the first two entries of the map i need to decide if the map is keyed or entried by the new node
for edge in move_list_tuples:
    if edge[0] == random_node:
        # This means the dictionary list is keyed by the other node
        other_node = edge[1]
        other_sig = graph.node[other_node]['sig']
        other_unsats = [ x[1] for x in graph.edge[edge[0]][edge[1]]['unsat_info'] if x[0] == other_sig ]
        map_found = 'no'
        # Might need to keep track of other unsats utilized so that they're only hit once
        for other_unsat in other_unsats:
            for map in self.network.edge[chosen_replacement_node][other_sig]['map_dict']
                [(other_sig, other_unsat)]:
                map_items_1 = [ x[1] for x in map.items() ]
                missing_other_unsats = [ x for x in other_unsats if x not in map_items_1 ]
                if len(missing_other_unsats) == 0:
                    map_found = 'yes'
                    graph.edge[edge[0]][edge[1]]['unsat_info'][:] = []
                    new_randy_unsats = [ x for x in map.keys() if x in self.network.node
                                        [chosen_replacement_node]['unsaturated_nodes']]
                    for randy_unsat in new_randy_unsats:
                        graph.edge[edge[0]][edge[1]]['unsat_info'].append(
                            ( chosen_replacement_node , randy_unsat ))

```

```

        break

    if map_found == 'yes':
        break

    if map_found == 'yes':
        for other_unsat in other_unsats:
            graph.edge[edge[0]][edge[1]]['unsat_info'].append( (other_sig,other_unsat) )

        to_remove = [ x for x in graph.graph['move_list'] if x[0] == edge[0] and
            x[1] == edge[1] ][0]

        new_move_list.append( ( random_node, other_node, map.copy() ) )

    elif map_found == 'no':
        print 'Map not found'
        break

else:
    # This means the dictionary list is keyed by the random node
    other_node = edge[0]
    other_sig = graph.node[other_node]['sig']
    other_unsats = [ x[1] for x in graph.edge[edge[0]][edge[1]]['unsat_info'] if x[0] == other_sig ]
    # Now I'll search through all map keys and maps to search for one which covers all unsats
    map_found = 'no'
    for random_unsat in self.network.node[chosen_replacement_node]['unsaturated_nodes']:
        if self.network.has_edge(chosen_replacement_node,other_sig):
            for map in self.network.edge[chosen_replacement_node][other_sig]['map_dict']
                [(chosen_replacement_node,random_unsat)]:
                uncovered_other_unsats = [ x for x in other_unsats if x not in map.keys() ]
                if len(uncovered_other_unsats) == 0:
                    map_found = 'yes'
                    break

            if map_found == 'yes':
                break

    if map_found == 'yes':
        graph.edge[edge[0]][edge[1]]['unsat_info'][:] = []
        for other_unsat in other_unsats:
            graph.edge[edge[0]][edge[1]]['unsat_info'].append( (other_sig,other_unsat) )
        random_unsats = [x for x in map.keys() if x in self.network.node
            [chosen_replacement_node]['unsaturated_nodes'] ]
        for randy_unsat in random_unsats:
            graph.edge[edge[0]][edge[1]]['unsat_info'].append(

```

```

        (chosen_replacement_node, randy_unsat) )

    to_remove = [ x for x in graph.graph['move_list'] if x[1] == random_node and x[0] ==
                  other_node ][0]

    new_move_list.append( ( other_node, random_node, map.copy() ) )

if map_found == 'yes':
    graph.graph['move_list'] = new_move_list
    mapping_condition = 'yes'
    zero_list = [ x[0] for x in graph.graph['move_list'] ]
    one_list = [ x[1] for x in graph.graph['move_list'] ]
    zero_set = set ( zero_list )
    for each in zero_set:
        if each in one_list:
            if one_list.index(each) < zero_list.index(each):
                shuffle(graph.graph['move_list'])
                mapping_condition = 'no'
                break
    while mapping_condition == 'no':
        mapping_condition = 'yes'
        zero_list = [ x[0] for x in graph.graph['move_list'] ]
        one_list = [ x[1] for x in graph.graph['move_list'] ]
        zero_set = set ( zero_list )
        for each in zero_set:
            if each in one_list:
                if one_list.index(each) < zero_list.index(each):
                    shuffle(graph.graph['move_list'])
                    mapping_condition = 'no'
                    break
    graph_added = 'yes'
    graph.graph['conformers'][:] = []
    graph.node[random_node]['sig'] = chosen_replacement_node
    move_status = 'good'
    for move in graph.graph['move_list']:
        if move[0] not in graph.nodes() or move[1] not in graph.nodes():
            print 'Uh oh a move was found without a bond for an insertion mutaiton'
            move_status = 'bad'
            break

```

```

        if move_status == 'good':
            print 'New graph added to city by insertion mutation.'
            new_city.append(graph)
            new_pop += 1
            insertion_mutation_count += 1
        else:
            print 'This one was not connected and was created by insertion mutation!'
    else:
        continue

# Crossover Mutation

elif operation[0] == 'crossover':
    graph_1 = operation[1]
    graph_1.remove_edge(operation[2][0], operation[2][1])
    two_graphs1 = nx.connected_components(graph_1)
    list_to_remove1 = [ x for x in two_graphs1 if operation[3] not in x ][0]
    print 'Graph 1 before:', graph_1.nodes(), graph_1.edges()
    for node in list_to_remove1:
        graph_1.remove_node(node)
        for move in graph_1.graph['move_list']:
            if node == move[0] or node == move[1]:
                graph_1.graph['move_list'].remove(move)
    print 'Graph 1 after:', graph_1.nodes(), graph_1.edges()
    graph_2 = operation[4]
    print 'Graph 2 before:', graph_2.nodes(), graph_2.edges()
    graph_2.remove_edge(operation[5][0], operation[5][1])
    two_graphs2 = nx.connected_components(graph_2)
    list_to_remove2 = [ x for x in two_graphs2 if operation[6] not in x ][0]
    for node in list_to_remove2:
        graph_2.remove_node(node)
        for move in graph_2.graph['move_list']:
            if node == move[0] or node == move[1]:
                graph_2.graph['move_list'].remove(move)
    print 'Graph 2 after:', graph_2.nodes(), graph_2.edges()

# Now I have these two graph with corrected move lists but I need to combine them and potentially change node labels

```

```

# Along with these node label changes I will also need to change the respective move list
max_in_1 = max( [x for x in graph_1.nodes() ] )

node_change_dict = defaultdict(int)

node_change_dict.clear()

new_label = max_in_1 + 1

for node in graph_2.nodes():

    node_change_dict[node] = new_label

    new_label += 1

# Now I have a dictionary with new labels

# I want to start by making the link between the two adjoined graphs

node_in_1 = operation[3]

sig_in_1 = graph_1.node[node_in_1]['sig']

node_in_2 = operation[6]

sig_in_2 = graph_2.node[node_in_2]['sig']

map = operation[7]

graph_1.add_node(node_change_dict[node_in_2])

graph_1.node[ node_change_dict[node_in_2] ]['sig'] = sig_in_2

graph_1.add_edge(node_change_dict[node_in_2],node_in_1)

node_in_1_nodes = [ x for x in self.network.node[sig_in_1]['conformers'][0].nodes() ]

# Now I need to add all of the unsaturated nodes associated with the new map

# Sig in one should be the keys

unsat_for_sig_1 = [ (node_in_1,x) for x in self.network.node[sig_in_1]['unsaturated_nodes'] if x in map.keys() ]

map_ones = [ x[1] for x in map.items() ]

unsat_for_sig_2 = [ (node_change_dict[node_in_2],x) for x in self.network.node[sig_in_2]['unsaturated_nodes'] if
                    x in map_ones ]

total_new_unsat = []

for each in unsat_for_sig_1:

    total_new_unsat.append(each)

for each in unsat_for_sig_2:

    total_new_unsat.append(each)

graph_1.edge[node_change_dict[node_in_2]][node_in_1]['unsat_info'] = total_new_unsat[:]

for move in graph_2.graph['move_list']:

    node_1 = node_change_dict[move[0]]

    node_2 = node_change_dict[move[1]]

    graph_1.add_node(node_1)

    graph_1.node[node_1]['sig'] = graph_2.node[move[0]]['sig']

```

```

graph_1.add_node(node_2)
graph_1.node[node_2]['sig'] = graph_2.node[move[1]]['sig']
graph_1.add_edge(node_1,node_2)
graph_1.edge[node_1][node_2]['unsat_info'] = graph_2.edge[move[0]][move[1]]['unsat_info'][:]
new_move = ( node_change_dict[move[0]], node_change_dict[move[1]], move[2] )
graph_1.graph['move_list'].append(new_move)

# This piece will determine if there is a move that doesn't correspond to a node in the graph
map_status = 'continue'
for move in graph_1.graph['move_list']:
    if move[0] not in graph_1.nodes() or move[1] not in graph_1.nodes():
        print 'Uh oh a move didnt have a edge for a crossover graph!'
        map_status = 'break'
        break
if map_status != 'break':
    iterations = 0
    mapping_condition = 'yes'
    zero_list = [ x[0] for x in graph_1.graph['move_list'] ]
    one_list = [ x[1] for x in graph_1.graph['move_list'] ]
    zero_set = set ( zero_list )
    for each in zero_set:
        if each in one_list:
            if one_list.index(each) < zero_list.index(each):
                shuffle(graph_1.graph['move_list'])
                mapping_condition = 'no'
                break
    while mapping_condition == 'no':
        iterations += 1
        mapping_condition = 'yes'
        zero_list = [ x[0] for x in graph_1.graph['move_list'] ]
        one_list = [ x[1] for x in graph_1.graph['move_list'] ]
        zero_set = set ( zero_list )
        for each in zero_set:
            if each in one_list:
                if one_list.index(each) < zero_list.index(each):
                    shuffle(graph_1.graph['move_list'])

```

```

        mapping_condition = 'no'
        break

    print 'New graph added to city by crossover.'
    graph_1.graph['conformers'][:] = []
    new_city.append(graph_1)
    new_pop += 1
    crossover_count += 1

list_of_conformer_indices = []
for node in graph.nodes():
    sig = graph.node[node]['sig']
    # I will have to change this later when I'm looking at conformational ensembles
    graph.node[node]['sig_graph'] = self.network.node[sig]['conformers'][0].copy()
    conf_count = len(self.network.node[sig]['conformers'])
    index_list = [x for x in range(conf_count)]
    list_of_conformer_indices.append(index_list)

# I'm throwing this in for later when I want to look at a conformational ensemble
subgraph_permutations = list(itertools.product(*list_of_conformer_indices))
for permutation in subgraph_permutations:
    permutation = subgraph_permutations[0]
    node_with_conformer_tuples = []
    for node in graph.nodes():
        node_index = graph.nodes().index(node)
        node_with_conformer_tuples.append((node, permutation[node_index]))
    changed_nodes = []
    xi = 0
    # I need to rearrange the move list
    for move in graph.graph['move_list']:
        node1 = move[1]
        node2 = move[0]
        map = move[2]
        sig2_index = 0
        sig2 = graph.node[node2]['sig']
        sig2_graph = self.network.node[sig2]['conformers'][sig2_index].copy()
        # This is basically saying that if this is my first iteration then I need to begin the graph
        if graph.graph['move_list'].index(move) == 0:

```

```

sig1 = graph.node[node1]['sig']
sig1_graph = self.network.node[sig1]['conformers'][0]
conformer = sig1_graph.copy()
else:
    sig1_graph = graph.node[node1]['sig_graph']
node_2a = choice( map.keys() )
node_1a = [ x[1] for x in map.items() if x[0] == node_2a ][0]
node_2a_coordinates = sig2_graph.node[node_2a]['xyz']
node_1a_coordinates = sig1_graph.node[node_1a]['xyz']
translation_vector = numpy.subtract(node_1a_coordinates,node_2a_coordinates)
translation_matrix = numpy.array([[1,0,0,translation_vector[0]],[0,1,0,translation_vector[1]],
                                  [0,0,1,translation_vector[2]],[0,0,0,1]])
# Now I need to translate all nodes in the second signature copy
for node in sig2_graph.nodes():
    new_coordinates = numpy.dot( translation_matrix , sig2_graph.node[node]['xyz'] )
    new_coordinates_t = numpy.transpose(new_coordinates)
    sig2_graph.node[node]['xyz'] = new_coordinates_t
# Now I need to choose a second mapped node to create a rotation
node2b_choices = [x for x in map.keys() if not x ==node_2a]
node_2b = choice(node2b_choices)
node_1b = map[node_2b]
node_1b_coordinates = sig1_graph.node[node_1b]['xyz']
node_2b_coordinates = sig2_graph.node[node_2b]['xyz']
nodes_1ab_vector = numpy.subtract(node_1b_coordinates[:3],node_1a_coordinates[:3])
nodes_2ab_vector = numpy.subtract(node_2b_coordinates[:3],node_2a_coordinates[:3])
angle_1 = angle_between(numpy.transpose(nodes_1ab_vector),nodes_2ab_vector)
axis_1 = numpy.cross(nodes_1ab_vector,nodes_2ab_vector)
# I still need to normalize this axis
if numpy.linalg.norm(axis_1) == 0:
    axis_1u = axis_1
else:
    axis_1u = axis_1/numpy.linalg.norm(axis_1)
x = axis_1u[0]
y = axis_1u[1]
z = axis_1u[2]

```



```

a = angle_1
rotation_matrix_1 = numpy.array([[cos(a)+(x*x)*(1-cos(a)), x*y*(1-cos(a))-z*sin(a), x*z*(1-cos(a))+y*sin(a)],
                                [y*x*(1-cos(a))+z*sin(a), cos(a)+(y*y)*(1-cos(a)), y*z*(1-cos(a))-x*sin(a)],
                                [z*x*(1-cos(a))-y*sin(a), z*y*(1-cos(a))+x*sin(a), cos(a)+(z*z)*(1-cos(a))]])

for node in sig2_graph.nodes():

    new_coordinates = numpy.dot(rotation_matrix_1,numpy.transpose(sig2_graph.node[node]['xyz'][:3]))

    new_coordinates_t = numpy.transpose(new_coordinates)

    new_coordinates_t_a = numpy.append(new_coordinates_t,1)

    sig2_graph.node[node]['xyz'] = new_coordinates_t_a

# Now I need to do the final rotation calculation where I try various angles until the difference in mapped atom
# coordinates is a minimum

# First, I'll have to define the new axis of rotation

# This will be the difference vector between points a1 and a2 or b1 and b2
axis_2u = nodes_2ab_vector/numpy.linalg.norm(nodes_2ab_vector)

x = axis_2u[0]
y = axis_2u[1]
z = axis_2u[2]

rotation_list = []

for d in range(0,360,2):

    a = numpy.radians(d)

    rotation_matrix_2 = numpy.array([[cos(a)+(x*x)*(1-cos(a)), x*y*(1-cos(a))-z*sin(a), x*z*(1-
cos(a))+y*sin(a)], [y*x*(1-cos(a))+z*sin(a), cos(a)+(y*y)*(1-cos(a)), y*z*(1-cos(a))-x*sin(a)],
[z*x*(1-cos(a))-y*sin(a), z*y*(1-cos(a))+x*sin(a), cos(a)+(z*z)*(1-cos(a))]])

    temp_node_dictionary = defaultdict(numpy.array)

    # Now im going to create a list of all mapped nodes in the second signature

    node_list = map.keys()

    for node in node_list:

        new_coordinates = numpy.transpose(numpy.dot(rotation_matrix_2,numpy.transpose(
sig2_graph.node[node]['xyz'][:3])))

        temp_node_dictionary[node] = new_coordinates

    total_distance = 0

    for node in node_list:

        sig_1_node = map[node]

        total_distance += numpy.linalg.norm(temp_node_dictionary[node]-sig1_graph.node
[sig_1_node]['xyz'][:3])

    temp_node_dictionary.clear()

    rotation_list.append((d,a,total_distance))

# Now I must identify the lowest total distance

```

```

best_radian = min(rotation_list, key=itemgetter(2))[1]

a = best_radian

# Now I need to actually rotate all of the atoms in the second signature
rotation_matrix_3 = numpy.array([[cos(a)+(x*x)*(1-cos(a)), x*y*(1-cos(a))-z*sin(a), x*z*(1-cos(a))+y*sin(a)],
                                  [y*x*(1-cos(a))+z*sin(a), cos(a)+(y*y)*(1-cos(a)), y*z*(1-cos(a))-x*sin(a)],
                                  [z*x*(1-cos(a))-y*sin(a), z*y*(1-cos(a))+x*sin(a), cos(a)+(z*z)*(1-cos(a))]])

for node in sig2_graph.nodes():

    new_coordinates = numpy.transpose(numpy.dot(rotation_matrix_3,numpy.transpose(
        sig2_graph.node[node]['xyz'][:3])))

    new_coordinates_a = numpy.append(new_coordinates,1)

    sig2_graph.node[node]['xyz'] = new_coordinates_a

graph.node[node2]['sig_graph'] = sig2_graph.copy()

# Now I need to add to the growing graph

# First, I'll add the nodes not involved in docking
nodes_to_add = [x for x in sig2_graph.nodes() if x not in map.keys()]

for node in nodes_to_add:

    # Adding in label change because of conflicting fragments with the same labels
    new_node_id = 100*(xi+1) + node

    conformer.add_node(new_node_id, atom= sig2_graph.node[node]['atom'],
                      xyz = sig2_graph.node[node]['xyz'], old_id = node)

# Next, I'll need to add all edges located in signature two which don't involve docking nodes
edges_to_add = [edge for edge in sig2_graph.edges() if edge[0] not in map.keys() and edge[1] not in map.keys()]

for edge in edges_to_add:

    new_edge_0 = edge[0] + 100*(xi+1)

    new_edge_1 = edge[1] + 100*(xi+1)

    conformer.add_edge(new_edge_1, new_edge_0, type = sig2_graph.edge[edge[0]][edge[1]]['type'])

# Now I need to add edges within the docking zone involving one node from signature two and one from signature one
next_edges_to_add = []

for edge in sig2_graph.edges():

    if ( edge[0] not in map.keys() and edge[1] in map.keys() ):

        if xi == 0:

            node_in_1 = map[ edge[1] ]

        else:

            node_in_1 = map[ edge[1] ] + (100*xi)

        new_edge_0 = 100*(xi+1) + edge[0]

        next_edges_to_add.append( ( new_edge_0 , node_in_1 , sig2_graph.edge[

```

```

edge[0]][edge[1]]['type'])
elif ( edge[0] in map.keys() and edge[1] not in map.keys() ):
    if xi == 0:
        node_in_1 = map [ edge[0] ]
    else:
        node_in_1 = map[ edge[0] ] + (100*xi)
        new_edge_1 = 100*(xi+1) + edge[1]
        next_edges_to_add.append ( ( node_in_1 , new_edge_1 , sig2_graph.edge[
            edge[0]][edge[1]]['type'] ) )
    else:
        pass
for edge in next_edges_to_add:
    conformer.add_edge(edge[0],edge[1],type = edge[2])
xi += 1
graph.graph['conformers'].append(conformer)
print 'Conformer created!'

```

A.5 – Code for Expedited Molecular Mechanics Analysis

The following piece of code was created to handle large data sets when performing conformational analysis using the BOSS program. The code scans the conformational space of each mol file in a directory and creates a file with each identified conformational isomer while cleaning up the remaining log files.

```

import subprocess
import glob
import linecache
from decimal import *
class Directory:
    def __init__(self,full_parent_directory):
        self.full_parent_directory = full_parent_directory
    def conformational_analysis(self,cutoff):
        """This function canonizes the mol files in directory and stores hydrogen suppressed signatures in self.signatures dictionary"""
        cutoff_decimal = Decimal(cutoff)
        if self.full_parent_directory[-1] == '/':

```

```

self.full_parent_directory = self.full_parent_directory[:-len(self.full_parent_directory)-1]
mol_files = self.full_parent_directory + '/*.mol'
file_iter = glob.iglob(mol_files)
subprocess.call('mkdir conformers',shell=True,cwd=self.full_parent_directory)
import_cs_command = 'cp /opt/asn/apps/boss_4.6/BOSS/scripts/xCS100 '+ self.full_parent_directory + '/' + 'xCS100'
import_xmolz_command = 'cp /opt/asn/apps/boss_4.6/BOSS/scripts/xMOLZ '+ self.full_parent_directory + '/' + 'xMOLZ'
import_xcsmol_command = 'cp /opt/asn/apps/boss_4.6/BOSS/scripts/xCSMOL'+self.full_parent_directory + '/' + 'xCSMOL'
subprocess.call(import_xmolz_command,shell=True)
subprocess.call(import_cs_command,shell=True)
subprocess.call(import_xcsmol_command,shell=True)
for fn in file_iter:
    mol_file = fn.replace(self.full_parent_directory,"")
    mol_file = mol_file.replace('/', "")
    mol_file = mol_file.replace('.mol',"")
    xmolz_command = 'xMOLZ ' + mol_file
    xmolz_subprocess = subprocess.Popen(xmolz_command,shell=True,cwd=self.full_parent_directory)
    xmolz_subprocess.wait()
    rm_log = 'rm' + ' log'
    rm_optzmat = 'rm' + ' optzmat'
    rm_out = 'rm' + ' out'
    rm_plt = 'rm' + ' plt.pdb'
    rm_sum = 'rm' + ' sum'
    subprocess.Popen(rm_log,shell=True,cwd=self.full_parent_directory)
    subprocess.Popen(rm_optzmat,shell=True,cwd=self.full_parent_directory)
    subprocess.Popen(rm_out,shell=True,cwd=self.full_parent_directory)
    subprocess.Popen(rm_plt,shell=True,cwd=self.full_parent_directory)
    subprocess.Popen(rm_sum,shell=True,cwd=self.full_parent_directory)
    xcs_command = 'xCS100 ' + mol_file
    xcs_subprocess = subprocess.Popen(xcs_command,shell=True,cwd=self.full_parent_directory)
    xcs_subprocess.wait()
    rm_z = 'rm' + mol_file + '.z'
    rm_cs = 'rm' + mol_file + '.cs.CSV'
    rm_cs_sum = 'rm' + mol_file + '.cs.sum'
    rm_csz = 'rm' + mol_file + '.cs.z'
    rm_cs_out = 'rm' + mol_file + '.cs.out'

```

```

subprocess.Popen(rm_z,shell=True,cwd=self.full_parent_directory)
subprocess.Popen(rm_cs,shell=True,cwd=self.full_parent_directory)
subprocess.Popen(rm_cs_sum,shell=True,cwd=self.full_parent_directory)
subprocess.Popen(rm_csz,shell=True,cwd=self.full_parent_directory)
subprocess.Popen(rm_cs_out,shell=True,cwd=self.full_parent_directory)
subprocess.Popen('rm log',shell=True,cwd=self.full_parent_directory)
xcsmol_subprocess = subprocess.Popen('xcSMOL',cwd=self.full_parent_directory,stdin=subprocess.PIPE)
input_text = mol_file + '.cs.mol'
xcsmol_subprocess.communicate(input = input_text)[0]
xcsmol_subprocess.wait()
rm_csmol = 'rm ' + mol_file + '.cs.mol'
subprocess.call(rm_csmol,shell=True,cwd=self.full_parent_directory)
first_conformer = self.full_parent_directory + '/cs001.mol'
lowest_energy = Decimal((linecache.getline(first_conformer,3)[43:51]).strip())
print 'Lowest energy:', lowest_energy
mv_best_conformer = 'mv ' + '/cs001.mol ' + '/conformers/' + mol_file + '.1.mol'
cs_files = self.full_parent_directory + '/cs*.mol'
file_iter_2 = glob.iglob(cs_files)
subprocess.call(mv_best_conformer,shell=True,cwd=self.full_parent_directory)
n = 2
for fn in file_iter_2:
    energy = Decimal((linecache.getline(fn,3)[45:51]).strip())
    if energy < lowest_energy + cutoff_decimal:
        mv_conformer_cmd = 'mv ' + fn + ' ' + self.full_parent_directory + '/conformers/' + mol_file + '!' + str(n) + '.mol'
        subprocess.call(mv_conformer_cmd,shell=True)
    else:
        rm_conformer_cmd = 'rm ' + fn
        subprocess.call(rm_conformer_cmd,shell=True)
n += 1
lowest_energy = 0

```

A.6 – Geometry Verification Code

The following code was written to verify the geometry generation process for a given data set. It is formatted such that it can be used for sensitivity analysis when choosing the

optimal cutoff value for network compression such that significant conformational isomers are still created during the search.

```
import networkx as nx
import networkx.algorithms.isomorphism as iso
import math
from math import fabs
import itertools
import re
import glob
import subprocess
import linecache
from collections import defaultdict
from operator import eq
from pprint import pprint
import numpy
from operator import itemgetter
from random import choice
sin = numpy.sin
cos = numpy.cos
valence_dictionary = {'C':4,'O':2,'H':1,'N':3,'S':2,'Cl':1,'F':1,'Br':1}
bond_dictionary = {1:1,2:2,4:1.5}
def unit_vector(vector):
    return vector/numpy.linalg.norm(vector)
def angle_between(v1, v2):
    v1_u = unit_vector(v1)
    v2_u = unit_vector(v2)
    angle = numpy.arccos(numpy.dot(v1_u,v2_u))
    if numpy.isnan(angle):
        if (v1_u == v2_u).all():
            return 0.0
        else:
            return numpy.pi
    return angle
class Directory:
```

```

def __init__(self,full_parent_directory):
    self.full_parent_directory = full_parent_directory

def create_graphs(self,height):
    """This function will create nx graphs with underlying signatures stored in each node"""
    self.height = height
    self.graphs = []
    if self.full_parent_directory[-1] == '/':
        self.full_parent_directory = self.full_parent_directory[:len(self.full_parent_directory)-1]
    mol_files = self.full_parent_directory + '/*.mol'
    file_iter = glob.iglob(mol_files)
    for fn in file_iter:
        # Creating a nx.Graph from the mol file
        G = nx.Graph()
        G.graph['fn'] = fn
        if 'OpenBabel' in str(linecache.getline(fn,2)):
            atom_bond_number_line = 4
        else:
            atom_bond_number_line = 5
        atomn = int((linecache.getline(fn,atom_bond_number_line)[1:4]).strip())
        bondn = int((linecache.getline(fn,atom_bond_number_line)[4:8]).strip())
        coordinate_begin_line = atom_bond_number_line + 1
        coordinate_end_line = atom_bond_number_line + atomn
        bond_begin_line = coordinate_end_line + 1
        bond_end_line = coordinate_end_line + bondn
        for i in range(bond_begin_line, bond_end_line + 1):
            atom1 = int((linecache.getline(fn,i)[0:4]).strip())
            atom2 = int((linecache.getline(fn,i)[4:8]).strip())
            bondtype = int((linecache.getline(fn,i)[8:10]).strip())
            G.add_edge(atom1, atom2, type=bondtype)
        for i in range(coordinate_begin_line,coordinate_end_line + 1):
            j = i - (atom_bond_number_line)
            G.node[j]['xyz'] = numpy.array([float((linecache.getline(fn,i)[4:11]).strip()),
                                           float((linecache.getline(fn,i)[14:21]).strip()),float((linecache.getline(fn,i)[24:31]).strip())])
            G.node[j]['atom'] = ((linecache.getline(fn,i)[31:33]).strip())
    # Now assigning subgraphs to each node

```

```

nodes = G.nodes_iter()
for node in nodes:
    subgraph=nx.ego_graph(G, node, radius=self.height)
    atom = G.node[node]['atom']
    unsaturated_nodes = []
    for node2 in subgraph.nodes():
        actual_saturation = 0
        desired_saturation = valence_dictionary[subgraph.node[node2]['atom']]
        for neighbor in subgraph.neighbors(node2):
            actual_saturation += bond_dictionary[subgraph.edge[node2][neighbor]['type']]
        if actual_saturation < desired_saturation:
            unsaturated_nodes.append(node2)
    subgraph.graph['unsaturated_nodes'] = unsaturated_nodes
    G.node[node]['atomic_signature'] = subgraph
self.graphs.append(G)
print "Graph created for: ", fn
print 'All',len(self.graphs),'graphs have been imported.'
print 'Compressing graphs...'
self.compressed_graphs = []
for graph in self.graphs:
    match_status = 0
    if len(self.compressed_graphs) > 0:
        for graph_list in self.compressed_graphs:
            nm = iso.categorical_node_match('atom', 'C')
            em = iso.categorical_edge_match('type', 4)
            GM = iso.GraphMatcher(graph,graph_list[0],node_match = nm, edge_match = em)
            if GM.is_isomorphic():
                match_status = 1
                new_graph = graph.copy()
                new_graph=nx.relabel_nodes(subgraph, GM.mapping, copy=True)
                graph_list.append(new_graph)
                break
    if match_status == 0:
        new_list = []
        new_graph = graph.copy()

```



```

        new_list.append(new_graph)

        self.compressed_graphs.append(new_list)

    else:

        new_list = []

        new_graph = graph.copy()

        new_list.append(new_graph)

        self.compressed_graphs.append(new_list)

total_graphs_after_compression = 0

for graph_list in self.compressed_graphs:

    total_graphs_after_compression += len(graph_list)

print 'Graphs compressed.'

print 'Number of unique graphs is', len(self.compressed_graphs), 'with', total_graphs_after_compression, 'graphs accounted
for.'

def identify_move_lists(self):

    def create_first_size_list(graph):

        subgraph_size_list = []

        for node in graph.nodes_iter():

            unsat_nodes = len(graph.node[node]['atomic_signature'].graph['unsaturated_nodes'])

            subgraph_size = len(graph.node[node]['atomic_signature'].nodes())

            if unsat_nodes == 0:

                return False

                break

            else:

                size_metric = subgraph_size/unsat_nodes

                subgraph_size_list.append((node,size_metric))

        return subgraph_size_list

    for graph_list in self.compressed_graphs:

        graph = graph_list[0]

        move_list = []

        required_nodes = [x for x in graph.nodes()]

        defined_nodes = []

        utilized_nodes = []

        graph.graph['move_list'] = []

        a = create_first_size_list(graph)

        if a == False:

            print "This graph is too small for this methodology."

```

```

        continue
    else:
        subgraph_size_list = a
        # Now I have developed a subgraph size list and will identify the largest size
        max_metric = max(subgraph_size_list, key=itemgetter(1))[1]
        largest_node_list = [x[0] for x in subgraph_size_list if x[1] == max_metric]
        if len(largest_node_list) > 1:
            #randomly choose one
            first_node = choice(largest_node_list)
        else:
            #choose the only one
            first_node = largest_node_list[0]
        for node in graph.node[first_node]['atomic_signature'].nodes():
            defined_nodes.append(node)
        utilized_nodes.append(first_node)
        undefined = [x for x in required_nodes if x not in defined_nodes]
        last_node_used = first_node
        while len(undefined) > 0:
            # Now I need to identify all atoms having at least three in common with the defined graph
            unused_nodes = [x for x in graph.nodes() if x not in utilized_nodes]
            overlapping_signatures = []
            for unused_node in unused_nodes:
                common_nodes = [x for x in graph.node[unused_node]['atomic_signature'].nodes() if x in defined_nodes]
                unsaturated_nodes = [x for x in graph.node[unused_node]['atomic_signature'].graph['unsaturated_nodes'] if x
                                     not in common_nodes]
                newly_defined_nodes = [x for x in graph.node[unused_node]['atomic_signature'].nodes() if x not in
                                       common_nodes]
                if len(common_nodes) >= 3:
                    if len(newly_defined_nodes) > 0:
                        overlapping_signatures.append((unused_node, len(newly_defined_nodes), len(unsaturated_nodes),
                                                       common_nodes))
            # Now I have defined all possible overlapping signatures
            # The task remains to choose one which minimizes unsaturated_nodes while having the most in common
            least_unsat_remaining = min(overlapping_signatures, key=itemgetter(2))[2]
            tuples_with_least_unsat_remaining = [x for x in overlapping_signatures if x[2] == least_unsat_remaining]
            most_newly_defined = max(tuples_with_least_unsat_remaining, key=itemgetter(1))[1]

```

```

tuples_with_most_newly_defined = [x for x in tuples_with_least_unsat_remaining if x[1] == most_newly_defined]
if len(tuples_with_most_newly_defined) > 1:
    chosen_tuple = choice(tuples_with_most_newly_defined)
    next_signature = chosen_tuple[0]
    utilized_nodes.append(next_signature)
    move_list.append((last_node_used, next_signature, chosen_tuple[3]))
    newly_defined_nodes = [x for x in graph.node[next_signature]['atomic_signature'].nodes() if x not in
                           chosen_tuple[3]]
    for x in newly_defined_nodes:
        defined_nodes.append(x)
    last_node_used = next_signature
    undefined = [x for x in required_nodes if x not in defined_nodes]
elif len(tuples_with_most_newly_defined) == 1:
    chosen_tuple = tuples_with_most_newly_defined[0]
    next_signature = chosen_tuple[0]
    utilized_nodes.append(next_signature)
    move_list.append((last_node_used, next_signature, chosen_tuple[3]))
    newly_defined_nodes = [x for x in graph.node[next_signature]['atomic_signature'].nodes() if x not in
                           chosen_tuple[3]]
    for x in newly_defined_nodes:
        defined_nodes.append(x)
    last_node_used = next_signature
    undefined = [x for x in required_nodes if x not in defined_nodes]

graph.graph['move_list'] = move_list
def create_docking_map(self, network):
    # The first step is to identify the graphs necessary for docking in the network
    self.network = network
    for graph_list in self.compressed_graphs:
        graph = graph_list[0]
        graph.graph['docking_map_list'] = []
        for move in graph.graph['move_list']:
            working_list = []
            node_1 = move[0]
            node_2 = move[1]
            signature_1 = graph.node[node_1]['atomic_signature']
            signature_2 = graph.node[node_2]['atomic_signature']

```

```

# Identifying signature 1 in network
for node in network:
    conformer = network.node[node]['conformers'][0]
    nm = iso.categorical_node_match('atom', 'C')
    em = iso.numerical_edge_match('type', 1)
    GM = iso.GraphMatcher(signature_1, conformer, node_match = nm, edge_match = em)
    if GM.is_isomorphic():
        working_list.append(node)
        mapped_node_list_1 = []
        for node_id in move[2]:
            mapped_node_id = GM.mapping[node_id]
            mapped_node_list_1.append(mapped_node_id)
        working_list.append(mapped_node_list_1)
        break

# Identifying signature 2 in network
for node in network:
    conformer = network.node[node]['conformers'][0]
    nm = iso.categorical_node_match('atom', 'C')
    em = iso.numerical_edge_match('type', 1)
    GMb = iso.GraphMatcher(signature_2, conformer, node_match = nm, edge_match = em)
    if GMb.is_isomorphic():
        working_list.insert(1, node)
        mapped_node_list_2 = []
        for node_id in move[2]:
            mapped_node_id = GMb.mapping[node_id]
            mapped_node_list_2.append(mapped_node_id)
        working_list.append(mapped_node_list_2)
        break

graph.graph['docking_map_list'].append(working_list)

def create_conformers(self):
    self.conformer_lists = []
    for graph_list in self.compressed_graphs:
        conformer_list_i = []
        graph = graph_list[0]
        number_of_docks = len(graph.graph['docking_map_list'])

```

```

# Now I need to calculate all possible permutations of subgraph conformers to utilize here
list_of_conformers = []
list_of_conformer_indices = []
conformer_count = 0
for each in graph.graph['docking_map_list']:
    if conformer_count == 0:
        list_of_conformers.append(each[0])
        list_of_conformers.append(each[1])
        conformer_count += 1
    else:
        list_of_conformers.append(each[1])
for conformer in list_of_conformers:
    index_number = len(self.network.node[conformer]['conformers'])
    index_list = [x for x in range(index_number)]
    list_of_conformer_indices.append(index_list)
subgraph_permutations = list(itertools.product(*list_of_conformer_indices))
permutation_count = 0
for permutation in subgraph_permutations:
    permutation_count += 1
print 'There were this many conformers created:', permutation_count
for permutation in subgraph_permutations:
    for xi in range(number_of_docks):
        yi = xi + 1
        if xi == 0:
            first_signature = self.network.node[list_of_conformers[xi]]['conformers'][permutation[xi]]
            conformer = first_signature.copy()
        else:
            first_signature = second_signature.copy()
# I'll keep this as the working graph
#This is where I need to decide on the first point for translation (zero index chosen as place holder)
#Later I could change this to only consider non-H atoms
node_1a = graph.graph['docking_map_list'][xi][2][0]
node_1a_coordinates = first_signature.node[node_1a]['xyz']
second_signature = self.network.node[list_of_conformers[yi]]['conformers'][permutation[yi]].copy()
node_2a = graph.graph['docking_map_list'][xi][3][0]

```

```

node_2a_coordinates = second_signature.node[node_2a]['xyz']
translation_vector = numpy.subtract(node_1a_coordinates,node_2a_coordinates)
translation_matrix = numpy.array([[1,0,0,translation_vector[0]],[0,1,0,translation_vector[1]],
                                [0,0,1,translation_vector[2]],[0,0,0,1]])

# Now I need to translate all nodes in the second signature copy
for node in second_signature.nodes():
    new_coordinates = numpy.dot(translation_matrix,second_signature.node[node]['xyz'])
    new_coordinates_t = numpy.transpose(new_coordinates)
    second_signature.node[node]['xyz'] = new_coordinates_t

# Now I need to choose a second mapped node to create a rotation
node_1b = graph.graph['docking_map_list'][xi][2][2]
node_1b_coordinates = first_signature.node[node_1b]['xyz']
node_2b = graph.graph['docking_map_list'][xi][3][2]
node_2b_coordinates = second_signature.node[node_2b]['xyz']
nodes_1ab_vector = numpy.subtract(node_1b_coordinates[:3],node_1a_coordinates[:3])
nodes_2ab_vector = numpy.subtract(node_2b_coordinates[:3],node_2a_coordinates[:3])
angle_1 = angle_between(numpy.transpose(nodes_1ab_vector),nodes_2ab_vector)
axis_1 = numpy.cross(nodes_1ab_vector,nodes_2ab_vector)

# I still need to normalize this axis
if numpy.linalg.norm(axis_1) == 0:
    axis_1u = axis_1
else:
    axis_1u = axis_1/numpy.linalg.norm(axis_1)
x = axis_1u[0]
y = axis_1u[1]
z = axis_1u[2]
a = angle_1
rotation_matrix_1 = numpy.array([[cos(a)+(x*x)*(1-cos(a)), x*y*(1-cos(a))-z*sin(a), x*z*(1-
cos(a))+y*sin(a)],[y*x*(1-cos(a))+z*sin(a), cos(a)+(y*y)*(1-cos(a)),
y*z*(1-cos(a))-x*sin(a)],[z*x*(1-cos(a))-y*sin(a), z*y*(1-cos(a))+x*sin(a),
cos(a)+(z*z)*(1-cos(a))]])

for node in second_signature.nodes():
    new_coordinates = numpy.dot(rotation_matrix_1,numpy.transpose(
second_signature.node[node]['xyz'][:3]))
    new_coordinates_t = numpy.transpose(new_coordinates)
    new_coordinates_t_a = numpy.append(new_coordinates_t,1)
    second_signature.node[node]['xyz'] = new_coordinates_t_a

# Now I need to do the final rotation calculation where I try various angles until the difference in mapped atom coordinates is a

```

minimum

```
# First, I'll have to define the new axis of rotation

# This will be the difference vector between points a1 and a2 or b1 and b2

axis_2u = nodes_2ab_vector/numpy.linalg.norm(nodes_2ab_vector)

x = axis_2u[0]

y = axis_2u[1]

z = axis_2u[2]

rotation_list = []

for d in range(0,360,2):

    a = numpy.radians(d)

    rotation_matrix_2 = numpy.array([[cos(a)+(x*x)*(1-cos(a)), x*y*(1-cos(a))-z*sin(a), x*z*(1-
    cos(a))+y*sin(a)], [y*x*(1-cos(a))+z*sin(a), cos(a)+(y*y)*(1-cos(a)),
    y*z*(1-cos(a))-x*sin(a)], [z*x*(1-cos(a))-y*sin(a), z*y*(1-cos(a))+x*sin(a),
    cos(a)+(z*z)*(1-cos(a))]])

    temp_node_dictionary = defaultdict(numpy.array)

    # Now im going to create a list of all mapped nodes in the second signature

    node_list = graph.graph['docking_map_list'][xi][3]

    for node in node_list:

        new_coordinates = numpy.transpose(numpy.dot(rotation_matrix_2,
            numpy.transpose(second_signature.node[node]['xyz'][:3])))

        temp_node_dictionary[node] = new_coordinates

    # This will allow me to identify the mapped nodes in signature one

    tuples_list = zip(graph.graph['docking_map_list'][xi][3],graph.graph['docking_map_list'][xi][2])

    total_distance = 0

    for node in node_list:

        for each_tuple in tuples_list:

            if each_tuple[0] == node:

                sig_1_node = each_tuple[1]

                break

            total_distance += numpy.linalg.norm(temp_node_dictionary[node]first_signature.node[sig_1_node]
                ['xyz'][:3])

        temp_node_dictionary.clear()

        rotation_list.append((d,a,total_distance))

# Now I must identify the lowest total distance

best_radian = min(rotation_list, key=itemgetter(2))[1]

a = best_radian

# Now I need to actually rotate all of the atoms in the second signature

rotation_matrix_3 = numpy.array([[cos(a)+(x*x)*(1-cos(a)), x*y*(1-cos(a))-z*sin(a), x*z*(1-
    cos(a))+y*sin(a)], [y*x*(1-cos(a))+z*sin(a), cos(a)+(y*y)*(1-cos(a)),
    y*z*(1-cos(a))-x*sin(a)], [z*x*(1-cos(a))-y*sin(a), z*y*(1-cos(a))+x*sin(a),
    cos(a)+(z*z)*(1-cos(a))]])
```

```

        y*z*(1-cos(a))-x*sin(a)], [z*x*(1-cos(a))-y*sin(a), z*y*(1-cos(a))+x*sin(a),
        cos(a)+(z*z)*(1-cos(a))])

for node in second_signature.nodes():

    new_coordinates = numpy.transpose(numpy.dot(rotation_matrix_3, numpy.transpose
        (second_signature.node[node]['xyz'][:3])))

    new_coordinates_a = numpy.append(new_coordinates, 1)

    second_signature.node[node]['xyz'] = new_coordinates_a

# Now I need to add to the growing graph

# First, I'll add the nodes not involved in docking

nodes_to_add = [x for x in second_signature.nodes() if x not in graph.graph['docking_map_list'][xi][3]]

for node in nodes_to_add:

    # Adding in label change because of conflicting fragments with the same labels

    new_node_id = 100*(xi+1) + node

    conformer.add_node(new_node_id, atom= second_signature.node[node]['atom'], xyz =
        second_signature.node[node]['xyz'], old_id = node)

# Next, I'll need to add all edges located in signature two which don't involve docking nodes

edges_to_add = [edge for edge in second_signature.edges() if edge[0] not in graph.graph['docking_map_list']
    [xi][3] and edge[1] not in graph.graph['docking_map_list'][xi][3]]

for edge in edges_to_add:

    new_edge_0 = edge[0] + 100*(xi+1)

    new_edge_1 = edge[1] + 100*(xi+1)

    conformer.add_edge(new_edge_1, new_edge_0, type = second_signature.edge[edge[0]][edge[1]]['type'])

# Now I need to add edges within the docking zone involving one node from signature two and one from signature one

next_edges_to_add = []

for edge in second_signature.edges():

    if edge[0] not in graph.graph['docking_map_list'][xi][3] and edge[1] in graph.graph
    ['docking_map_list'][xi][3]:

        node_in_1_index = graph.graph['docking_map_list'][xi][3].index(edge[1])

        if xi == 0:

            node_in_1 = graph.graph['docking_map_list'][xi][2][node_in_1_index]

        else:

            node_in_1 = graph.graph['docking_map_list'][xi][2][node_in_1_index] + (100*xi)

        new_edge_0 = 100*(xi+1) + edge[0]

        next_edges_to_add.append( ( new_edge_0 , node_in_1 , second_signature.edge
            [edge[0]][edge[1]]['type'] ) )

    elif edge[0] in graph.graph['docking_map_list'][xi][3] and edge[1] not in graph.graph['docking_map_list']
    [xi][3]:

        node_in_1_index = graph.graph['docking_map_list'][xi][3].index(edge[0])

        if xi == 0:

```



```

        node_in_1 = graph.graph['docking_map_list'][xi][2][node_in_1_index]
    else:
        node_in_1 = graph.graph['docking_map_list'][xi][2][node_in_1_index] + (100*xi)
        new_edge_1 = 100*(xi+1) + edge[1]
        next_edges_to_add.append((node_in_1, new_edge_1, second_signature.edge[edge[0]]
        [edge[1]]['type']))
    else:
        pass
    for edge in next_edges_to_add:
        conformer.add_edge(edge[0],edge[1],type = edge[2])
    conformer_list_i.append(conformer)
    self.conformer_lists.append(conformer_list_i)
conformer_count = 0
conformer_list_count = 0
for list_i in self.conformer_lists:
    conformer_list_count += 1
    conformer_count += len(list_i)
print 'Overall I have generated:', conformer_list_count, 'unique graphs with:', conformer_count, 'total conformers.'
def quicker_geom_test(self):
    for x in range(len(self.compressed_graphs)):
        graph_1 = self.compressed_graphs[x][1]
        graph_2 = self.compressed_graphs[x][0]
        nm = iso.categorical_node_match('atom','C')
        em = iso.categorical_edge_match('type',4)
        GM = iso.GraphMatcher(graph_1,graph_2, node_match=nm, edge_match=em)
        if GM.is_isomorphic():
            print 'Isomorphic'
        else:
            print 'Not Isomorphic'
def quick_geom_test(self,cutoff_2):
    for x in range(len(self.compressed_graphs)):
        total_graphs = len(self.compressed_graphs[x])
        print 'Total graphs', total_graphs
        matched_graphs = 0
        for graph_1 in self.compressed_graphs[x]:
            for graph_2 in self.conformer_lists[x]:

```

```

match = 'yes'

nm = iso.categorical_node_match('atom', 'C')

em = iso.categorical_edge_match('type', 4)

GM = iso.GraphMatcher(graph_1, graph_2, node_match=nm, edge_match=em)

if GM.is_isomorphic():
    for edge in graph_1.edges():
        nodeA = edge[0]
        nodeB = edge[1]
        nodeC = GM.mapping[nodeA]
        nodeD = GM.mapping[nodeB]

        d1 = numpy.linalg.norm(graph_1.node[nodeA]['xyz'][:3]-graph_1.node[nodeB]['xyz'][:3])
        d2 = numpy.linalg.norm(graph_2.node[nodeC]['xyz'][:3]-graph_2.node[nodeD]['xyz'][:3])

        if fabs(d1-d2) > float(cutoff_2):
            match = 'no'
            break

    if match == 'yes':
        matched_graphs += 1
        break

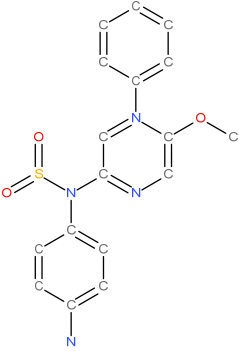
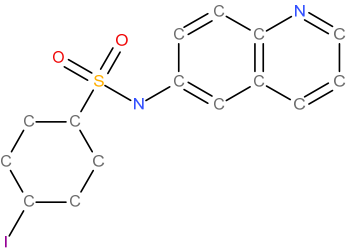
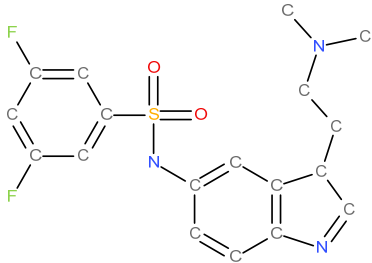
    else:
        continue

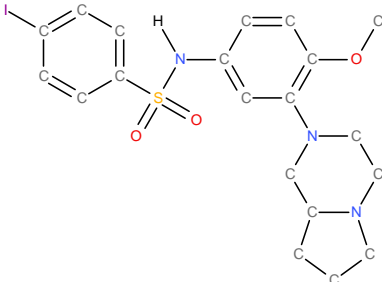
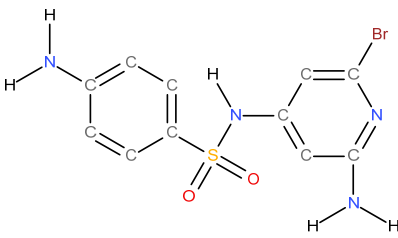
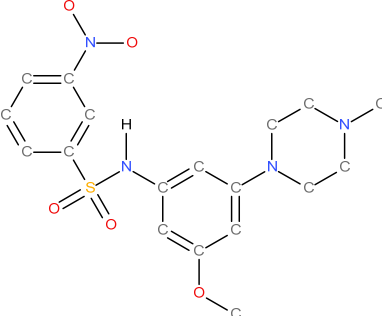
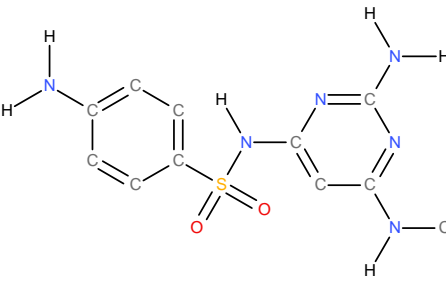
else:
    print 'These were not isomorphic.'

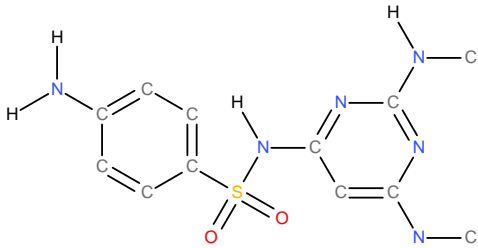
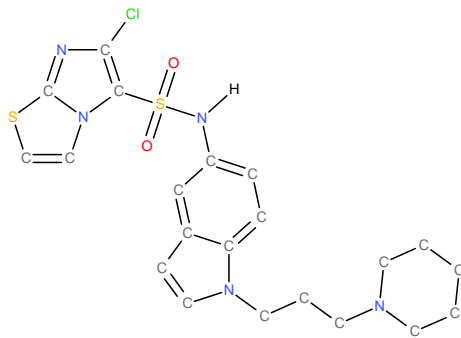
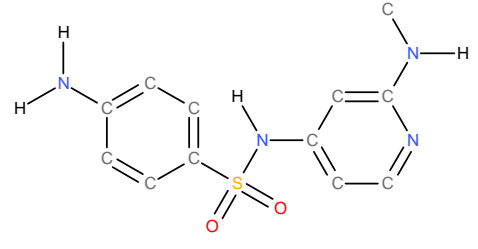
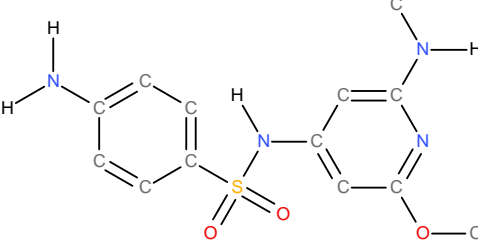
print 'For graph:', x, '!', matched_graphs*100/total_graphs, 'percent of the graphs were matched.'

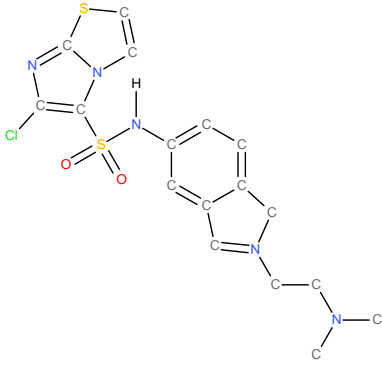
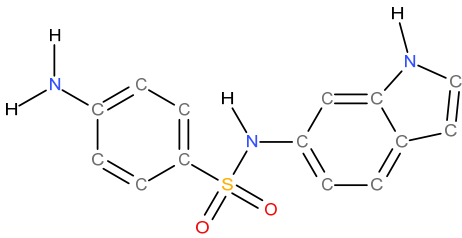
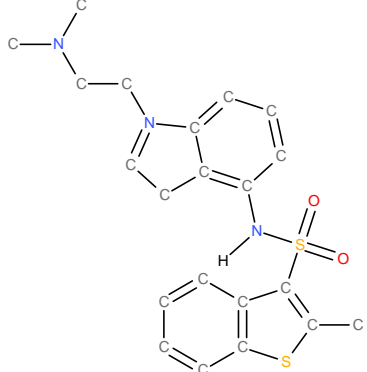
```

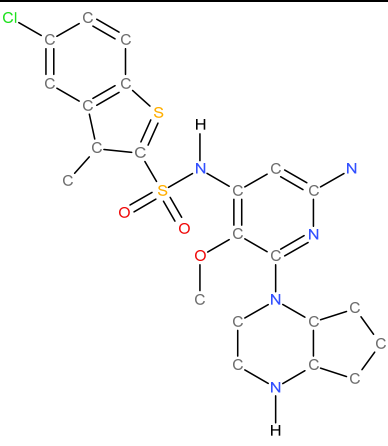
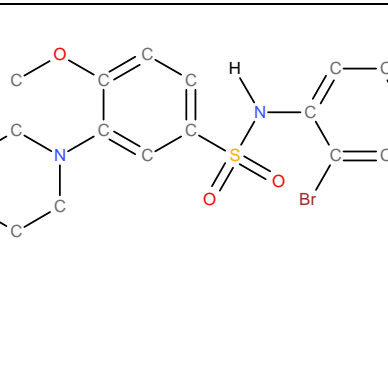
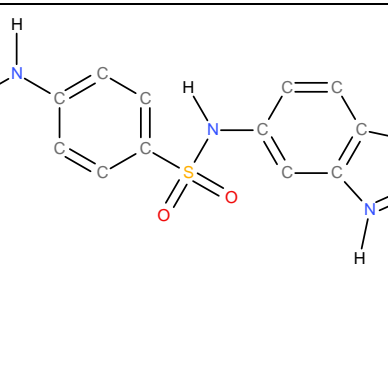
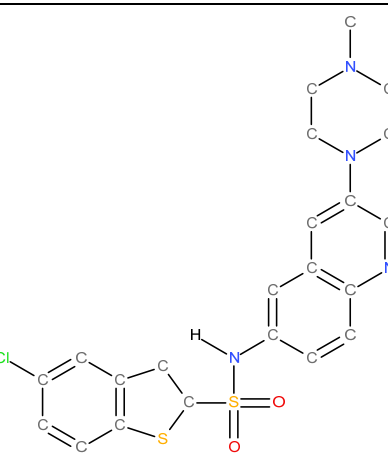
Appendix B – Solutions to Pharmacophore Case Study

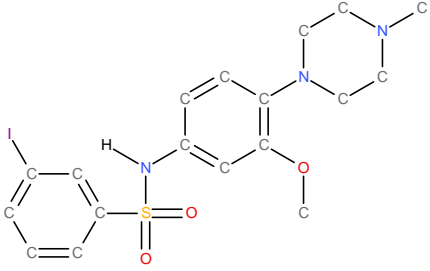
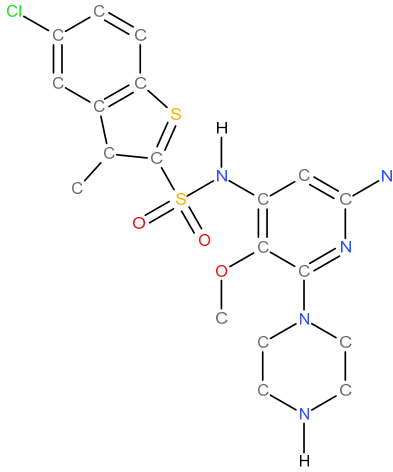
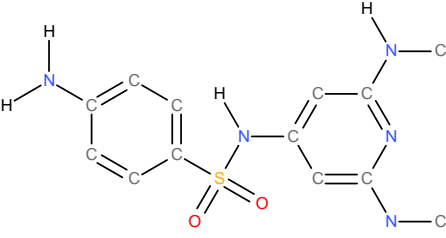
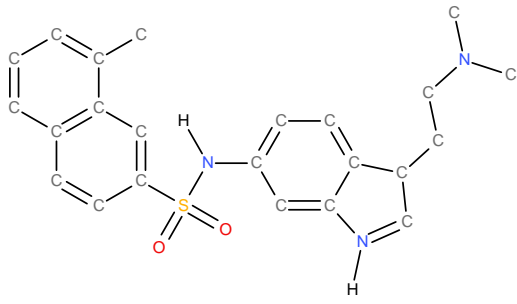
Solution Rank	Structure	Metric (Strain/Local Minima Strain)
1		1.09
2		1.13
3		1.45

4		1.47
5		1.55
6		1.61
7		1.61

8		1.80
9		1.93
10		1.96
11		2.04

12		2.15
13		2.22
14		2.39

15		2.56
16		3.11
17		3.28
18		3.40

19		3.88
20		3.92
21		4.68
22		5.24