Generation of a virtual library of terpenes using graph theory, and its application in exploration of the mechanisms of terpene biosynthesis

A thesis submitted in partial requirements for the degree of

MASTER OF SCIENCE
IN
BIOINFORMATICS

RHODES UNIVERSITY, SOUTH AFRICA

Department of Biochemistry and Microbiology

By

Washington Dendera

DECEMBER 2018

RHODES UNIVERSITY
Where leaders learn

Abstract

Terpenes form a large group of organic compounds which have proven to be of use to many living organisms being used by plants for metabolism (Pichersky and Gershenzon, 1934; McGarvey and Croteau, 1995; Gershenzon and Dudareva, 2007), defence or as a means to attract pollinators and also used by humans in medical, pharmaceutical and food industry (Bicas, Dionísio and Pastore, 2009; Marmulla and Harder, 2014; Kandi *et al.*, 2015). Following on literature methods to generate chemical libraries using graph theoretic techniques, complete libraries of all possible terpene isomers have been constructed with the goal of construction of derivative libraries of possible carbocation intermediates which are important in the elucidation of mechanisms in the biosynthesis of terpenes.

Virtual library generation of monoterpenes was first achieved by generating graphs of order 7, 8, 9 and 10 using the Nauty and Traces suite. These were screened and processed with a set of collated Python scripts written to recognize the graphs in text format and translate them to molecules, minimizing through Tinker whilst discarding graphs that violate chemistry laws. As a result of the computational time required only order 7 and order 10 graphs were processed. Out of the 873 graphs generated from order seven, 353 were converted to molecules and from the 11,7 million produced from order 10 half were processed resulting in the production of 442928 compounds (repeats included).

For screening, 55 366 compounds were docked in the active site of limonene synthase; of these 2355 ligands had a good Vina docking score with a binding energy of between -7.0 and -7.4 kcal.mol^{-1}. When these best docked molecules were overlaid in the active site a map of possible ligand positions within the active site of limonene synthase was traced out.

Declaration

I Washington Dendera, declare that this thesis submitted to Rhodes University is wholly my own work and has not been previously submitted for a degree at this or any other institution.

The research described in this thesis was carried out as part of the one-year MSc coursework and research thesis programme in Bioinformatics and Computational Molecular Biology, from July 2018 to 14 December 2018 under the supervision of Prof Kevin Lobb.

Signature:

Date:

Acknowledgements

This work is supported by the National Research Foundation (NRF) South Africa (Grant Number 105267) allocated to Prof Ozlem Tastan Bishop.

I would like to thank Rhodes University for the learning resources. I also would like to thank the CSIR Centre for High Performance Computing (the CHPC) for their supercomputing resources.

I would like to thank Prof Ozlem Tastan Bishop to whom I am grateful for being not only supportive financially but in a motherly way made it her business to be supportive in every manner possible, as well as instilling the sense of urgency with which every matter had to be attended to maximize efficiency. I also thank her for accepting me to join the RUBI (Research Unit in Bioinformatics) group at Rhodes University. I thank my supervisor Prof Kevin Lobb for a great mentorship experience and guiding me through all the new scopes I had to learn, in and outside normal curricula so as to meet the required computational chemistry expertise the research demanded.

With great pleasure I also take to acknowledge efforts of all who enabled the success of my study, my lectures Dr Vuyani Moses, Dr Rowan Hatherley, Miss Caro Ross, Mr Jeremy Baxter. I thank Dr Thomas Musyoka for his support with literature prior to my commencement of studies and Mr Michael Glenister and Mr Olivier Sheik Amamuddy whose pc maintenance skills came in extra handy in rescuing my data each time I encountered data threatening errors.

I am also thankful for non-academic assistance I received to make my studies a success, I thank Dr C Grobler for her financial assistance in getting a study visa, I thank Mr Ryan Ferguson for providing me with a monitor to use at home. I also am grateful to all RUBI group colleagues, Chemistry department colleagues Arthur Sarron and Tendamudzimu Tshiwawa my family for their constant support.

Table of Contents

Contents

List of Figures

List of Tables

List of web servers and applications

Nauty and Traces:	http://pallini.di.uniroma1.it/
Protein Dabank RCSB:	https://www.rcsb.org/
Tinker:	https://dasher.wustl.edu/Tinker/

List of acronyms

CDP-MEP:	4-diphosphocytidyl-2-C-methyl-Derythritol-2-phosphate
DMAPP:	Dimethylallyl pyrophosphate
DPP:	Dimethylallyl diphosphate
DXP	1-Deoxy-D-xylulose 5-phosphate
GPP	Geranyl pyrophosphate
Hbond:	Hydrogen Bonds
HTVS:	High Throughput Virtual Screening
IPP:	Isopentenyl diphosphate
LS:	Limonene synthatase
MD:	Molecular Dynamics
MM:	Molecular Mechanics
MEP:	Methylerythritol Phosphate Pathway
MVA:	Mevalonate pathway
MVK:	Mevalonic Acid Kinase
NAUTY:	No AUTomorphism Yes
PDB:	Protein Data Bank
PMK:	Phosphomevalonate Kinase
TS:	Terpene synthases
QM:	Quantum Mechanics

Chapter 1: Literature review

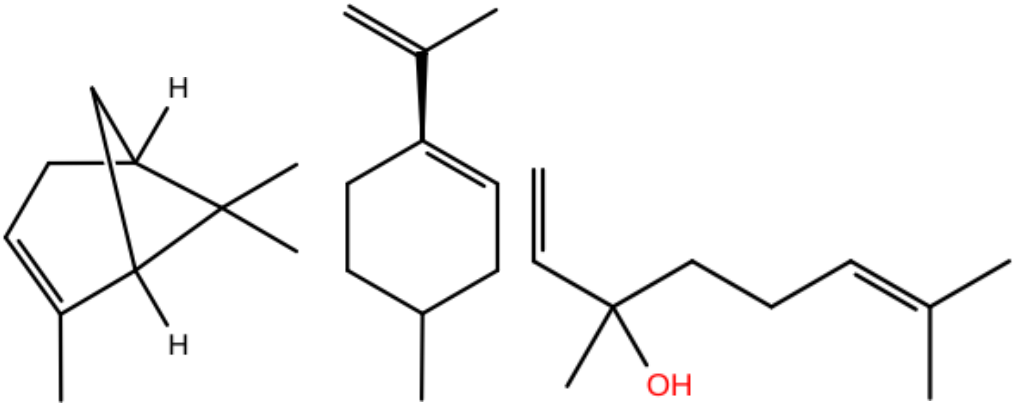
1.1 Introduction

Terpenes form a large group of plant natural products (Pichersky and Gershenzon, 1934; McGarvey and Croteau, 1995; Gershenzon and Dudareva, 2007), which consists of about 63,000 known chemical compounds (Oldfield and Lin, 2012; Tian, Poulter and Jacobson, 2016). Their sources of production span all the branches of living organisms, and in each of these living organisms they participate in crucial roles (Dicke, 1999; Kessler and Baldwin, 2002; Köllner, Gershenzon and Degenhardt, 2009; Kandi *et al.*, 2015). These natural hydrocarbons are constructed from isoprene units (containing 5 carbon atoms) which combine to form a variety of products each containing a multiple of 5 carbon atoms (Pichersky and Gershenzon, 1934; Ajikumar *et al.*, 2008; Morehouse *et al.*, 2017). In plants the isoprene units are formed either through the mevalonate pathway (MVA) (Bloch, 1965; Miziorko, 2011) in the cytosol or the methylerythritol phosphate pathway (MEP) (Chang *et al.*, 2013) in plastids (Jia et al., 2018). Due to structural cellular differences with plants, other organisms resort to the MVA pathway, examples being, yeast, animals and archaea while bacteria often use the MEP pathway, however some bacteria use both (Boucher and Doolittle, 2000; Vickers and Sabri, 2015; Zebec *et al.*, 2016). The word terpene itself is derived from the word turpentine; the substance turpentine contains a mixture of terpenes (Hostettmann *et al.*, 2014).

After formation, the isoprene units isopentenyl pyrophosphate (IPP) and its isomer dimethylallyl pyrophosphate (DMAPP) undergo condensation reactions which result in the formation of precursors of terpenes, geranyl diphosphate (GPP, C10), farnesyl diphosphate (FPP, C15), and geranylgeranyl diphosphate (GGPP, C20) (Bochar et al., 1999; Breitmaier, 2006; Miziorko, 2011; Chang et al., 2013).

Structural variations amongst terpenes are accredited to the action of a group of enzymes called terpene synthases (TP). Terpene synthases further process the terpene intermediates to form various terpene products (Cane, 1990; Sacchettini and Poulter, 1997; Christianson, 2006, 2017; Yao, Chen and Guo, 2018). These are classified depending on the number of

isoprenes added to the terpene structures. Terpenes composed of 2, 3, 4, 5, 6 or 7 isoprene units are named monoterpenes (C10), sesquiterpenes (C15), diterpenes (C20), sesterterpenes (25), triterpenes (C30) and sesquiterpenes (C35) respectively (Tian, Poulter and Jacobson, 2016) . As an example, γ-humulene synthase from *Abie grandis* produces 52 kinds of sesquiterpenes from the sole substrate farnesyl pyrophosphate (Steele *et al.*, 1998; Little and Croteau, 2002; Yoshikuni, Ferrin and Keasling, 2006; Xu *et al.*, 2018). Terpenoids or (isoprenoids) are terpenes which contain an additional functional group, usually oxygen in the form of an alcohol or a ketone (Bicas, Dionísio and Pastore, 2009) (see Figure 1.00) shows examples of two monoterpenes and a terpenoid.



α-Pinene Limonene Linalool

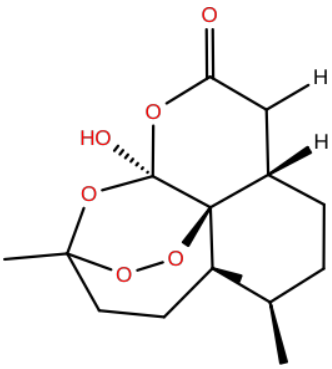
Figure 1.00: The terpenes alpha-pinene and limonene, and the terpenoid linalool (Gindulyte et al., 2018)

The action of different terpene syntheses on GPP and FPP, for example, results in a variety of of terpenes forming, such as monoterpenes, sesquiterpenes and diterpenes etc. (Tian, Poulter and Jacobson, 2016).; however, of major interest to this research are the monoterpenes. Monoterpenes are members of the terpene family containing ten carbon atoms (two isoprene units) (Croteau, 1987; Yao, Chen and Guo, 2018).

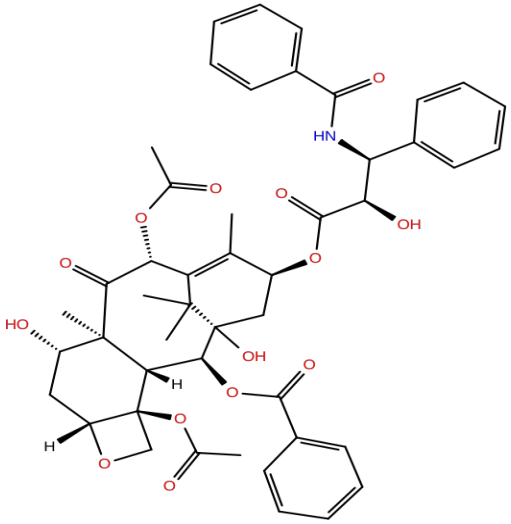
1.1.1 Uses of terpenes

The uses of terpenes span all walks of life, from plants that produce them using them in metabolical and defence systems (Dicke, 1999; Kessler and Baldwin, 2002; Köllner, Gershenzon and Degenhardt, 2009), to humans using them in home and industry (Bicas, Dionísio and Pastore, 2009). In plants terpenes, such as 1,8-cineole act as a repellent of insects or as a means to attract specific insects for pollination. Furthermore, they can be used as a means of indirect defense, where their release as volatile compounds attract enemies or predators of phytophagous insects (insects which feed on plant material) (Dicke, 1999; Al-Alawi, 2014; Filipe *et al.*, 2017). Plants like conifers defend themselves through the use of the oleoresin terpenoids defense system which makes use of diterpene resin acids (DRAs) (Hamberger *et al.*, 2011). In grapes like *Cabernet sauvignon* some terpenes like (+)-valencene and (-)-7-*epi*-*α*-*selinene* are produced in the anthers and pollen grains and are believed to be as a means of plant defense as well as a means of aid for pollination (Martin *et al.*, 2009; Kandi *et al.*, 2015). In most of those plants producing insect repellents, the main ingredients include at least one of the terpenes citronella, menthol, eucalyptol (1,8 cineole), limonene or linalool (Kandi *et al.*, 2015).

Terpenes also have a wide range of uses in the industrial realm, in medical, pharmaceutical, perfumery and food industries. In the medical and pharmaceutical industry, they have been used for the treatment of diseases (Bicas, Dionísio and Pastore, 2009). As Zhang et al notes, in 2002 the worldwide sales for terpene-based pharmaceuticals was approximately US $12 billion. Renowned terpene-based medicines include the anti-malaria drug Artemisinin (Klayman, 1985) and the anticancer drug Paclitaxel (Taxel®) (Wani *et al.*, 1971) (see Figure 1.01) (Zhang and Demain, 2005; O'Brien et al., 2018).



Artemisinin



Paclitaxel

Figure 1.01: Terpene based drugs antimalarial drug Artemisinin and anticancer drug Paclitaxel (Taxel®) (Gindulyte et al., 2018)

After extraction from natural sources, some monoterpenes become active ingredients for a variety of terpene products namely, resins, oils and various flavourings and fragrances (Bicas, Dionísio and Pastore, 2009). Essential oils are fragrant oils extracted from plants and comprise of a complex mixture of volatile liquids and lipophilic compounds, their composition is of mixtures of phenylpropanoids and terpenes (predominantly monoterpenes and sesquiterpenes (Rai and Kon, 2013; Herrera *et al.*, 2015). Essential oils have many uses which are of economic importance, such as additives in food, cosmetics and pharmaceuticals (Tongnuanchan and Benjakul, 2014).

Other pure terpenoid systems with medical use include eucalyptol (1,8- cineole), (see Figure 1.02), which is an oxygen containing monoterpenoid which is used at a broader scale in pharmaceutical and medical industries, as an antiseptic and analgesic. It also has immunostimulant activity, anti-tussive bronchodilator effects and myorelaxant, properties (Bastos *et al.*, 2011; Rocha Caldas *et al.*, 2015). 1,8-ceneole is also a constituent of *Elettaria cardamomum* (Zingii bereceae) also known as cardamom which is used in treating various gastrointestinal, cardiovascular and neuronal disorders (Miguel, 2010; Rocha Caldas *et al.*, 2015) .

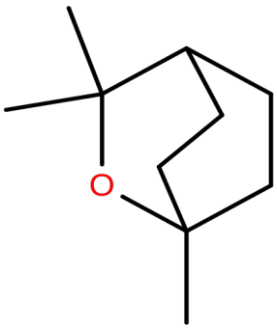


Figure 1.02: 1,8-cineole (Gindulyte et al., 2018)

Eucalyptus based medicines attribute their anti-inflammatory action to the presence to this 1,8–cineole (eucalyptol). There is potential use with 1,8-cineole to deal with side effects of immunosuppressive treatments such as chemotherapy or treatments involving steroids (Williams, 2011).

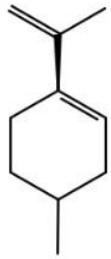
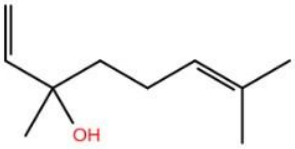
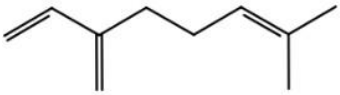
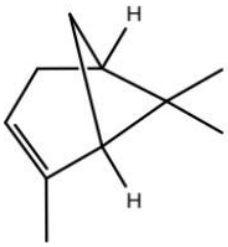
STRUCTURE	NAME OF TERPENE	SOURCE(S)
	Limonene	Lemon
	Linalool	Citrus fruits, lavender, rose, rosewood and coriander
	Myrcene	Myrtles and cannabis
	α-Pinene	Pines

Figure 1.03: A range of terpenes together with their sources (Gindulyte et al., 2018)

Figure 1.03 shows some more terpenes or terpenoid systems, together with the natural source from which they are extracted.

1.1.2 Mechanism of formation of terpenes

Biosynthesis of terpenoids involves four stages, the first stage involving the formation of IPP and DMAPP through the MVP or the MEP pathways (Boucher and Doolittle, 2000; Tiwari, Brunton and Brennan, 2013; Zebec *et al.*, 2016). The second stage involves isoprenyl diphosphate synthases which convert the products of the MVP or MEP pathways (Jia et al., 2018). If the condensation reaction happens in a 1'- 4' or "head-to-tail" fashion, geranyl diphosphate (GPP, C10), farnesyl diphosphate (FPP, C15), and geranylgeranyl diphosphate (GGPP, C20) are consecutively formed (Boucher and Doolittle, 2000; Dewick, 2002). The third stage involves the cyclizations of C10-C20 diphosphates to produce parent carbon skeletons of each terpene class. GPP is converted to monoterpenes, FFP to sesquiterpenes and GGPP to diterpenes. FFP and GGP can also dimerise to form the precursors of C30 and C40 terpenoids respectively. In the 4th stage conversions of parent terpene groups to various terpene products is done through a multiple of oxidations, reductions, isomerizations and conjugation and transformations (Tiwari, Brunton and Brennan, 2013)

1.1.3 The Mevalonate pathway (MVP)

The mevalonate pathway also known as the isoprenoid pathway or HMG-CoA reductase pathway, is a metabolic pathway found in most eukaryotes, archaea, and some eubacteria (Zhao *et al.*, 2013). This pathway is responsible for the conversion of acetyl-CoA to isopentenyl pyrophosphate (IPP) and its isomer dimethylallyl diphosphate (DPP) (Dewick, 2002; Miziorko, 2011). Miziorko states that, the metabolical products of this pathway accounts for the production of sterols and polyisoprenoids in fungi, plant cytoplasm, animals, eukaryotes, archaea and some eubacteria.

The mevalonate pathway (see Figure 1.04) begins by acetylation of coenzyme A (CoA) to form acetyl-CoA, condensation of two acetyl-CoA then produces acetoacetyl-CoA. Consequently, addition of a third acetyl-CoA to the acetoacetyl-CoA by a carbonyl condensation reaction results in the formation of 3-hydroxy-3-methylglutaryl-CoA (HMG-

CoA). HMG reductase (HMGR) facilitates the reduction of HMG-CoA to give mevalonate (MVA). After going through a series of phosphorylation steps which are catalysed by mevalonic acid kinase (MVK) and phosphomevalonate kinase (PMK), MVA is converted to mevalonate-5-diphosphate. This is then converted, in an ATP-coupled decarboxylation reaction by the enzyme mevalonate-5-diphosphate decarboxylase (MPD) to IPP. The interconversion between two of the isomers IPP and DMAPP is then facilitated by the enzyme IPP: DMAPP isomerase (Bloch, 1965; Vella, 2005; Miziorko, 2011).

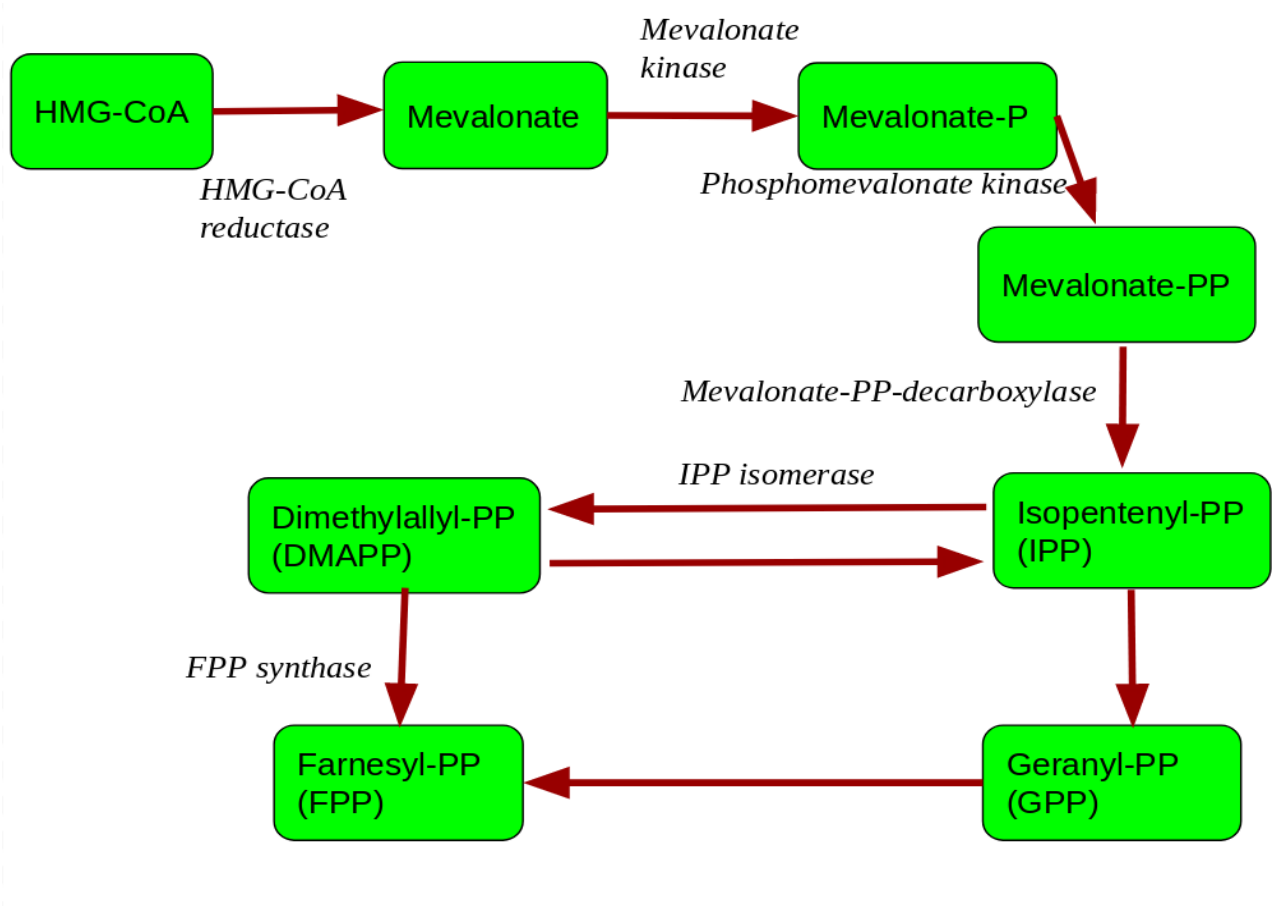


Figure 1.04: The Mevalonate Pathway (Miziorko, 2011; Thurnher, Nussbaumer and Gruenbacher, 2012)

1.1.4 The Non-mevalonate or Methylerythritol phosphate (MEP) pathway

The "MEP pathway was discovered in the 1990's" (Rohmer *et al.*, 1993; Zhao *et al.*, 2013), and in the same way as the MVP pathway its purpose is for generating IPP and its isomer DMAPP. The MEP pathway is found in eubacteria, green algae and higher plants. In plants both the MVP and MEP pathway exist but in segmented regions (Eisenreich *et al.*, 2004; Chang *et al.*, 2013).

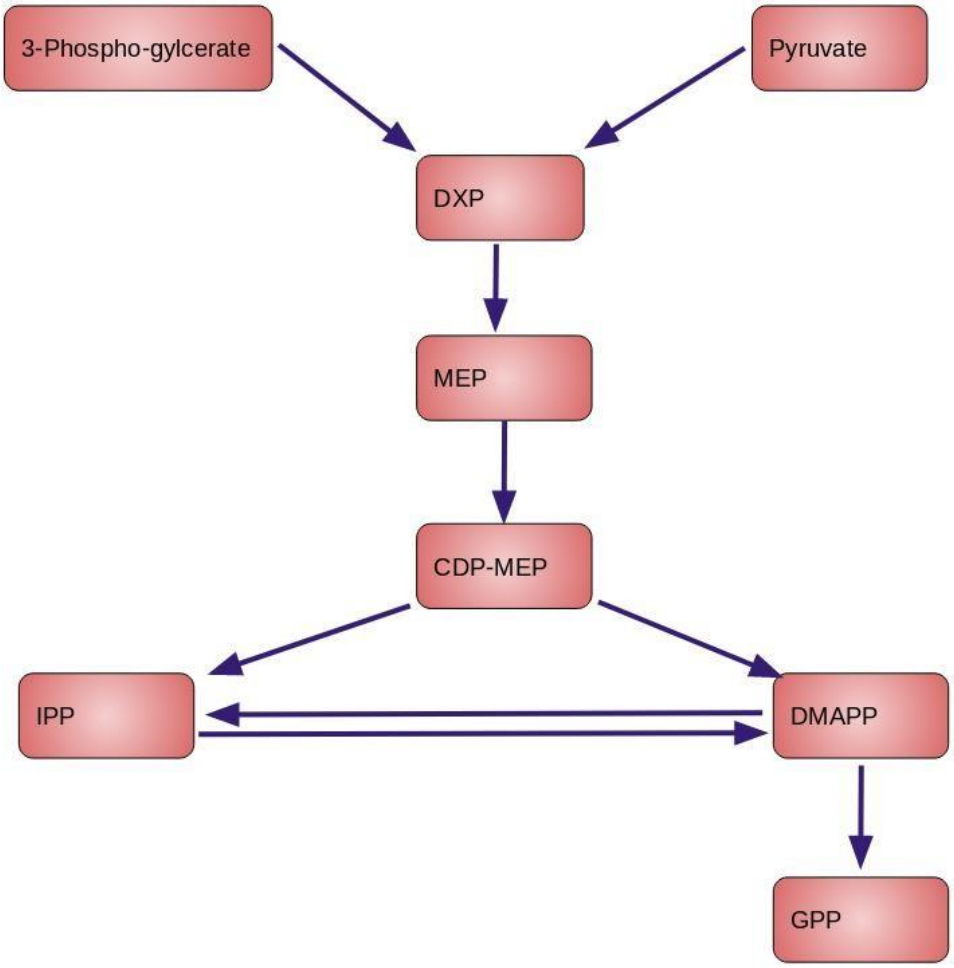


Figure 1.05: Methylerythritol phosphate (MEP) pathway or Non-mevalonate pathway (Zhao et al., 2013)

Zhao *et al* states that the initial reaction in this pathway (see Figure 1.05) is a thiamin diphosphate dependant condensation between D-glyceraldehyde-3-phosphate and pyruvate resulting in 1-Deoxy-D-xylulose 5-phosphate (DXP), which, through reductions is isomerized to MEP by the enzyme DXP reductoisomerase (DXR/IspC). Through the enzyme

CDP-ME synthetase (CMS, IspD), 4-diphosphocytidyl-2-*C*-methylerythritol (CDP-ME) is formed from the MEP and cytidine-5'-triphosphate precursors. The hydroxyl group of (CDP-ME) is then phosphorylated by an ATP dependant enzyme (ISPE) to produce 4-diphosphocytidyl-2-*C*-methyl-D-erythritol-2-phosphate (CDP-MEP) which is then catalysed by IspF (MCS) to 2-*C*-methyl-D-erythritol-2,4-cyclodiphosphate (MecPP). Through the catalysis of enzyme IspG (HDS), 4-hydroxy-3-methyl-butenyl-1-diphosphate (HMBPP) is formed from the opening of the ring of the cyclic pyrophosphate and C3 reduction of MecPP.

In the final step of the MEP pathway, unlike the MVP (which requires IPP: DMAPP isomerase, IDI) to produce the isomers IPP and DMAPP, the MEP pathway rather makes use of IspH (HDR) to convert HMBPP to IPP and DMAPP (Zhao *et al.*, 2013 and references therein).

1.1.5 Laboratory synthesis of terpenes

Methods of artificially synthesizing terpenes differ, Zebec *et al.* in a synthetic biology study showed that terpene production is also possible through manipulation of a natural approach by using engineered bacteria and yeast (Zebec *et al.*). They demonstrated production of terpenes *via* biosynthetic pathways with the use of *Escherichia Coli* and Yeast (*Saccharomyces Cerevisae*). Gene enhancement in the *E. coli* production system involved addition of genes for the MVA pathway as well as inclusion of genes for enzymes GPP synthase and Limonene synthase. In *S. Cerevisae* genes for an engineered Farnesyl pyrophosphate synthase (FPP synthase) Erg20 enzyme were introduced for the production of sibinene and limonene. Production titre was boosted by optimization of gene regulation and growth conditions. Product diversity of the mutant libraries was attained by alteration of genes by incorporating genes for appropriate enzymes.

From a purely synthetic organic chemistry perspective, full and partial syntheses of terpenes are extensive in the literature. In an example of a partial synthesis, the monoterpene myrecene may be used as a starting material in the synthesis of (-)-menthol. The synthesis process involves the conversion of myrecene to diethylgeranylamine by adding diethylamine in the presence of a lithium catalyst. The product diethylgeranylamine is isomerised to 3R-

citronellal enamine, and this is transformed to (-)-menthol (Leffingwell and Shackelford, 1974; STINSON, 1996; Iwata, Okeda and Hori, 2004).

1.1.6 Terpene synthases

All organisms spanning from plants, bacteria, insects, social amoeba, fungi to archaea possess terpene synthases (Bohlmann, Meyer-Gauen and Croteau, 1998; Yamada *et al.*, 2015; Beran *et al.*, 2016; Chen *et al.*, 2016; Jia *et al.*, 2018). As highlighted before, several enzymes, including terpene synthases are responsible for catalyzing the production of terpenes, from small 5C units called isoprenes (Pichersky and Gershenzon, 1934; Ajikumar *et al.*, 2008; Morehouse *et al.*, 2017). A means of classification of terpene synthases is by splitting them into two categories, Type 1 and Type 2 terpene synthases. Type 1 terpene synthases utilize a divalent cation (Mg^{2+} or Mn^{2+}) to remove the diphosphate group (from IPP/GPP/GGPP) in order to generate carbocations, whereas Type 2 terpene synthases catalyse the protonation of a C-C π-bond (or an epoxide) to accomplish the same result (McGarvey and Croteau, 1995; Christianson, 2006; O'Brien *et al.*, 2018) .

Most of the known plant terpene synthases are of Type 1 (Chen *et al.*, 2011). As an example, experimental observations by Chen *et al* demonstrated, using grand fir, that the monoterpene synthases, myrcene synthase, (-)-pinene synthase and, (-)-limonene synthase all require Mn^{2+} for catalytic activity; however, Mg^{2+} was proved ineffective as a divalent cofactor. They noted that structurally the enzyme sequences contain an aspartate-rich element (DDXXD) which is believed to be connected to metal binding (Ashby and Edward, 1990; Chen *et al.*, 2011). In addition to the Mn^{2+} the enzymes require a monovalent potassium cation, K^{+}, a feature which applies to gymnosperms and not angiosperm monoterpene synthases (Savage, Hatch and Croteau, 1994; Chen *et al.*, 2011).

Since carbocations intermediates are highly reactive, the terpene synthase active sites are usually non-polar (greasy) enabling them to accommodate carbocations without allowing nucleophilic centres to quench the carbocationic centre. The non-polar active sites will allow for a longer-lived carbocation centre. The absence of water in the active site prevents the formation of alcohols from the carbocationic species. Lastly, they make use of weak

interactions to promote creation of one terpene product over another (Bojin and Tantillo, 2006; Hong and Tantillo, 2010, 2011, 2013; O'Brien *et al.*, 2018).

Yamanda *et al* states that terpene synthases from bacterial, fungal, and plant origin have a construct containing two conserved metal binding domains which is composed of an amino acid rich motif [D/N)DXX(D/E) or DDXXXE] positioned in either of the ranges of 80-120 or 230-270 amino acids of the N-terminus. Plant terpene synthases contain a conserved N-terminal region (PF01397) which bacterial terpene synthases lack (Whittington *et al.*, 2002; Yamada *et al.*, 2015).

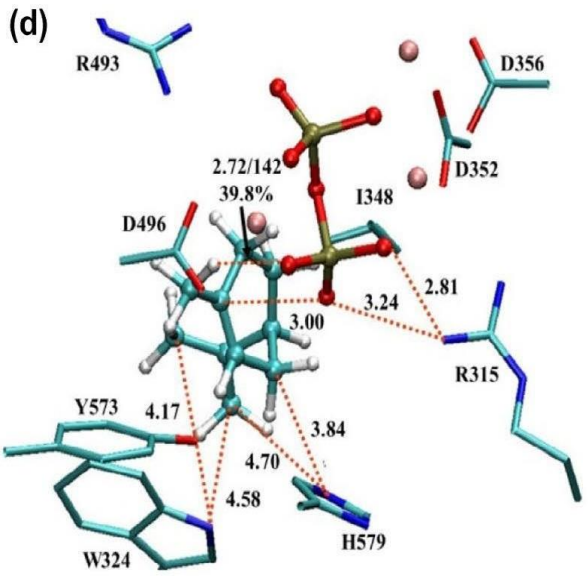
Hyatt *et al.* state that limonene synthase consists of two helical domains; the C-terminal possessing the active site and the N-terminal which is of unknown function (Christianson, 2006; Hyatt *et al.*, 2007). Extending past the N-terminal is the N-terminal strand which folds back across the C-terminal forming a part of the active site "lid or cap". Hyatt *et al.* presume that this lid shields the carbocations from water. The significance of the well-conserved N-terminal region is yet to be discovered; it is imperative however to note that this region is well conserved (Hyatt *et al.*, 2007).

1.1.7 Monoterpene synthase reaction mechanisms

In the past research has established that the reaction mechanism for the formation of a terpene skeleton mediated by a Type 1 monoterpene synthase is initiated by a divalent metal ion-dependent ionization of the substrate (Christianson, 2006; Degenhardt, Köllner and Gershenzon, 2009; O'Brien *et al.*, 2018). This cationic intermediate will subsequently go through a string of cyclizations, hydride shifts or various rearrangements until the reaction is terminated by loss of a proton or the addition of a nucleophile (Croteau, 1987; Degenhardt, Köllner and Gershenzon, 2009).

Yao *et al.* focused on the reactions producing 4(*S*)-α-terpinyl carbocation starting from 3(*S*)-linalyl diphosphate ((3*S*)-LPP) and in generating the 4(*S*)-limonene product. In their research

they noted that in 4(*S*)-limonene synthase ((4*S*)-LS), 4(*S*)-α-terpinyl carbocation is changed by deprotonation to 4(*S*)-limonene using the histidine His579 as a base and stabilizer for the 4(*S*)-α-terpinyl carbocation (see Figure 1.06). Using a QM/MM approach they observed that both wild type and recombinant (4*S*)-LS in mint species were restricted to producing a specific main product (4*S*)-limonene. This was unlike in (+)-bornyl diphosphate synthase (BPPS) which produced BPP as the main product. As a point of difference lying in the highly conserved active sites of LS and BPPS they noted that this difference in observation was due to the difference His579 (LS)/Phe578 (BPPS). Since they found in the LS system that the distance between the His579 of the product state of the (4*S*)-α-terpinyl carbocation and the C atom of the positively charged methyl group was 4.70 Å (see Fig 1.06), they wanted to further establish if this was the main base or one of the bases involved in the protonation of 4(S)-α-terpinyl carbocation. The resolve was that in the active site of (4S)-LS, His579 may accept a proton from the positively charged methyl group of the 4(S)-α-terpinyl carbocation to promote formation of limonene, but in the case of BPPS Phe578 in the active site provides hydrophobic interactions which promote the formation of BPP, through an intermediate 2-bornyl carbocation. The results from their further tests after decreasing the distance between the hydrogen atom (H) and nitrogen atom (N) of His579 proved that proton transfer does not occur synchronously with other reactions since this would require free movement of the substrate within the active site (Yao, Chen and Guo, 2018). This study highlights how nuances in the varying active sites may result in very large differences in product outcome for terpene synthases. In conclusion they noted that the reaction pathway of (4S)-LS-catalysed



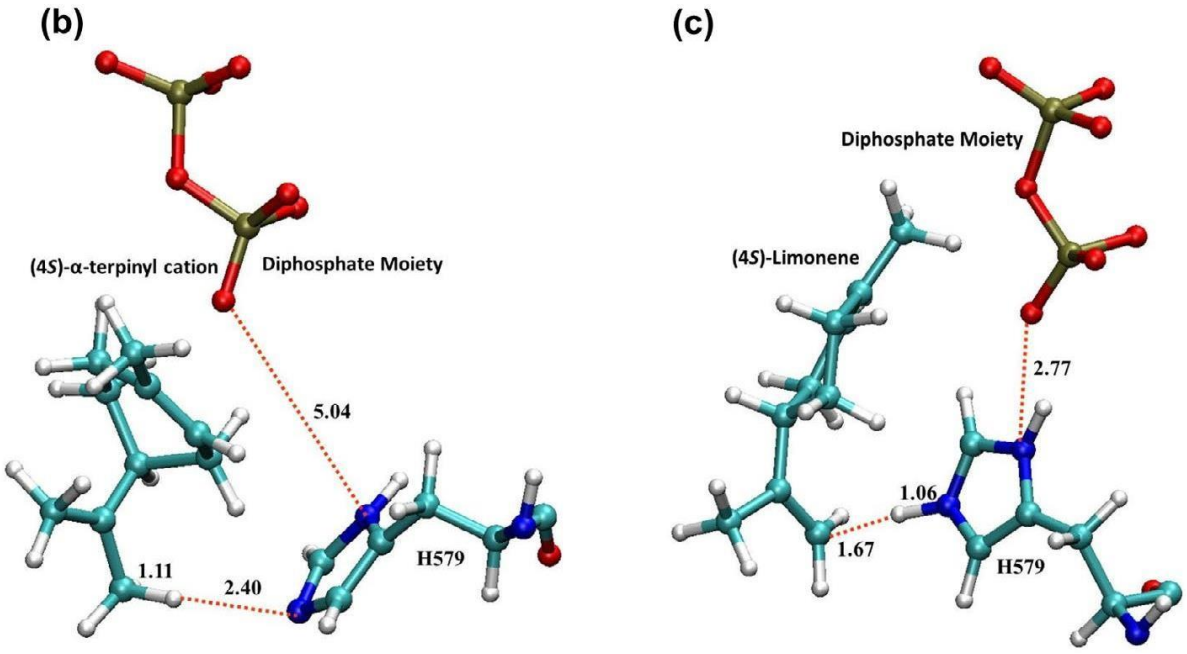


Figure 1.06: (b) and (c) respectively show 4(S)-α-terpinyl carbocation before protonation and after protonation. (d) shows a distance of 4.70 Å, between His579 and the C atom on positively charged methyl group of the product state of the (4S)-α-terpinyl carbocation. (Yao, Chen and Guo, 2018). (Reproduced with permission)

formation of limonene follows a coordinated-asynchronous (three-steps) reaction pathway consisting of isomerisations,cyclisation process and proton transfer process. The formation of the 4(S)-α-terpinyl carbocation results from the 3(S)-LPP isomerization and cyclisation process and this is was noted as the rate determining step. In addition they also concluded that proton transfer from 4(S)-α-terpinyl carbocation may use His 579 as a general base.

1.1.8 Terpene synthase product space exploration

Terpene synthases, during formation of terpenes accommodate carbocation intermediates (Christianson, 2017). Tian *et al.* performed an investigation of all possible cations (containing 10 C atoms) as intermediates for reactions within terpene synthases. Their research was aimed at inferring enzyme function (in terms of target terpenes) to protein sequences amongst terpene synthases. The strategy was to explore possible enzyme activity amongst uncategorized enzymes. They achieved this by building a library of potential carbocations using a computer aided simulation approach. Out of the 74 possible cyclic systems identified and constructed at least 5 were represented among the known characterized terpene products.

In terms of generating a full set of carbocations from the starting systems, simulations to

perform virtual carbocation rearrangements were performed (in the gas phase).

Figure 1.07 illustrates their general workflow in generating the full carbocation set from starting structures, together with the five reaction types were considered in order to generate the full set (intramolecular alkylation of double bonds, alkyl shifts, hydride shifts, 1,2-methyl shifts and proton transfers).

.

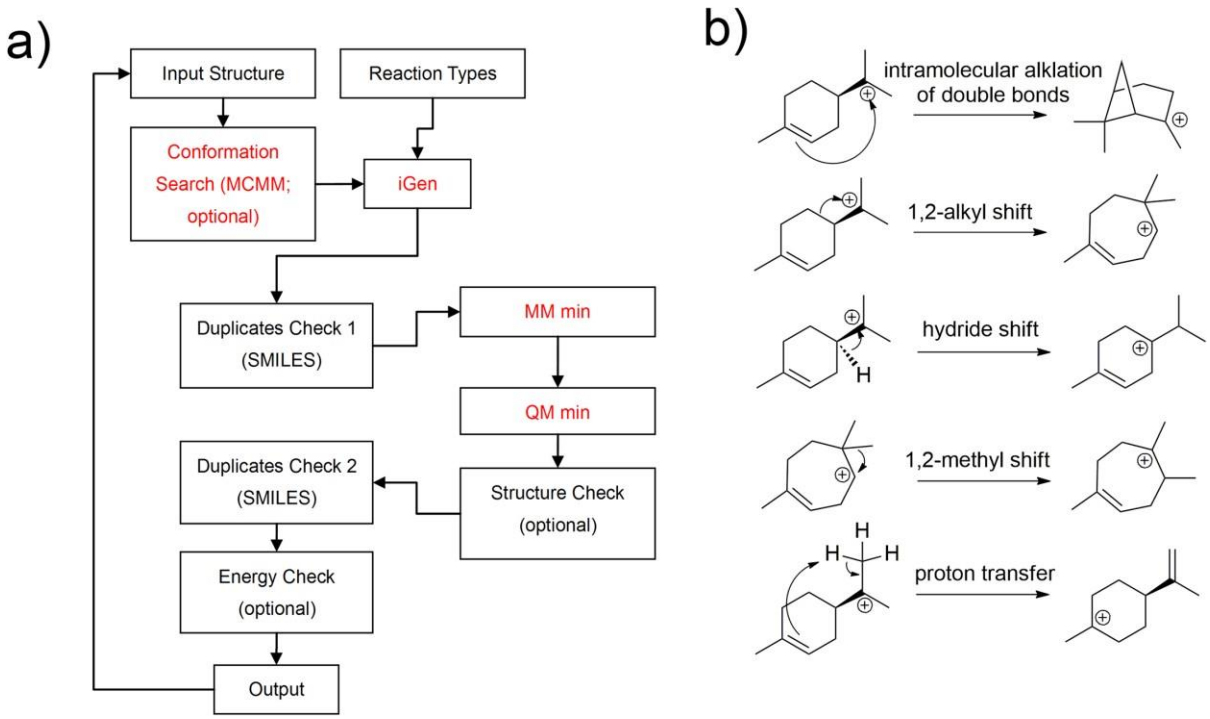


Figure 1.07: a) Generation of a full set of carbocations from a starting set; b) 5 allowed transformations (reaction types) to build the full set (Tian, Poulter and Jacobson, 2016). (Reproduced with permission).

After carrying each of these reactions (in the virtual sense) for each of the carbocationic intermediates, 18758 distinct carbocations were obtained. To consider next was stability of the carbocations with the intention of avoiding secondary carbocations ("since secondary carbocations are avoided in most terpene reactions"). Use of more stringent energy filters ranging from 0 kcal/mol -10 kcal/mol energy cut offs ("energies relative to the geranyl carbocation, in kcal/mol") resulted in decrease particularly of the less stable secondary carbocations from 48% to 16% in the set. After classification of their compounds 5 out of the identified 74 ring systems could be identified as corresponding to existing natural compound.

1.1.9 Post modification of terpenes

Although the focus of this work is on mechanism within terpene synthases, it is important to remember that post-modification often occurs in the process of terpenoid biosynthesis. Enzymes such as cytochrome P450 monooxygenases and oxidoreductases are entailed in the post-modification of the terpene skeletons to yield the various terpenoids found in nature (Marta *et al.*). For example, (+)-camphor is converted by cytochrome P450 monooxygenase from *Pseudomonas putida* to 5-*exo*-hydoxycamphor, **α**-pinene is oxidised to **α**-pinene oxide by pinene oxygenase, and eucalyptol (1,8-cineole) is converted by P450 monooxygenase from *Bacillus cerus* to 2*R-endo*- or *2R-exo*-hydroxy-1,8-cineole (Marmulla and Harder, 2014). Yet another example is that of (+)-nootkatone, a sesquiterpene, which is produced *via* oxidation of valencene by a member of the P450 monooxygenase superfamily (Kutyna and Borneman, 2018).

This research will exploit the use of a terpene synthase limonene synthase as a target docking site since the virtual library being generated comprises of precursors of terpenes. Use of mono-terpene synthases then, helps to prove the closeness in relation between virtual library compounds and natural terpene precursors as well as help in elucidating on possible reaction mechanisms within the active site of the mono-terpene synthases. Limonene synthase is available as a crystal structure, with substrate bound from the RCSB (2ONG) (Hyatt et al. 2007).

1.2 Problem statement and justification

The Chemical Abstracts Service reports that over 60 million compounds have been produced through synthetic chemistry over the past 20 years and that the combined academic and commercial collections of small molecules worldwide are at an estimate of above 100 million (Lipkus *et al.*, 2008; ARNAUD, 2011; Ruddigkeit *et al.*, 2012) . A point to note however is that the existence of a large number of compounds cannot ensure that there are compounds that have efficacy against particular disease targets, including absence of side effects and

toxicity, and thus there is a need for computational drug design to search for innovative structures that may display better selectivity and efficacy. However, due to the immense number of possible compounds systematic exploration of chemical space is difficult. In order to work around this Ruddigkeit *et al.* have made use of the graph theory to this end. Their idea has involved the enumeration of molecules from first principles using sets of non-isomorphic graphs. In these graphs the nodes represented the atoms (C, N, O and more) and the edges bonds. The graphs were run through a series of filters (algorithms developed to screen the molecules according to rules of the protocol's chemistry) to generate a diverse set of compounds with acceptable chemistry, taking note that all valency rules were observed. Filters enabled screening of problematic functional groups and all undesirable compounds, the result then was a diversification of skeletons into "CNO" types, which after post processing further generated additional diverse molecules and the result was 166.4 billion substances which make up the complete GDB-17 chemical universe. The GDB-17 dataset contains millions of isomers and analogues of known drugs. When compared to the molecules in PubChem, the GDB-17 molecules are much richer in non-aromatic heterocycles, quaternary centres and stereoisomers. When GDB-17 and PubChem are compared with respect to molecules of a similar size, the number of molecules in GDB-17 proved to contain many orders of magnitude more of these compounds, with PubChem-17 being only 0.001% of GDB-17) (Ruddigkeit *et al.*, 2012) .

Various mechanisms on the formation of terpenes have been presented in the literature, however this information is not sufficient to deal with all terpene formation mechanisms. In consideration, the establishment of the GDB-17 database (a huge chemical database) has demonstrated that graph theoretic approaches are powerful (Ruddigkeit *et al.*, 2012). It is therefore in the light of the principles of the graph theory, that the interest of this MSc research seeks to employ and modify such techniques so as to follow a rigorous graph theoretical approach to generate a library of monoterpenes and monoterpene carbocations which will then be used to explore the substrate chemical space of a terpene synthase enzyme active site. Extension of this work is being considered to accommodate monoterpenoid ketones such as camphor, verbenone and fenchone (in terms of oxidative post-modification of terpenes by cytochrome P450 family enzymes).

1.3 Aim

The project seeks to generate a library of monoterpenes using graph theory techniques, and through high-throughput virtual screening (docking) on enzyme active sites, determine which of the library are possible product or intermediates in the mechanism mediated by the terpene synthase enzyme.

1.3.1 Objectives

1. To generate a comprehensive library of monoterpenes using graph theoretic techniques and to confirm if this matches literature
2. To perform docking of the generated library to monoterpene synthases and thus establish the most likely monoterpenes to be produced or to be intermediates in the transformation from GPP mediated by the synthase.

1.4 Overview of methodology

The application of graph theory exploits algebraic invariants to establish non-biased selection of possible binding patterns. In this research non-isomorphic graphs are generated using **geng** in the Nauty and Traces suite of software. All graphs generated are converted to 3D molecules by use of scripts (see appendix) to provide input to the Tinker suite of software to enable use of this software to perform molecular mechanics optimizations. Docking of the generated library to the terpene synthases is performed through the use of AutoDock Vina.

1.4.1 Graph theoretic techniques

This is based on the graph theory which makes use of graphs to model chemical structure. In this case the graph represents a molecule which is represented by atoms as vertices and bonds as edges. The procedure in question will be characterised by construction of complete sets of

non-isomorphic graphs through the use of the package "**geng**" in the Nauty and Traces suite. As a test case the **geng** code will be called to enumerate all graphs containing 7 vertices (given that this is a relatively small number). This will then be used to explore graphs with 10 vertices, ultimately corresponding to the generation of structures containing 10 carbon atoms. Non-isomorphic graphs generated will be converted to 3D through MM techniques by the exploitation of the program "Tinker", controlled through the use of Python scripts (see appendix).

1.4.2 Tinker

The Tinker suite of software provides for efficient geometry optimization of systems using a molecular mechanics force-field based approach. In particular scripts (see appendix) will be used to screen the sets of graphs and write them in a Tinker-readable format, from which molecular mechanics geometry optimizations will be possible.

1.4.3 Virtual screening (Docking)

As part of a high throughput virtual screening procedure for the structures generated. Molecular docking will be performed using AutoDock Vina. All filtered compounds from the library will be docked to a synthase target, limonene synthase, obtained from the PDB database. The detailed methodology for this will be discussed in chapter 4.

Chapter 2: Concepts and Software

2.1 Graph theory

Since molecules may be represented as a graph, that is a series of atoms (nodes) connected in various ways by bonds (edges) (Kearnes *et al.*, 2016), this research makes of use the principles of graph theory to create a graph representation of a molecule, followed by construction of the corresponding molecular models from these graphs. It is therefore imperative to point out that the graphs mentioned here are not the graphs of functions that many are familiar with, visually illustrated on an x-y plot. In the context of graph theory and this research, a graph is "a set of objects (nodes or vertices) that are connected together through edges or links" (Bang-Jensen, Jørgen and Gutin, 2007; accessed 31 January 2019). Graphs can either be directed or undirected (Mckay, 2013). Directed graphs are graphs with edges pointing in one direction whilst undirected graphs are graphs with non-directional edges (without pointers) (Bang-Jensen, Jørgen and Gutin, 2007; accessed 31 January 2019). Graphs with bidirectional edges may be argued as being undirected (if all edges are bidirectional) (Figure 2.00 A). For this research undirected graphs were used.

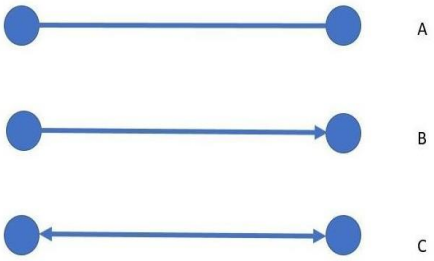


Figure 2.00: A is an undirected graph, while, B is a directed and C a bidirectional graph with only two nodes.

An example of a complex undirected graph with several nodes is illustrated in Figure 2.01. In this graph of order 11 (there are 11 vertices), the degree of this graph is 4 (the maximum connectivity or degree for any vertex in the graph is 4).

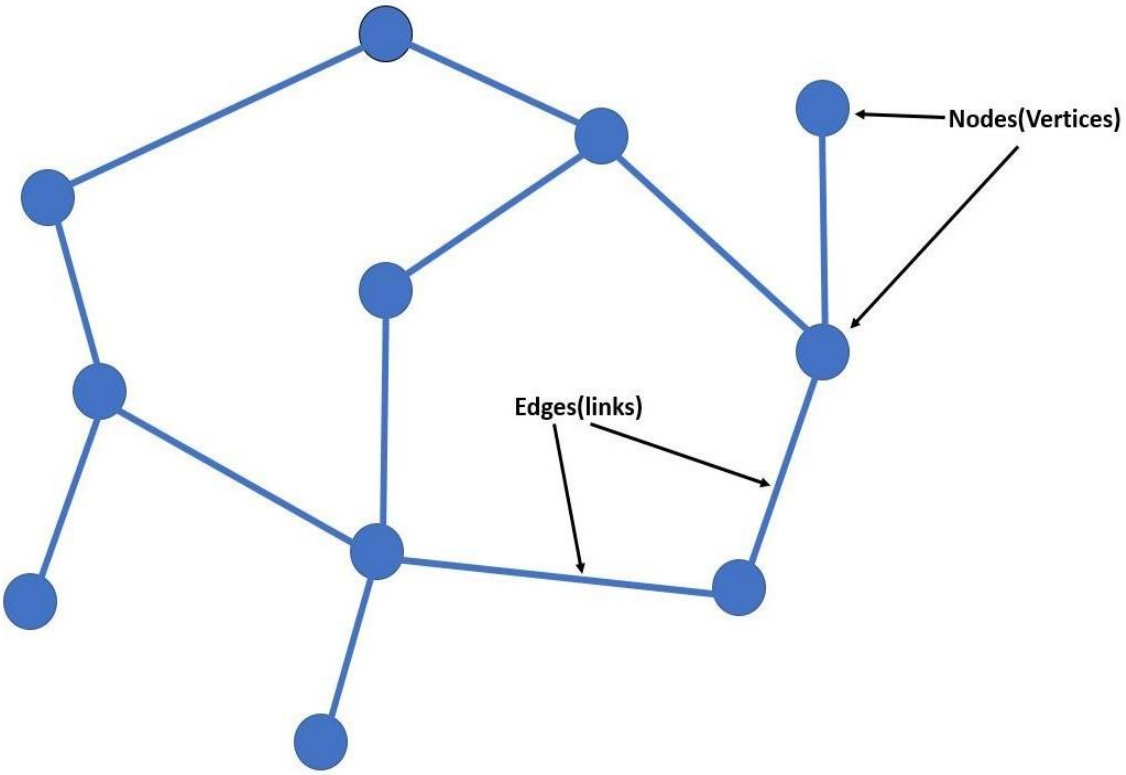


Figure 2.01: An illustration of a typical undirected graph.

2.1.1 Graph isomorphism

Since graphs are defined by their vertices and edges, and not by the way they are drawn it is important to be able to compare two graphs as being identical or different. Identical graphs are said to be isomorphic, that is they have the same number of nodes and exactly the same connectivity between nodes. More formally, two graphs are considered isomorphic if there is "a bijection between their vertex sets that maintains adjacency" (Tian, Liu and Xie, 2012; McKay and Piperno, 2014). This implies that there is freedom to represent a single graph in a variety of different shapes whilst preserving the integrity of connections as well as the number of vertices and edges. Non-isomorphic graphs on the other hand do not maintain the same type of connections, even if the number of vertices might be the same (Mckay, 2013). Figure 2.02 demonstrates this relationship, in the figure G1 and G2 are isomorphic, and G3 is non-isomorphic to either of the two G1 and G2. For example, it can be seen that G3 has a

vertex of degree 1, whilst the other two graphs have no such vertex. On the other hand, G1 and G2 as the correspondence A-1, B-2, C-3, D-4, E-5 maintains identical adjacency between the graphs and so these two graphs are isomorphic (there are other correspondences that work with this pair, but this suffices). G1 also has a set of isomorphisms with itself (e.g A-B, B-C, C-D, D-E), called automorphisms, the set of which is an automorphism group (McKay and Piperno, 2014).

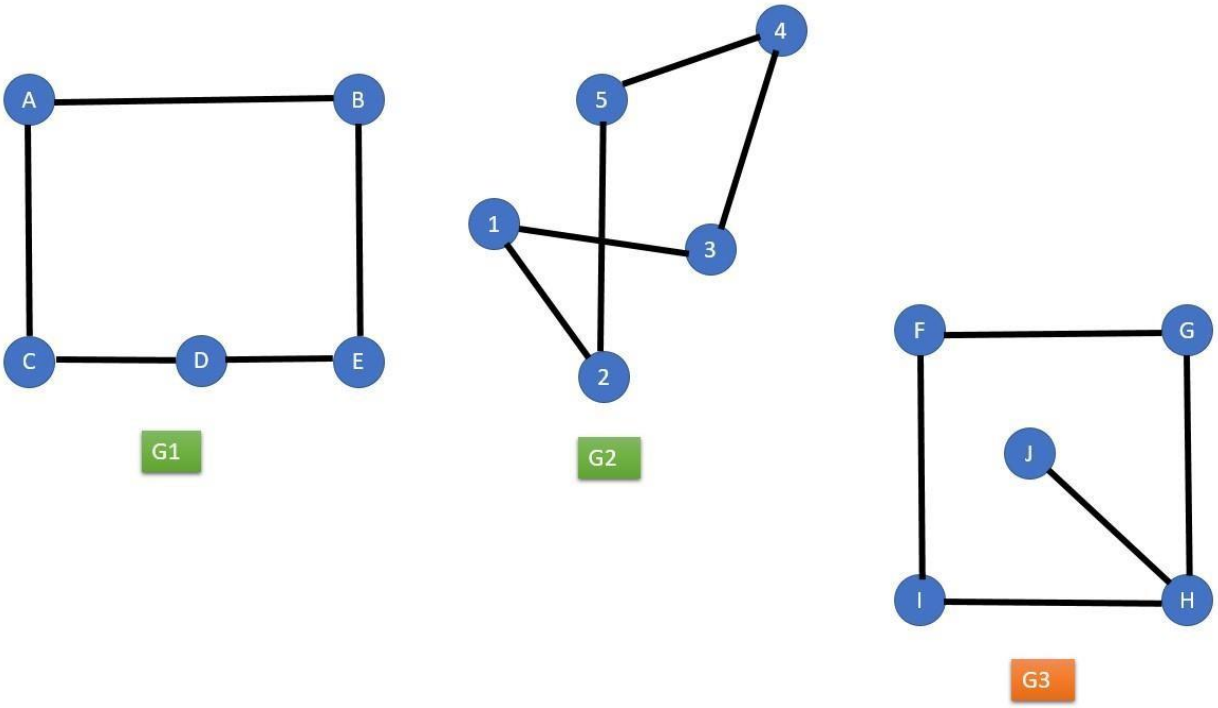


Figure 2.02: An illustration of isomorphism. In this figure G1 & G2 are isomorphic and G3 is non-isomorphic to neither G1 nor G2. Labels on graphs are for clarification purposes and serve no bias to orientation

Graphs in a non-isomorphic relationship show no correspondence no matter how many times the vertices in the graphs are spatially arranged; no edge connectivity correspondence may be found (Jonathan L. Gross, 1999). In the description of a molecule as a graph, where all vertices are identical (such as all being C atoms for example) a pair of non-isomorphic graphs of exactly the same degree are representations of two different molecules (isomers if, upon addition of hydrogen atoms, the atom count of each type is preserved) (Kearnes *et al.*, 2016). Further the existence of tools with the capability of generating a *complete* set of non-

isomorphic graphs of a particular degree provides a means by which one can generate a complete set of isomers (chirality issues withstanding).

In this project, a tool was required to generate these graphs, conducting graph creation in a manner that met these two requirements: firstly, that all graphs had to be non-isomorphic; and secondly that all possible graphs were created without exception. To this end the tool “geng” within the Nauty and Traces suite of software was used (McKay and Piperno, 2014).

2.1.2 Nauty and Traces

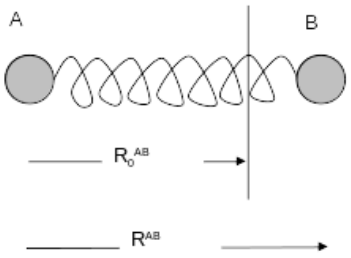
Nauty derives its name from the words “No AUTomorphism Yes” and is a package that includes a suite of programs (together with an API) within which there are the so-called “g tools” that are capable of processing files or graphs stored in graph6 or sparse6 format. However, the primary purpose of nauty is to work with graph automorphism groups. Traces is an alternate set of programs that performs much the same tasks as does nauty. One of the useful programs used from the Nauty and Traces suite, in this project was “geng” which is a tool for generating a *complete* set of non-isomorphic graphs of a particular order; in order to analyse the output, tools in the suite were used to change graph formats to easily readable formats included “listg” and the similar but less powerful “showg” (showg is a more compact program than listg with a subset of the functionality) (Mckay, 2013). These tools were used to generate graphs, and to format these graphs appropriately for further processing.

2.2 Tinker and Molecular Mechanics

2.2.1 Molecular Mechanics

Molecular mechanics is a modelling technique where bonded and non-bonded interactions are calculated (Patrikeev, 1973) using a "ball and spring" approach centred on Hooke's law (Lamberti *et al.*, 2002). The bonded interactions include the stretch, bend and torsion interactions, while the non-bonded interactions include van der Waal's and electrostatic interactions (Patrikeev, 1973; Burkert and Allinger, 1982).

For the bond energy calculation, the distance between two atoms, and a constant defining that particular pair of atoms are used to calculate the energy as a function of the interatomic distance (Equation 1) (Van Westen, Vlugt and Gross, 2011).



$$E(R^{AB}) = k^{AB}\left(R^{AB} - R_0^{AB}\right)^2 \quad (1)$$

In the MM3 forcefield (used in this study) this is elaborated further to describe more accurately a Morse potential, and this elaboration is in the form of a polynomial expression (Equation 2) (Allinger, Yuh, and Lii 1989).

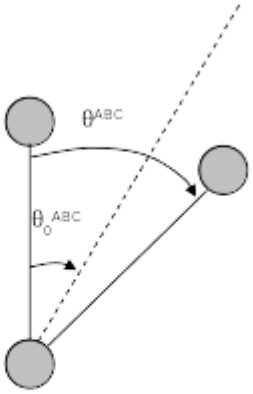
$$E(R^{AB}) = 71.94k^{AB}(R^{AB} - R_0^{AB})^2[1 - 2.55(R^{AB} - R_0^{AB}) + \frac{7}{12} \times 2.55(R^{AB} - R_0^{AB})^2] \quad (2)$$

In all cases k^{AB} is the force constant, while the difference describes the deviation of the bond length from equilibrium. In order to calculate energy of a system due to bond deviation, the contribution to the total energy of all bonds must be calculated according to the sum

(Equation 3).

$$E_{str} = \sum_{i}^{bonds} E(R^i) \tag{3}$$

Similar terms may be set up for bond angles and dihedrals (involving three or four consecutively bonded atoms) where the appropriate constants are dependent on the three (or four) atoms involved. So, generally for bond angles in molecular mechanics, again the deviation from equilibrium provides a means to calculate an energy cost due to this deviation (Equation 4).



$$E(\theta^{ABC}) = k^{ABC}\left(\theta^{ABC} - \theta_0^{ABC}\right)^2 \tag{4}$$

Again, there is a force constant that is dependent on the three atoms A, B and C (k^{ABC}).

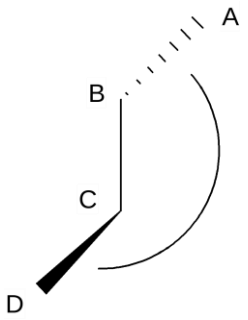
In MM3, this is extended again with a polynomial expression (Equation 5)

$$E(\theta^{ABC}) = 0.021914k^{ABC}(\theta^{ABC} - \theta_0^{ABC})^2[1 - 0.014(\theta^{ABC} - \theta_0^{ABC}) + 5.6 \times 10^{-5}(\theta^{ABC} - \theta_0^{ABC})^2 - 7.0 \times 10^{-7}(\theta^{ABC} - \theta_0^{ABC})^3 + 9.0 \times 10^{-10}(\theta^{ABC} - \theta_0^{ABC})^4] \tag{5}$$

This term has to be calculated across all bond angles within a molecule and these values summed to provide for the contribution of bending to the total energy of the molecule (Equation 6).

$$E_{bend} = \sum_{i}^{bond\ angles} E(\theta^i) \tag{6}$$

In a slightly different manner dihedral angles may be described in terms of a periodic function (a cosine function) in order to calculate the energy due to rotation about bonds (Equation 7). In this equation *n* defines the periodicity of the function.



$$E_{tors}(\omega^{ABCD}) = 0.5 \times V^{ABCD}(1 + \cos(n\omega^{ABCD})) \quad (7)$$

For the MM3 force field, the torsion is defined by a Fourier expansion (Equation 8)

$$E(\omega^{ABCD}) = 0.5V_1^{ABCD}(1 + \cos\omega) + 0.5V_2^{ABCD}(1 + \cos 2\omega) + 0.5V_3^{ABCD}(1 + \cos 3\omega) \quad (8)$$

Again, this energy value must be calculated over all torsions in a molecule to provide for the torsional contribution to the total energy (Equation 9).

$$E_{tors} = \sum_{i}^{all\ torsions} E(\omega^i) \quad (9)$$

Non-bonded terms also contribute to the total energy. Van der Waals interactions are determined between all pairs of non-bonded atoms (Equation 10).There are minor differences with a softer potential in MM3.

$$E_{vdw} = \sum_{i}^{all\ pairs} E(R_{vdw}^{AB}) \quad (10)$$

The second non-bonded term included in molecular mechanics calculations is the electrostatic term. For a single pair of non-bonded atoms, it is calculated using Equation 11:

$$E(R_{elec}^{AB}) = \frac{Q^A Q^B}{\varepsilon R^{AB}} \quad (11)$$

Again there are slight differences in MM3. In Equation 11 Q^A and Q^B are the charges on atoms A and B and R^{AB} is the distance between them. This electrostatic value must be calculated over all pairs of nonbonded atoms to give E_{el}, the electrostatic contribution to the total energy.

Total energy calculated summing up all these terms (Equation 12)

$$E = E_{str} + E_{bend} + E_{tors} + E_{vdw} + E_{el} + E_{cross} \quad (12)$$

2.2.1.2 Geometry Optimization

Geometry optimization or energy minimization is a process that is executed to minimize a conformation of a molecule, through calculation of bond length and angles which lower the total energy of the conformation. Optimization involves adjustments to a system by variation of intramolecular degrees of freedom until a minimum on the potential energy surface is reached. To achieve all this a force field calculation of energy and first derivatives may be used. In this research geometry optimization is performed with energy calculations from the MM3 force field (designed by Allinger and detailed in part 2.2.1) (Patrikeev, 1973; Burkert and Allinger, 1982). The MM3 force field is used extensively in the processing of the virtual chemical library in Chapter 3. Optimizers or minimizers are algorithms that help to compute the new geometrical positions of atoms in a molecule. As shown in Fig 3.09 the optimizer used in this study was the Limited Memory Boryden-Fletcher-Goldfarb-Shanno minimizer (L-BFGS), an algorithm belonging to the set of Quasi-Newton methods (Andrew and Gao 2007; Byrd et al. 1995; Malouf 2002). L-BFGS efficiently locates the energy minimum. In Fig 3.09 the "Final Function Value" is regarded as the final low energy minimum.

2.2.2 Tinker

Tinker (Ponder and Richards) is a molecular modelling package, with a set of programs and procedures for molecular mechanics and dynamics, and various energy based structural manipulation calculations (Paine, 2018). Tinker has many capabilities but of interest to this research is its ability to perform energy minimization and structural optimization using molecular mechanics and its ability to use any of the common parameters such as Amber, Allinger (MM2 and MM3), OPLS, Merck Molecular Force Field, Liam Dang's polarizable model, and the AMOEBA Force Field (Mckay, 2013).

The Tinker suite includes individual programs such as **scan**, **monte**, **sniffer** for performing

global optimization; **dynamic** and **anneal** for dynamics; **minimize**, **optimize**, **newton** for local searches, and many more programs for structural manipulation, parameterization and for free energy calculations and properties (Rackers *et al.*, 2018). Even though all programs from Tinker suit can be communicated with directly through reading standard input, a "key word mechanism" can be used for external commands which can be effectively communicated through an external configuration file in **.key** format. In this **.key** configuration file may contain detailed options, so that these options do not have to be repeatedly input on the command line (Ponder, 2018).

2.3 Molecular docking

Molecular Docking is a method employed to establish the bound confirmations and affinity of receptors which are the macromolecules and ligands which are the smaller molecules (Trott and Olson, 2009). This is achieved through prediction of non-covalent binding of receptors and ligands (Trott and Olson, 2009; Forli *et al.*, 2016) through the aid of various tools such as AutoDock (Österberg *et al*., 2002), AutoDock Vina (Trott and Olson, 2009), Glide (Friesner *et al*., 2004), USCF Dock (Allen et al., 2015) and many more.

These tools make use of various scoring functions which rely on algorithms that calculate (often using a forcefield approach) energy functions and perform various searches for bound conformations, to statistically predict the different binding modes of ligand to receptor protein (regarded as a pose). The poses are the various modes in which the ligand may bind, as a result of different shapes and energy interactions (electrostatic, van der Waals, Coulombic and H bonds). Each pose will give a different binding affinity which can be used to then resolve the best binding pose. A very large negative binding energy is regarded as best when scoring (Huang, 2014).

2.3.1 AutoDock Vina

AutoDock Vina is a molecular docking program which relies on a simple scoring function as well as a rapid gradient-optimization conformational search (Trott and Olson, 2009). In order to carry out a docking procedure using Vina the target docking region on the protein must be

specified, and Vina makes use of a quadrant box to define this particular search area. If an active site is known prior to docking, and if the docking region required is the active site, the quadrant box is defined to cover that region, since this then defines the search space in which all likely ligand binding conformations are explored. This box can be adjusted in size depending on the search area to be explored, with the provision that the box is not too small since this could result in an inadequate number of ligand conformations. By contrast making the box too large may result in a lot of unnecessary poses, although, with extensive conformational search in a large box (blind docking) the best ligand binding site may be located (Feinstein and Brylinski, 2015).

As a requisite for input and output file format AutoDock Vina has been designed to recognize a special type of protein and ligand input file format called pdbqt (Trott and Olson, 2009) this file format contains information like atomic charges, atom type and topological information.

AutoDock Vina allows for the use of a rigid receptor which in turn allows for a reliable search since the size of the conformational space is reduced, in addition this then reduces the computational time needed to score each trial conformation (Forli *et al.*, 2016). The Scoring function for Autodock Vina is based on the combination between knowledge based and empirical approaches (Trott and Olson 2009).

Chapter 3: Generation of the Virtual Libraries

3.1 Graph generation

The main purpose of this study was to generate a virtual library of terpene precursors. In order to do this we used graph-theoretical software to produce a complete set of non-isomorphic graphs. These graphs were screened for suitability and then suitable ones were converted to molecular models. There were complicating factors in this process, particularly with respect to knotted molecules and the computational time required to process these graphs, and the process and complications are detailed in this chapter.

Using Nauty and Traces a complete set of non-isomorphic graphs were created by running selected programs from the suite on the command line. These commands are presented below, the "**geng**" command creates the graphs then the "**showg**" command translates the information into a readable text representation of the graph. The redirection of the output stream to "`number_of_nodesout.txt`" allows for access to the output stream byscripts.

```
geng -c -v number_of_nodes > number_of_nodes.txt

showg < number_of_nodes.txt > number_of_nodesout.txt
```

The aim was to generate graphs of order 7, 8, 9 and 10 thus "number_of_nodes" was set as the order of the graph in each case. The purpose was ultimately to use the nodes to represent carbon atoms (vertices) and the edges to represent bonds (or links). Bear in mind that the well-known norbornane (Gindulyte *et al*., 2018) has 7 carbon atoms (and therefore its carbocation derivatives such as the non-classical 2-norbornyl carbocation all have 7 carbon atoms), while all monoterpenes have 10 carbon atoms. All generated graphs were converted from the encoded graphs in "number_of_nodes.txt" to the more easily human readable graphs in text format in "number_of_nodesout.txt". The total number of graphs generated ranged from 873 (for all graphs of order 7) to 11 million (for graphs of order 10), with the final file size ranging from a few hundred kilobytes to 2 gigabytes as the number of nodes increased from 7 to 10.

The numbers of the complete set of non-isomorphic graphs of each order initially generated are shown in Figure 3.00

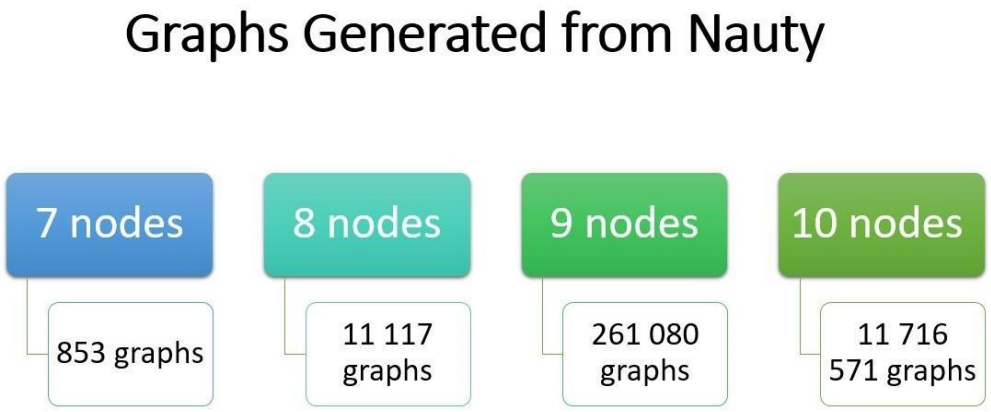


Figure 3.00: Results of graphs of order 7, 8, 9 and 10 created from Nauty and Traces

The 853 graphs of order 7 are simple to explore and manage; however, the 11 716 571 graphs of order 10 required strategies and filters in order to trim to a more manageable number. Each order in this case represents the number of nodes, which also represents the number of carbon atoms in each molecule generated from the graph. Graphs of the same order were all compiled into a single text file upon generation. This results in various text representations of graphs as shown below.

The initial encoding for the graphs is quite difficult to work with (Figure 3.01)

```
135 I???CBozo
136 I???CBo^o
137 I???CBozg
138 I???CBo^g
139 I???CBorw
140 I???CBoZw
141 I???CBo~o
142 I???CBo~g
143 I???CBozw
144 I???CBo^w
145 I???CBo~w
146 I???C@wp_
147 I???C@wx_
```

Figure 3.01: Raw encoding of graphs 135-147 of order 10 as generated by geng

The tool **showg** converts the encoded graph to a more readable format. Figure 3.02 shows the output from conversion of graph 146 (Figure 3.01) to more readable format using **showg**.

```
Graph 146, order 10.
0 : 7 9;
1 : 8 9;
2 : 8;
3 : 8;
4 : 8;
5 : 9;
6 : 9;
7 : 0;
8 : 1 2 3 4;
9 : 0 1 5 6;
```

Figure 3.02: Output of showg for graph 146 of order 10

In Figure 3.02 reading the graph is not difficult. There are 10 nodes (labelled 0 to 9), and the nodes they are connected to are listed next to the label, so node 0 is connected to nodes 7 and 9. Figure 3.03 shows what graph 146 looks like drawn out with its edges and nodes.

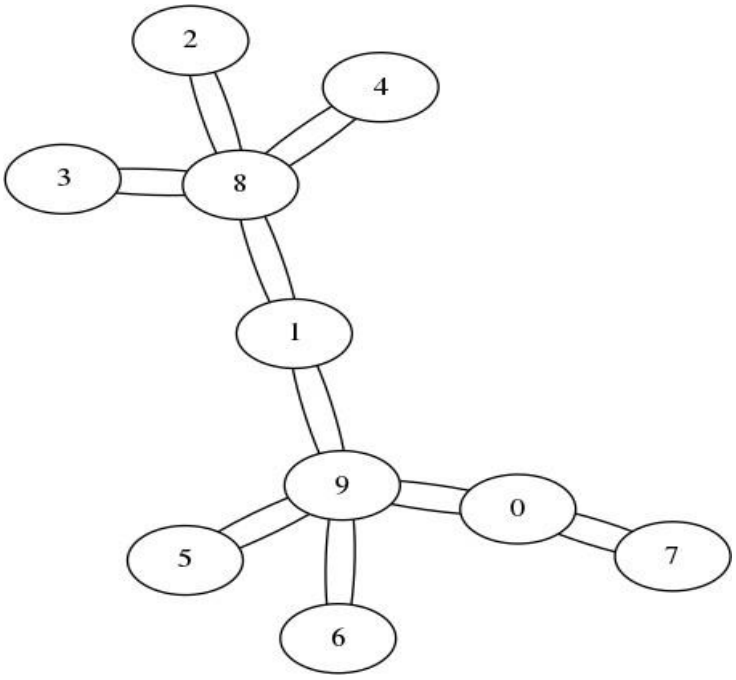


Figure 3.03: Graph 146 constructed in terms of nodes and edges. The edges have been drawn thicker to give the illusion of bonds in this illustration.

Conversion to molecular format will involve setting up bonds and atoms in Tinker format.

Geng creates a complete set of non-isomorphic graphs of a particular order. Each graph text representation shows this information about a molecule, the degree or number of connections per node (vertices) and the relationship between the vertices through edges (links). The degree of each vertex is important, because in the current problem nodes correspond to carbon atoms (with maximum valence of 4) and therefore the degree of any vertex may not be greater than

4. Many graphs in the set will have vertices with degree greater than four, and these correspond to impossible molecules and must be filtered out before further work.

An example of an impossible graph is the first graph created of order 10 by **geng** (Figures 3.04 and 3.05).

```
I??????~w
```

Figure 3.04: First graph of order 10 constructed by geng.

```
Graph 1, order 10.
 0 : 9;
 1 : 9;
 2 : 9;
 3 : 9;
 4 : 9;
 5 : 9;
 6 : 9;
 7 : 9;
 8 : 9;
 9 : 0 1 2 3 4 5 6 7 8;
```

Figure 3.05: Text representation of Graph 1, an impossible graph in terms of molecular representation.

By drawing the graph (Figure 3.06) it may be seen easily that one node (node 9) has order 9 (it is connected to nodes 0-8) while all remaining nodes have order 1. In the context of

chemistry this gives rise to an “impossible molecule” as it violates the valency rules expected per carbon. Normal carbon molecules have a valency of 4, and even though the aim is to generate as many theoretically possible compounds, it is still important to observe bonding rules, thus this deviation will disqualify all molecules failing to adhere to this rule.

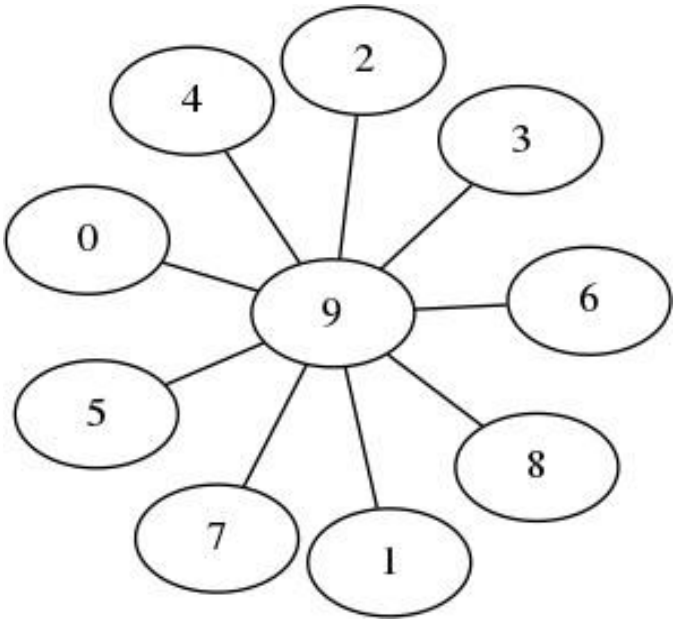


Figure 3.06: A is a constructed representation of Graph 1 showing the impossible connections

The previous graph on the other hand, graph 146 in Figure 3.02 however, has absolutely no issues upon transformation into a 3-dimensional molecule, all nodes demonstrate possible bonding patterns. Connectivity per node does not exceed 4 and so the degree of the feasible graphs are less than or equal to 4. It is through this criteria that the first screening of graphs was conducted. The Python code written to generate the molecules has an initial function written to recognize and discard graphs representing “impossible molecules”.

3.2 Graph processing

In order to process the various text representations of the graphs, the first Python script was written to deal with the data format generated by Nauty, filtering out graphs that are nonsensical in a molecular sense, translating these graphs to a molecular format ready for geometry optimization, and finally calling Tinker to produce minimized molecules.

In developing the script to process the graphs the first aim was to eliminate all graphs for which classical bonding to carbon (maximum of four bonds, due to the valency of carbon) was not represented. This involved eliminating all graphs which had vertices with a degree greater than 4 (see appendix). Screened graphs were then further processed individually by extracting the connectivity information for each node in a graph and writing this to a Tinker ".xyz" file. To this purpose each atom was defined as atom type 1 (aliphatic carbon in the MM3 force field) and this was written to the Tinker ".xyz" file, together with the appropriate connections according to each graph. Figure 3.07 shows an example of one of the Tinker input files. In the input, the first six columns relate to atom number, element, x, y and z coordinates, and atom type respectively. The remaining columns relate to connectivity. Note that the numbering has been altered from 0-9 to 1-10. The atom type is "1" in all cases. The x, y and z coordinates are random since the script incorporates a random coordinate generator supplying each atom with random coordinates in the range of 1.0-5.0 Å (see the appendix for the functions used in python).

```
10
1     C    1.492    2.792    3.243      1      8  10
2     C    1.707    2.204    3.275      1      9  10
3     C    1.586    2.89     2.582      1      9
4     C    1.613    2.92     2.624      1      9
5     C    1.709    1.835    2.81       1      9
6     C    2.226    1.818    2.169      1      10
7     C    1.592    3.715    1.81       1      10
8     C    2.729    1.957    2.928      1      1
9     C    1.959    3.447    3.008      1      2  3  4  5
10    C    2.289    3.672    3.166      1      1  2  6  7
```

Figure 3.07: Tinker input file with random coordinates and atom type 1 included

From the input from Figure 3.07 the fact that there are no hydrogens present in the molecule causes warnings, but exact geometries still result due to the force-field restraints for aliphatic carbon atoms. Examples of these warnings are in Figure 3.08.

Atoms with an Unusual Number of Attached Atoms :

Type	Atom Name	Atom Type	Expected	Found
Valence	1-C	1	4	2
Valence	2-C	1	4	2
Valence	3-C	1	4	1
Valence	4-C	1	4	1
Valence	5-C	1	4	1
Valence	6-C	1	4	1
Valence	7-C	1	4	1
Valence	8-C	1	4	1

Figure 3.08: Tinker warnings regarding valency

In order to obtain the best molecular structure per graph, the molecules generated were optimized, involving structural adjustments that maintained the appropriate connectivity among atoms (defined by the particular graph) whilst altering the angles, torsions and bond lengths until the lowest energy structure was generated (the energies are calculated in terms of the force-field parameters already described). The use of force-fields within Tinker enables the optimization of many structures across a large library, derived from large numbers of graphs in manageable time. (Paine, 2018). In order to perform local minimizations, the "BFGS (Broden-Fletcher-Goldfarb-Shanno) Quasi-Newton Optimization" method was used as implemented in Tinker. Figure 3.09 shows the result of use of this algorithm which, for the structure minimized, achieved geometric convergence in 101 steps, optimizing in Cartesian coordinate space (Rackers *et al.*, 2018). The result of those structural adjustments throughout the energy minimization are monitored Figure 3.09 and the resulting energy of the minimized molecule is recorded as "The final function value". In Figure 3.09 the Final function value is 5.4298 (kcal/mol).

```
Limited Memory BFGS Quasi-Newton Optimization:

QN Iter       F Value        G RMS        F Move X Move Angle FG Call  Comment

      0    0.1462D+08 0.9856D+08                        1
      1    0.6447D+06 0.2469D+07 0.1397D+08  0.1318    0.00    2       Success
      2    0.5703D+06 0.2047D+07 0.7438D+05  0.0033    1.43    3       Success
      3    0.4452D+06 0.1421D+07 0.1252D+06  0.0074   11.28    4       Success
      4    0.2670D+06 0.6759D+06 0.1782D+06  0.0185   17.74    5       Success
      5    0.1724D+06 0.3592D+06 0.9457D+05  0.0199   18.74    6       Success
      6    0.1106D+06 0.1906D+06 0.6184D+05  0.0251   20.29    7       Success
...
    100        5.4298     0.0170      0.0000 0.0003 82.05        103    Success
    101        5.4298     0.0073      0.0000 0.0001 77.19        104    SmallGrad

LBFGS -- Normal Termination due to SmallGrad

Final Function Value :     5.4298
Final RMS Gradient :       0.0073
Final Gradient Norm :      0.0232
```

Figure 3.09: The results of optimization of a system using the BFGS algorithm in Tinker

```
      10
      1  C    2.638283    4.519097    4.172044    1    8    10
      2  C    1.357549    2.594440    3.249577    1    9    10
      3  C    0.495794    2.289409    0.896903    1    9
      4  C   -0.095910    0.676166    2.674808    1    9
      5  C    2.213354    0.721629    1.793635    1    9
      6  C    3.761558    2.980549    2.572965    1   10
      7  C    1.992612    4.542408    1.790391    1   10
      8  C    3.095298    3.699156    5.375600    1    1
      9  C    1.010035    1.588739    2.150524    1    2    3    4    5
     10  C    2.433425    3.638406    2.938553    1    1    2    6    7
```

Figure 3.10: Resultant coordinate file after successful optimization in Tinker

Figure 3.10 shows the Tinker xyz file that is written by Tinker upon completion of the optimization. The coordinates are no longer random (clearly no atoms are overlapping).

Initial work with the order 7 graphs resulted in several molecules which were highly strained and defined to be disjoint when coordinates only were inputted to a visualization (*i.e.* there was more than one part of the molecule not connected by visible bonds upon visualization). To attempt to mediate this problem a perl script using the DiscoveryScript API in Discovery Studio Visualizer was used to screen these systems out (increasing the selection of atoms in a molecule through bonds should select every atom in a connected molecule; where this did not happen, it was clear a highly strained system was present). However, it became evident that the high stains were happening as a result of highly unfavourable conformations occurring sometimes during optimization on the random coordinates. A strategy was developed based on this evidence which ensured that no highly strained molecules resulted from an optimization.

For each individual graph eight separate Tinker ".xyz" input files were created with connectivity matching the graph; in each of these eight files random x, y, and z coordinates were provided for each carbon atom. These eight input files were then subjected to eight separate Tinker molecular mechanics optimizations using the MM3 force field. The MM3 parameters used were as follows. As mentioned previously all atoms were MM3 atom type 1 (Alkane (Csp3)), such that "equilibrium" bond distances, angles and dihedrals (i.e. R_0, θ_0, ω_0) were set to be 1.5247Å, 109.5°, and 180°, including force constants for the

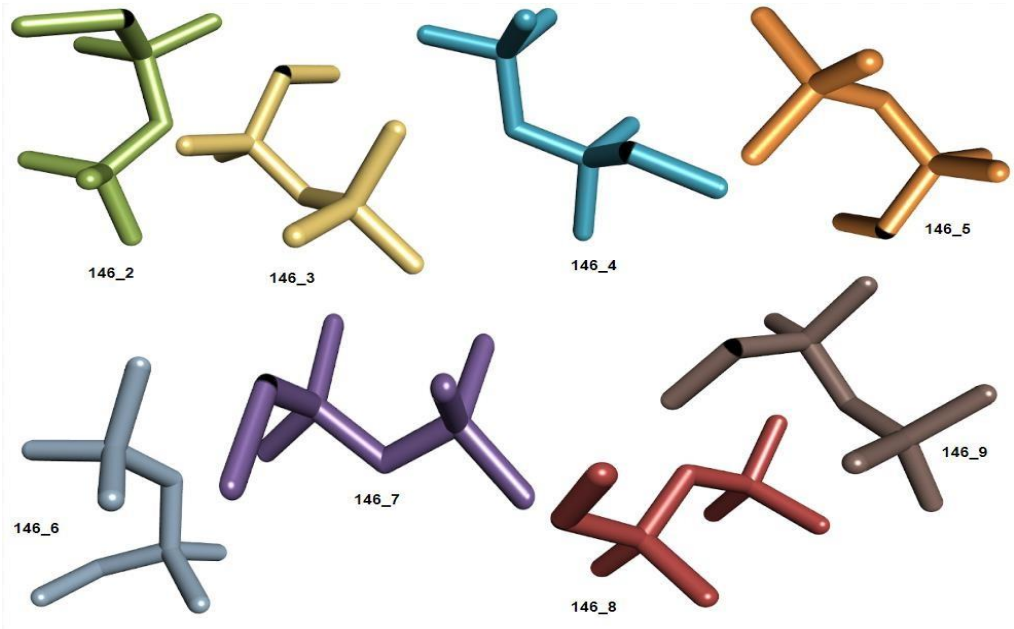


Figure 3.11: Graph 146 (Order 10)

stretches and bends as 4.490 mol.Å $^{-2}$ and 0.670 mol.deg^{-2} respectively. Hydrogen atoms were not present or included in the minimization of the carbon skeleton as already mentioned. All eight structures per graph were optimized (minimized) using the **minimize** program from the Tinker suite Figure 3.11 shows the results of minimization of molecules derived from graph 146 (order 10).

This series of eight optimizations proved to be a successful approach. When working with graphs of order 7, many of the molecules formed were inflexible, and this was reflected in the reproducibility of the optimization approach. For example, Table 3.00 **s**hows the minimization energy from eight sets of random coordinates for Graph 4 (of the order 7 graphs)

Table 3.00: Results of optimization of molecule generated from Graph 4 (order 7)

Graph number 4 repeat number	Minimization Energy (kcal/mol)
2	1.2459
3	1.2459
4	1.2459
5	1.2459
6	1.2459
7	1.2459
8	1.2459
9	1.2460

The variation in optimized energy is insignificant for these, showing high reproducibility of the structure formed. Where flexibility was possible the optimizations showed this in terms of the energy values. Table 3.01 shows the results of optimization of graph 9 (of order 7).

Table 3.01: Results of optimization of molecule generated from Graph 9 (order 7)

Graph number 9 repeat number	Minimization Energy (kcal/mol)
2	3.9739
3	2.3367
4	2.3367
5	2.3367
6	2.3367
7	2.3367
8	3.9739
9	2.3367

In the case of Graph 9 (from graphs of order 7) it may be observed that repeats 2 and 8 are identically a different conformation compared to the other repeats. This is confirmed when viewing the 3-dimensional structures generated by optimization of Graph-9 derived molecules (Figure 3.12)

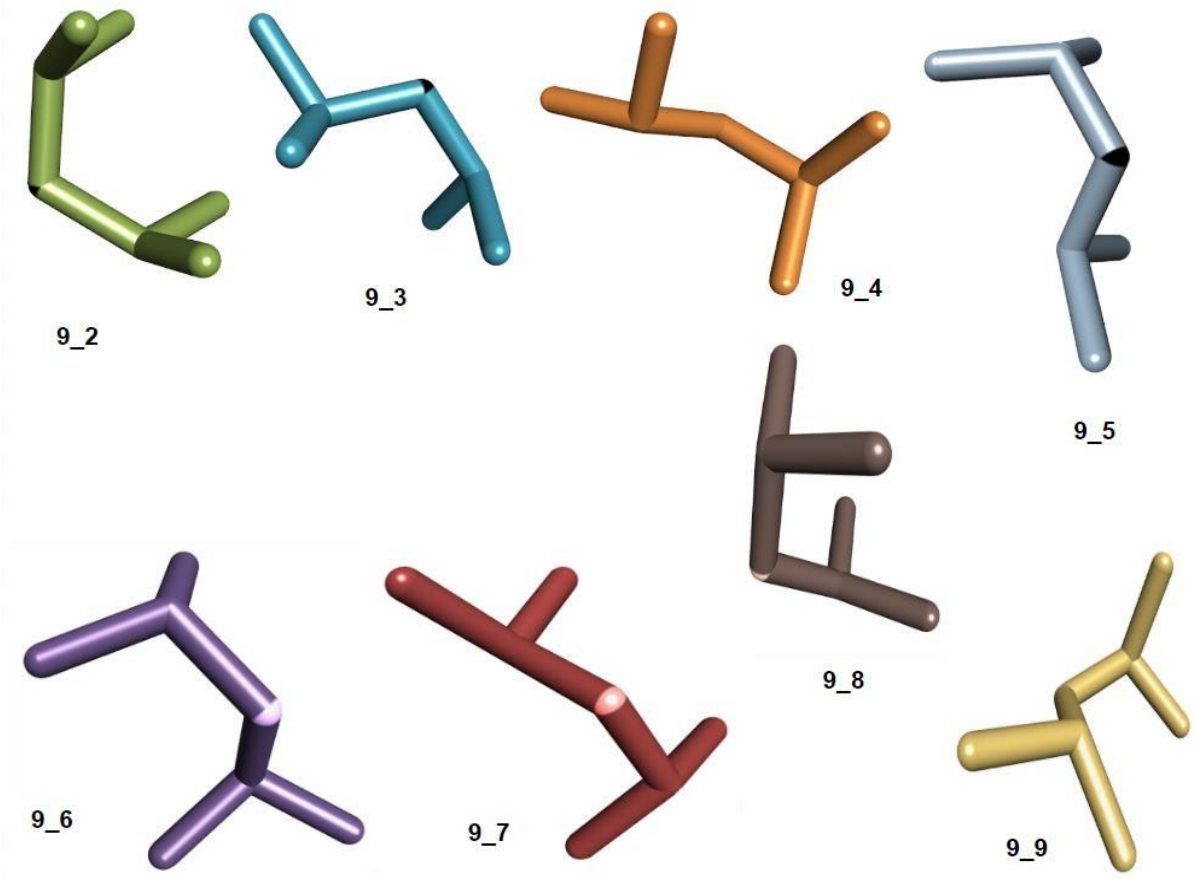


Figure 3.12: Repeats of optimization for Graph 9 (of order 7). Repeats 2 and 8 are identically a symmetrical conformation, different to the lower energy conformation obtained in all other optimizations.

The eight repeats of optimization clearly "saved the day" in some instances. For instance, with Graph 478 (Table 3.02), a problematic conformation, destabilized by 650 kcal/mol is evident.

Table 3.02: Results of optimization of molecule generated from Graph 478 (order 7)

Graph number 478 repeat number	Minimization Energy (kcal/mol)
2	440.0833
3	439.5004
4	440.0833
5	440.0833
6	440.0833
7	439.5004
8	440.0833
9	1110.8715

The 8 optimizations introduced some certainty that unrealistic 3-dimensional structures, such as knotted systems could be accounted for and removed. A knotted system would have just such a higher molecular mechanics energy than reasonable conformations and would give rise to disconnected systems upon import of coordinates into visualization software such as Discovery Studio Visualizer.

Since eight structures were generated from each individual graph, information from minimization included the final geometry and the resultant energy of each of these eight structures. This information was redirected to a log file accessed by a Python script. For graphs of order 7, conversion of minimized structures from Tinker xyz format to pdb format was handled by a system call to OpenBabel:

os.system("babel -ixyz temporary.xyz -h -opdb "+converted_pdb_name)

The recalculation of bond orders that Open Babel performs, may have also contributed to highly strained molecular systems for graphs of order 7.

In later cases this was replaced with functions written in the Python script (see appendix) managing conversion of the minimized Tinker xyz coordinates to pdb file format for each of

these 9 structures. However, the primary reason for handling Tinker xyz to pdb file format conversion within the script, was the time required for Open Babel (babel) to perform a single molecular conversions. This was too long to make treatment of tens of thousands of structures an efficient process, as was required for dealing with graphs of order 10. The custom code meant that this stage of the processing was no longer the time limiting step.

While exploring the minimization of molecules derived from the order 7 graphs, it was observed that the optimization process could lead to two low energy structures (with identical molecular mechanics optimized energy); however, these structures were observed to be differing in chirality. Since molecules of different chirality interact very differently with protein targets it was important that all enantiomers were included within the library. In order to automate this, a calculable measure of chirality was needed in order to identify chiral systems.

3.3 Calculation of Chirality

Wilson *et al.* have defined a chirality index that does not change under rotation and translation but is noted to change sign on mirror images of molecules. For non-chiral systems this metric assumes a value of zero. In this study, the calculation of chirality was conducted according to this metric (Wilson *et al.*) (Figure 3.13)

$$G_{OS} = \frac{4!}{N^4}\frac{1}{3}\left[\sum_{\substack{all\ permutations\ of \\ i,j,k,l=1,\ldots,N}}^{N} w_i w_j w_k w_l \times \frac{[(\mathbf{r}_{ij} \times \mathbf{r}_{kl}) \cdot \mathbf{r}_{il}](\mathbf{r}_{ij} \cdot \mathbf{r}_{jk})(\mathbf{r}_{jk} \cdot \mathbf{r}_{kl})}{(r_{ij} r_{jk} r_{kl})^n r_{il}^m}\right]$$

Figure 3.13: Equation showing calculation for chirality index as derived from Wilson *et al.*

In this equation, w_i, w_j, w_k, w_k, w_l, are values of physical quantities associated with atoms *i* - *l*; in this study, for carbon atoms with mass 12 these values are set to 12. The choice of *m* and *n* are important; choosing as constants *n=2* and *m=1* results in a chirality index which is dimensionless. *N* represents the number of atoms (7 for graphs of order 7, 10 for graphs of order 10) and $\boldsymbol{r}_{ij}$, $\boldsymbol{r}_{jk}$, $\boldsymbol{r}_{kl}$, $\boldsymbol{r}_{il}$ are atom to atom vectors across four atoms *i, j, k* and *l*. The values of r_{ij}, r_{jk}, r_{kl}, r_{il} are size of these vectors, which is simply the interatomic distance. The sum is over all permutations of four atoms from the full set of atoms in the molecule.

Table 3.03 shows that all the repeats of minimization of molecules derived from graph 146 were successfully minimised, and the calculated chirality index calculated is indicated. The

Table 3.03: Table showing minimization energy and chirality indexes for graph 146

Graph number 146 repeat number	Minimization Energy (kcal/mol)	Chirality index
2	5.4306	-110.382
3	6.9762	-2.816
4	5.5006	7.549
5	6.9762	-2.84
6	5.4551	83.669
7	5.4297	109.542
8	5.4375	73.986
9	5.5244	-113.295

lowest minimization energy recorded for Graph 146 molecules is 5.4297 kcal/mol on molecule repeat number 7 and the maximum is 6.9762 which is recorded for both molecule repeat number 3 and 5. The energy difference between the highest minimization energy and the lowest is 1.5465 kcal/mol; this is not as large in magnitude as for example graph 478 which showed a 671.3711 kcal/mol difference between the least and most stable systems (1110.8715 kcal/mol - 439.5004 kcal/mol). This implies that there are small structural differences in the molecules arising from graph 146. The repeats, molecules 3 and 5 both have the same minimization energy; this does not mean they have the same stereochemistry;

however, both have very low values for chirality index, and we would classify these as being non-chiral (our criteria for a non-chiral molecule is where the chirality index CI satisfies -3 < CI < +3). Optimizations 2 and 7 show opposite chirality (see Fig 3.11 previously in this chapter), in this case it means the conformations are mirror images (and that the two optimizations are not necessarily of enantiomers). Although it would appear in this case, simply looking at the energies, that there are two energy values (5.4 and 6.9 kcal/mol) the real diversity in the systems comes from looking at the chirality together with this energy value, in which case it is apparent that there are at least three conformations with energy value 5.4 (and enantiomers in one case) and a single conformation with energy value 6.9 kcal/mol.

For initial work, inclusion of both chirality optimizations in the library provides some redundant entries, but ensures when chiral molecules do exist, both enantiomers are certainly included. Several strategies were followed, the main two focused only on the lowest energy structure or focused on chirals (ensuring the enantiomer was present).

Comparisons were made per graph for best energy of minimization, this was done by selecting molecules with the lowest energy of minimization through a “helper” script that was constructed and run separately to analyse the log file. In early work, molecules with the lowest minimization energies were identified through the result-analysing script, which automated the copying of identified structures to a new folder. However, as observed there were many situations where various molecules generated per graph presented similar minimization energies, there was still the concern with regard to stereo-chemistry amongst molecules of the same graph, thus the chirality test was included and was then included in the updated result analysis script. The updated final script (see appendix) for processing the order 10 graphs, towards the end of the project, merged both the graph processing script and result analysis script into a single tool handling the conversion of graphs to a series of minimized molecules, and identifying both final energy and chirality of all systems. All molecules presenting chirality and the molecules exhibiting no chirality were copied to respective folders, with subfolders being titled according to graph name. Simultaneously, a summary of all minimization energies and chirality index values were included in a log file.

Table 3.04 and Figure 3.14 below are illustrative of the need to both minimize several times

and calculate chirality. Repeats 2 and 6 result in knotted molecules and the energy is

Table 3.04: Results of optimization of molecule generated from Graph 8096 (order 10)

Graph number	Minimization Energy	Chirality Index
8096		
Repeats		
2	659.3845	-16.73
3	166.7268	-17.417
4	166.7268	17.496
5	166.7268	-17.629
6	852.9841	-52.646
7	166.7268	17.377
8	166.7268	-17.709
9	166.7268	17.566

Figure 3.14: Minimizations of molecules from graph 8096 (order 10)

correspondingly very high. Other molecules are acceptable as a tricyclic system with fused 4, 4 and 6 membered rings (a dimethyl tricyclooctane). However, repeats 3, 5 and 8 all

h a v e e nded up with a negative chirality index and are the enantiomer of the molecule repeated in 4, 7 and 9 which has a positive chirality index.

The repeating of molecules certainly helps to overcome the possibility of generating knotted molecules.

Table 3.05: Minimization energy and chirality index for Graph 4858

Graph 4858		
Repeat	Minimization Energy	Chirality Index
2	226.0014	-93.782
3	226.0455	-361.727
4	228.5384	-8.696
5	229.3959	-227.794
6	229.2197	-95.944
7	229.3415	-30.433
8	228.5226	89.924
9	228.1573	53.275

The calculation of chirality index is not ideal. For instance, when examining Graph 4858 (Table 3.05 and Figure 3.15) which defines a very flexible 1,2-*n*-propyl-*t*-butylcyclopropane, both *cis* and *trans* isomers are possible. The *cis*-isomer is created 5 times in the set of 8 optimizations (repeats 4, 5, 6, 7, 8), while the *trans*-isomer is created 3 times (repeats 2, 3, and 9). However, interference from the flexible propyl substituent changes the chirality index significantly. The chirality index is an instantaneous chirality highly dependent on the conformation, and the chirality indexes are therefore not reproducible.

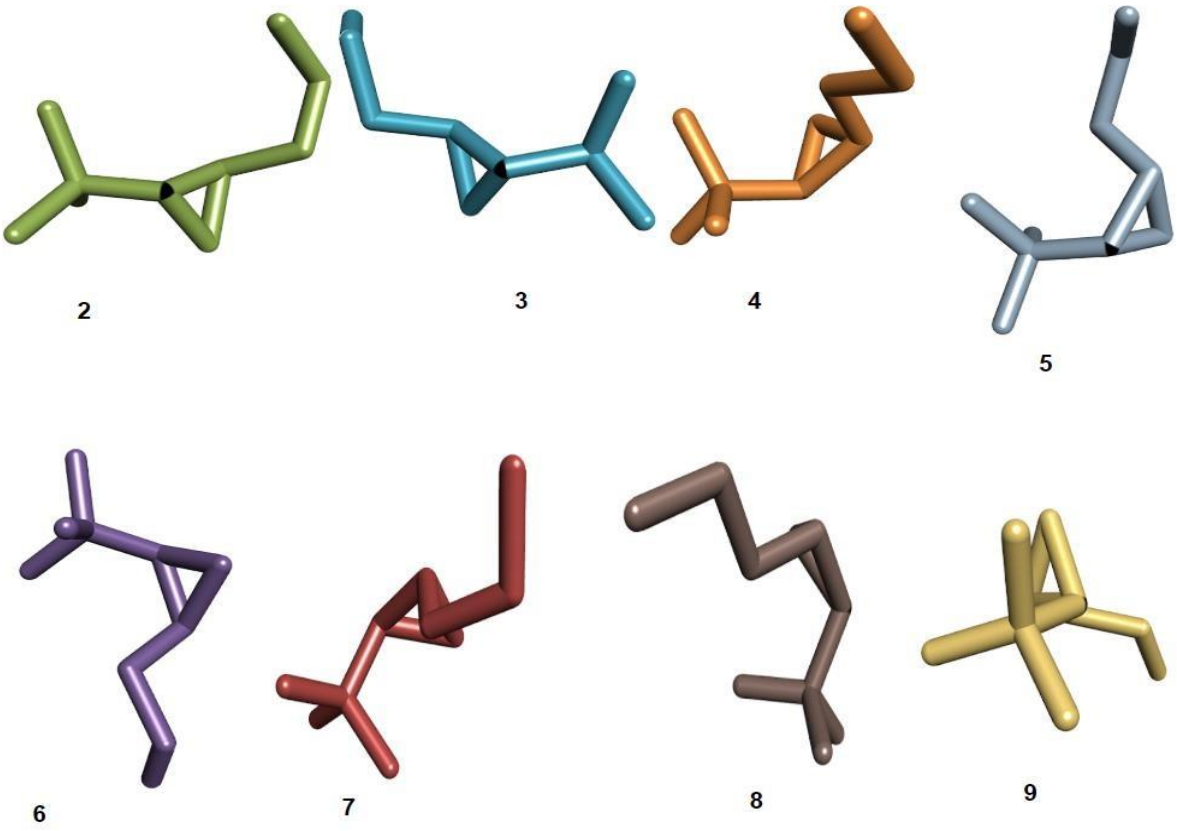


Figure 3.15: Repeat molecules of Graph 4858

One way to ensure diversity in the set was to generate mirror images of systems where chirality was certainly an issue, and this would ensure that with *cis/trans* substitution as for Graph 4858, both enantiomers of each diastereomer would be present. These mirror image pairs were termed “a” and “b” and were included in the chiral molecule dataset.

Knotting produces impossible molecules, and the repeats of minimization certainly reduce the chances of knotted systems ending in the final library. This characterization of impossible molecules comes not from valence information, but from structural information, in which it is clear that bonds are strained to the point that this bonding would not be possible. The minimization energy provides the main clue that this has occurred, and contextually the first comparison is between the eight repeat minimizations for a single system. It is possible to order all isomers in the final library according to energy, in order to further test for less possible (or very high energy systems). Further the library could be further refined using

Quantum Mechanical optimizations (such as semi-empirical AM1 or DFTB+ optimizations) to further filter out such systems.

To give a more general idea of the systems in the library of 10 carbon systems, the final chiral library included aliphatic systems (such as the chiral 3-ethyl-3-methyl-heptane 85517_6_a, Figure 3.16), spiro cyclic systems (such as the methyl spiro[4.4]nonane 87155_2_a, Figure 3.16). Fused cyclic systems were common (including the di-substituted bicyclo[3.2.0]heptane 87330_2_a, Figure 3.17). Not all fused cyclic systems are realistic, and QM-based optimizations will filter these out (the multifused ring system 1740727_3_a, Figure 3.17 is an example of such an unrealistic system).

Figure 3.16: 85517_6_a (left) and 87155_2_a (right)

Figure 3.17: 87330_2_a and 1740727_3_a

Knotted systems have already been mentioned as being unrealistic. However, as system sizes increase, they become more and more plausible. The knotted system 11362438_2_a was the one most plausibly possible (although still extremely strained) of all of the 10 membered knotted systems encountered in this study (Figure 3.18)

Figure 3.18: 11362438_2_a. A knotted molecule. The molecular mechanics bond threading through the loop is not shown.

3.4 Summary of the graph generation

3.4.1 Order 7 Graphs

Figure 3.19 shows a summary of graph creation. After running Nauty and Traces the number of graphs created for order 7 totalled 853. Screening of those graphs ensured that only valid graphs (all which had acceptable valencies) were reproduced as repeats eight per each graph. Graph screening resulted in 353 (Figure 3.19) graphs remaining from the 853 created. As a result, a total of 2824 (Figure 3.19) molecules were generated since 8 repeats were generated 8 molecules per graph out of the 353 valid graphs remaining. When the molecules were checked for chirality 136 graphs included chiral molecules. This also meant that 217 remained as non-chiral systems.

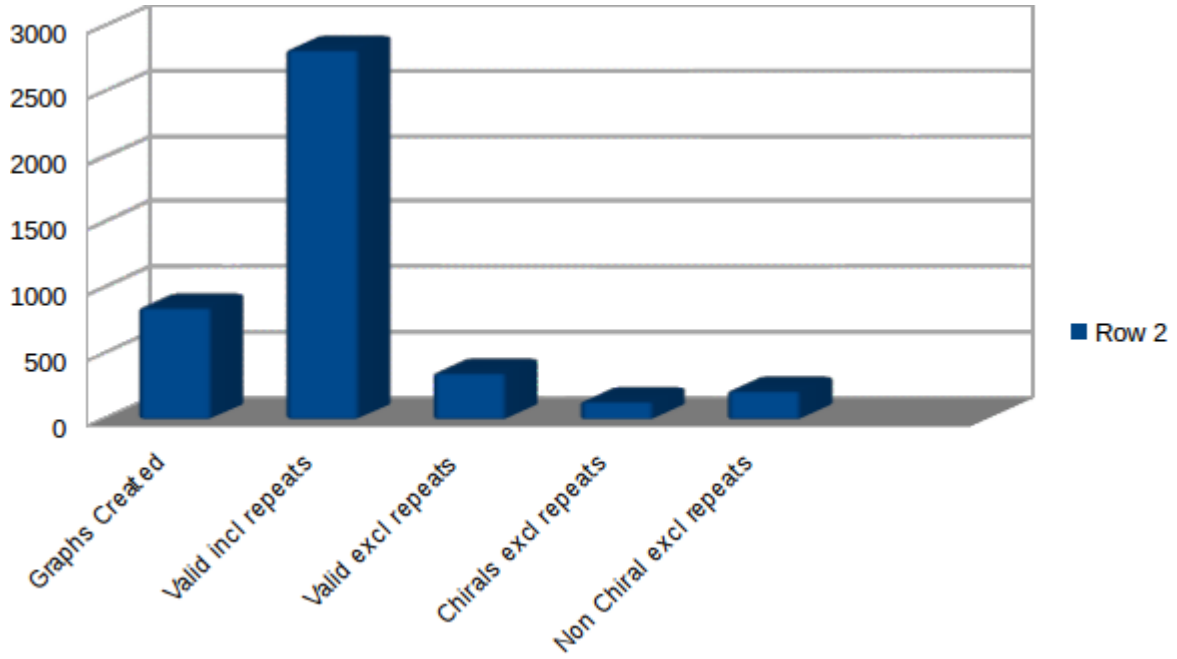


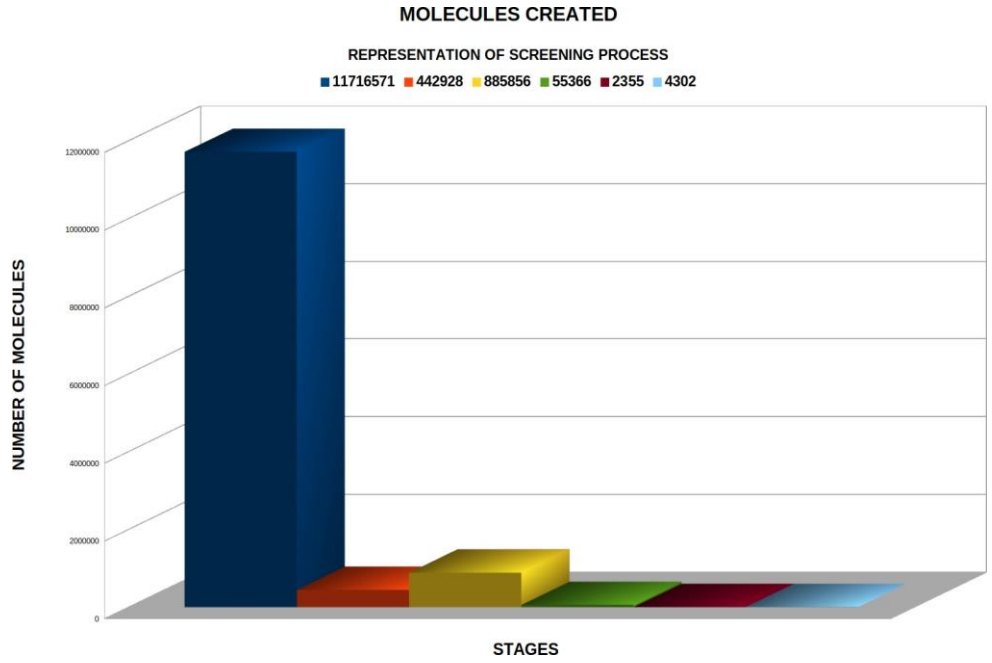
Figure 3.19: Bar graph showing the distribution of order 7 molecules throughout the screening procedure

3.4.2 Order 10 graphs

A summary of order 10 graph creation is shown in Figure 3.20. The total number of graphs created after running Nauty and traces is 11 716 571. Graph processing was done in parts and not all graphs created were processed to molecules due to computational time needed. Thus, of the total of 101161 valid graphs generated, only 55366 were processed due to time constraints. With repeats per graph, a total of 442928 optimizations were performed. Where enantiomers were forced into the set 885856 systems are now available for both "a" and "b" molecules. For the initial virtual screening only the best optimization for each graph screened was used, resulting in the use of 55366 systems in the high throughput virtual screening (Figure 3.20). Therefore, after all filters were applied a total of 55 366 molecules were left for docking. Although doubling up non chiral systems is not ideal, this ensures that if a chiral molecule is present in the set, its enantiomer will also be present in the set.

This number represents all the molecules (considered ligands in this case) which had the lowest minimization energy per graph. After docking on Vina, determination of the molecules with the lowest binding energy was done through a script that analyzed the log file. A total of 2355 (Figure 3.20) were selected after noting the molecules with the lowest binding energy fell between -6.9 kcal mol^{-1} and (including) -7.4 kcal mol^{-1} (mostly -7.0 kcal mol^{-1} and -7.4 kcal mol^{-1}). Since most of the graphs created were not processed an estimation was made to predict the total number of molecules that would score an energy a binding energy of -7.0 kcal mol^{-1} or lower. With the current information we may infer that an estimated 4302 (Figure 3.20) would provide optimal binding.

In this section a complete library of C7 molecules and C10 molecules were generated – in the case of C7 this was exhaustive, but in the case of C10 a significant portion of all possibilities was processed. Further computational time will easily complete this work. Complications have been identified and partially addressed particularly with respect to chirality.



Navy blue: Total number of graphs created

Orange: Valid graphs including repeats

Yellow: Valid graphs including a and b repeats

Light green: Docked ligands

Purple: Best ligands

Sky blue: Estimate outcome of best ligands

Figure 3.20: Bar graph showing distribution of order 10 molecules during the screening procedure.

Chapter 4: High Throughput Virtual Screening

4.1 Methodology

Given the extent of the library generated in Chapter 3, it was of interest to see how this library performed in docking studies with limonene synthase (the choice of limonene synthase is discussed in Chapter 1).

In screening the C10 compounds against a terpene synthase receptor, AutoDock Vina was used to establish the orientation of the ligands in the protein receptor (molecular docking). Limonene synthase from *Mentha spicata* (pdb code 2ONG) (Hyatt et al. 2007) was used as the protein receptor (Figure 4.00).

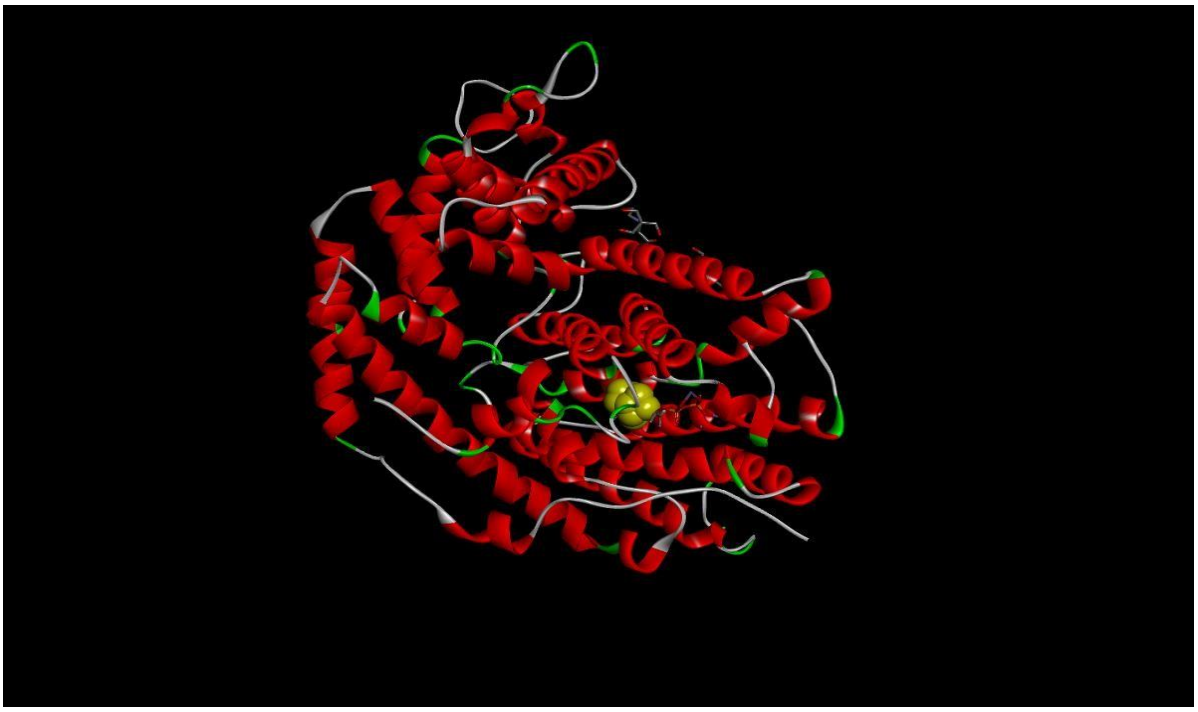


Figure 4.00: The 2ONG with crystal structure ligand (stick) and docked ligand (CPK) overlaid.

4.1.1 Ligand preparation

Ligands used for docking were selected with a preference being for those which simply had the lowest minimization energy amongst the repeats generated per graph and also all were not enantiomers of the repeats (only '_a' molecules were used). After using this criteria for

selection, the library used for docking contained 55 366 molecules. These ligands were copied from their respective library and converted to pdbqt formats files to meet AutoDock Vina file format requirements (Trott and Olson, 2009). The conversion of files from pdb to pdbqt was through an AutoDock Tools script for ligand preparation **prepare_ligand4.py**.

4.1.2 Receptor preparation

Receptor preparation commenced by downloading the required receptor, limonene synthase (pdb code 2ONG) from the Protein Databank website. Ligands and water downloaded with the molecule were manually removed using Discovery Studio Visualizer. The receptor was prepared for molecular docking with an AutoDock Tools script **prepare_receptor4.py**.

4.1.3 Molecular Docking

In nature binding of ligands to active sites involves an "induced fit" or "hand in glove mechanism". In order to mimic such, during a typical docking procedure, the binding of ligands to protein requires for molecular flexibility and which is made possible through either of the structures undergoing different conformations, this though is computationally intensive since there are a lot of rotatable bonds (Mukhopadhyay, 2014). To overcome time losses that may result from conformational searches, AutoDock Vina allows for use of a rigid receptor which in turn allows for acceleration of the search through ligand conformational space. (Feinstein and Brylinski, 2015; Forli *et al.*, 2016).

Since the target site is already known prior to docking, the docking parameters were established for the active site pocket. Molecular docking was performed using the following parameters. The centre of the grid box was set on the xyz coordinates, (23.454, 56.349, 53.915). The grid box was set to size 22 for all dimensions (x, y and z). This was adapted from the default as a standard to avoid the box being too big (to accelerate accurate pose prediction) and not too small (to avoid forced docking). Even though the aim of the docking was to ensure that the docking results be as accurate and reliable as possible, using a Vina exhaustiveness value of 4 was used as a trade-off between docking speed and accuracy; the

number of CPUs was set to one. Given the size of the molecules and the high abundance of inflexible rings in this dataset, this exhaustiveness was a good compromise for dealing with this system.

All ligand poses with lowest binding energy were extracted and compiled into a single file. A small subset of the library (2355 compounds) docked with binding energies less than -7.0 kcal/mol, and this subset was used in one analysis of the binding to the terpene synthase. Another selection was that of 1107 randomly docked molecules with no special preference for binding energy, as a comparison set for visualization (it would have been advantageous to have viewed the entire set, but due to the size of the set of dockings, this was not possible). All ligands were docked on the protein 2ONG and for visualisation purposes Discovery Studio Visualizer was used.

4.2 Results and Discussion

As discussed earlier, 55366 systems were docked using AutoDock Vina. After running AutoDock Vina the docking results showed that 2355 had energies lower than -7 kcal/mol as shown in Figure 3.20 (chapter 3), out of those 6 had the best at -7.4 kcal.mol^{-1}.

All selected ligands from the library were docked on 2ONG as the receptor. According to literature a higher exhaustiveness yields better results, as it gives a chance for many trials before scoring for a recordable pose. However, a very high exhaustiveness means that each individual docking run will require a longer time. Thus, a balance between a reasonable result and a reasonable time was sought. To this end an exhaustiveness of 4 was used to reduce the computational time given the large size of the library. Out of the 9 poses generated per molecule the binding energy range was -5 to -7.4 kcal.mol^{-1}.

Both visualizations showed the same interesting pattern. All the overlaid molecules in the active site, whether the overlay of the lowest binding energy set or the random set, showed that the ligands all bind to a particular region in the active site pocket. Further, the ligands mapped out the shape of the docking site. This overlay was compact with all ligands taking

up minimal space without shifting to reach for longer distances in the active site. Obrien *et al* observed a similar result in their research after docking of the reaction intermediates of terpenes. They showed that all the intermediates were localised in the same region with less movement between different docked structures. For future work it will be interesting to see if it is possible to deconvolute this shape in order to accurately predict the preferred products of the synthase. Since there are many ligands with the same binding energy (for the set of best binding ligands) detailed examination of the 2355 best docked systems, and their structural relationship to each other, may give insight into possible rearrangement mechanisms within the active site.

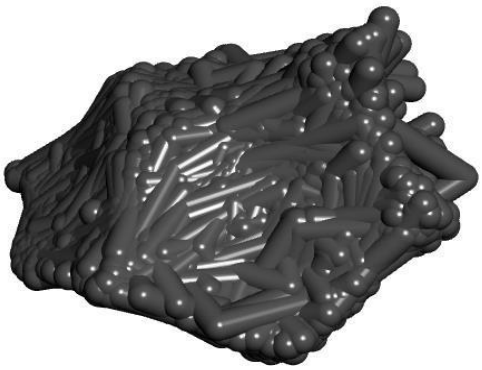


Figure 4.01: Random docked molecules overlay

Figure 4.01 illustrates the shape mapped out by the random ligands. Towards the left, the docking area available is accurately defined, to the right, there is more flexibility in the extent and position of the ligand.

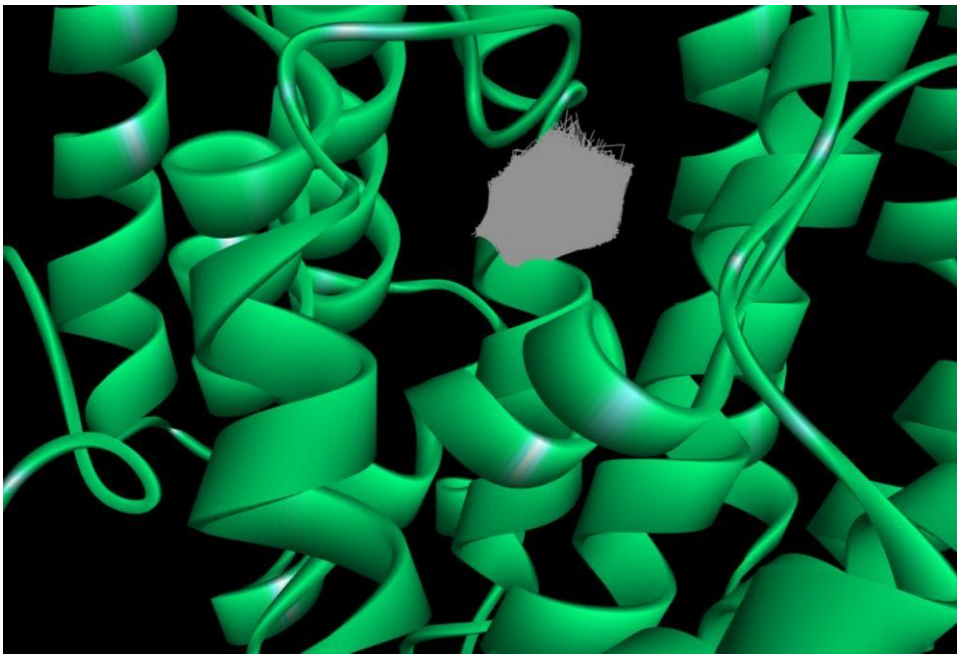


Figure 4.02: 2ONG with lowest binding energy ligands overlaid

Figure 4.02 shows much the same situation, but in this figure the overlay is in the context of the synthase. Again, part of the area spanned is precise at some parts (lower left), but looser in others (top right).

Since there were several ligands with the best binding energy of -7.4 kcal/mol, a random ligand pose was selected to illustrate a single ligand. Figure 4.03 shows a methylated fused tricyclic system with optimal binding to the active site of this enzyme.

Finally, another view of the best binding ligands is provided in Figure 4.04, but in this case the crystal structure ligand, 2-fluorogeranyl diphosphate (FGPP) is also overlaid to give context to the active site.

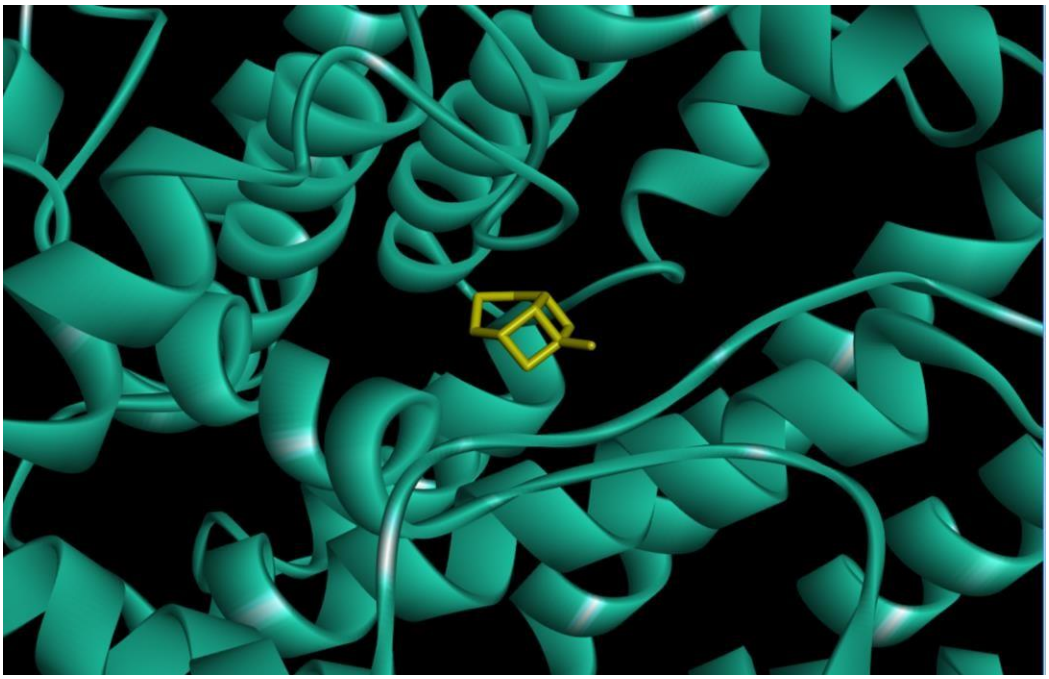


Figure 4.03: 2ONG with a single ligand randomly selected from the ligands with lowest binding energy.

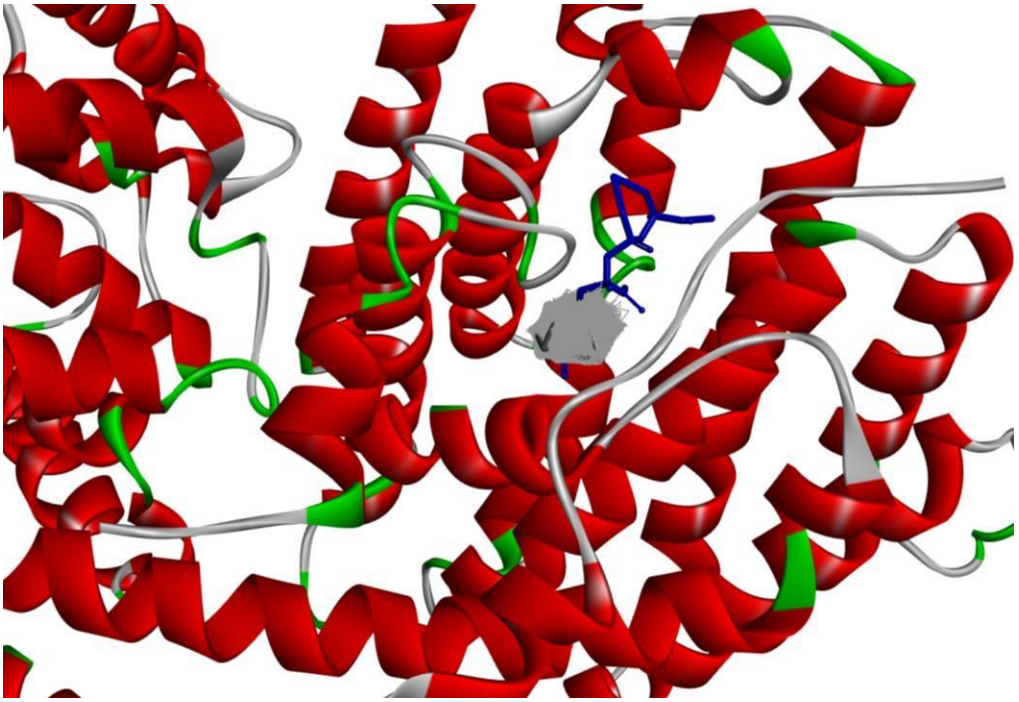


Figure 4.04: 2ONG with an overlay of ligands with lowest binding energy

4.2.1 Ligand interactions

Of course, it is only possible for there to be hydrophobic interactions, since no heteroatom is present on any system in this library. Figure 4.05 shows some of the residues interacting with a dimethylated tricyclic system that shows good binding.

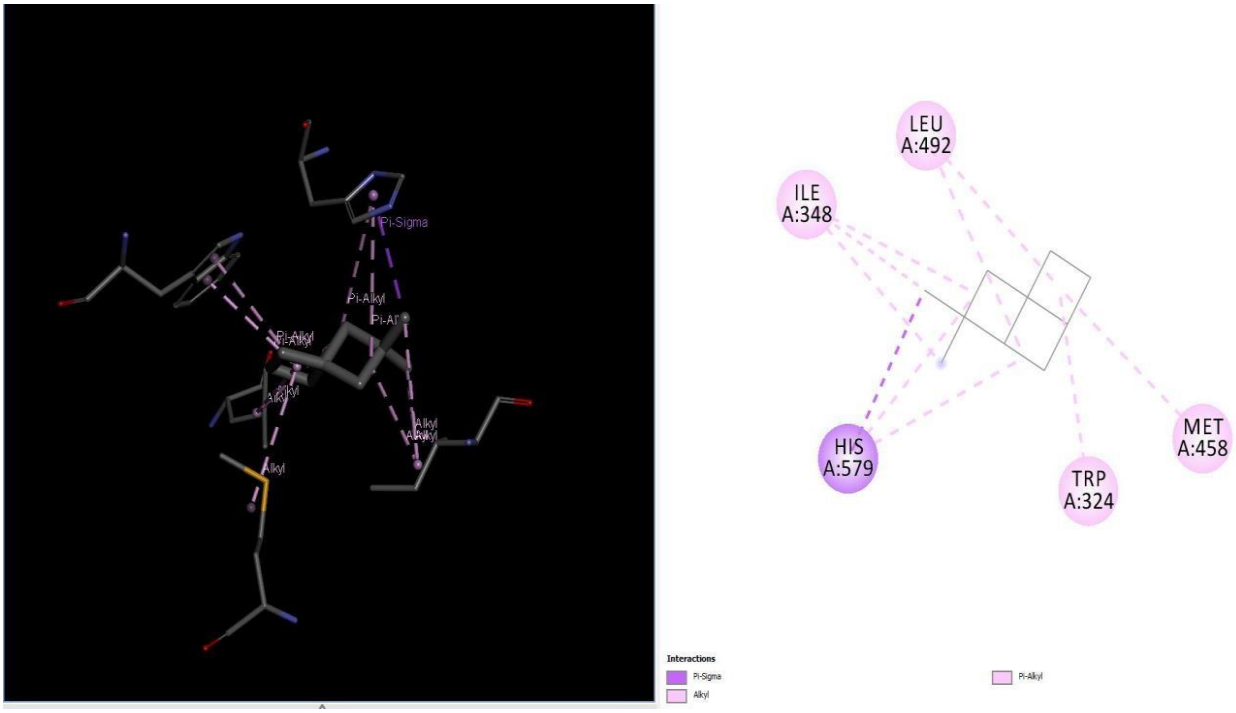


Figure 4.05: Protein ligand interactions between 2ONG receptor and molecule 20588_8

Chapter 5: Conclusion and future work

Exploration of both C7 and C10 hydrocarbon chemical product space has been attempted using a graph theoretic approach. We have successfully generated the compounds of choice by translating the graphs (in text representation from Nauty) into chemical data that has allowed for the optimization and visualization of these structures. In this case C7 and C10 molecules were generated, the initial number of graphs in C7 were 873; after screening there were 373 compounds. C10 graphs initially generated were 11 716 571 and out of those 101161 were valid.

After performing docking in the active site of limonene synthase, 2355 compounds of 55366 docked compounds provided the lowest binding energies of between -7.0 and -7.4 $kcal.mol^{-1}$. These were now compounds of interest to us in terms of the mechanism of terpene synthesis by limonene synthase. It was noted that the overlaid structures all the compounds either best or randomly selected, merged in the same region of the active site, this similar convergence was also demonstrated by O'Brian *et al*. In addition a closer view of compounds docked in the active site, whether randomly selected or best, provided a visual representation of the space into which hydrocarbons will feasibly dock.

The current work is not quite at the point where work may proceed in the prediction of rearrangement mechanisms within the active site. Future work will involve completing the library, extending to cations and also performing of molecular docking with different terpene synthases. QM/MM techniques need to be used in the evaluation of the best binding structures and also in the determination of reactivity. Future work also seeks to encompass libraries of terpenoids, of compounds such as the ketones camphor, verbenone and fenchone.

References

Ajikumar, Parayil Kumaran, Keith Tyo, Simon Carlsen, Oliver Mucha, Too Heng Phon, and Gregory Stephanopoulos. 2008. “Reviews Terpenoids : Opportunities for Biosynthesis of Natural Product Drugs Using Engineered Microorganisms.” 5(2):167–90.

Al-Alawi, Mohammad S. 2014. “Efficacy of Essential Oils from Medicinal Plants in Control of the Hairy Rose Beetle, Tropinota Squalida (Scopoli) and Their Comparative Toxicity to the Honey Bee, Apis Mellifera L.” *American Journal of Agricultural and Biological Science* 9(3):284–88.

Allen, William J., Trent E. Balius, Sudipto Mukherjee, Scott R. Brozell, Demetri T. Moustakas, P. Therese Lang, David A. Case, Irwin D. Kuntz, and Robert C. Rizzo. 2015. “DOCK 6: Impact of New Features and Current Docking Performance.” *Journal of Computational Chemistry* 36(15):1132–56.

Allinger, Norman L., Young H. Yuh, and Jenn Huei Lii. 1989. “Molecular Mechanics. The MM3 Force Field for Hydrocarbons. 1.” *Journal of the American Chemical Society* 111(23):8551–66.

Andrew, Galen and Jianfeng Gao. 2007. “Scalable Training of L1-Regularized Log-Linear Models.” Pp. 33–40 in *Proceedings of the 24th International Conference on Machine Learning, ICML ’07*. New York, NY, USA: ACM.

ARNAUD, CELIA HENRY. 2011. “SPEEDING UP MASS SPEC IMAGING.” *Chemical & Engineering News Archive* 89(51):38–39.

Ashby, N. and A. Edward. 1990. “Elucidation of the Deficiency in Two Yeast Coenzyme Q Mutants.” 265(22):13157–64.

Bang-Jensen, Jørgen and Gutin, Gregory Z. 2007. “Digraphs: Theory, Algorithms and Applications.” *Software Testing, Verification and Reliability* (August):59–60.

Bastos, Vasco P. D., Antoniella S. Gomes, Francisco J. B. Lima, Teresinha S. Brito, Pedro M. G. Soares, João P. M. Pinho, Claudijane S. Silva, Armênio A. Santos, Marcellus H. L. P. Souza, and Pedro J. C. Magalhães. 2011. “Inhaled 1,8-Cineole Reduces Inflammatory Parameters in Airways of Ovalbumin-Challenged Guinea Pigs.” *Basic & Clinical Pharmacology & Toxicology* 108(1):34–39.

Beran, Franziska, Peter Rahfeld, Katrin Luck, Raimund Nagel, Heiko Vogel, Natalie Wielsch, Sandra Irmisch, Srinivasan Ramasamy, Jonathan Gershenzon, David G. Heckel, and Tobias G. Köllner. 2016. “Novel Family of Terpene Synthases Evolved from *Trans* -Isoprenyl Diphosphate Synthases in a Flea Beetle.” *Proceedings of the National Academy of Sciences* 113(11):2922–27.

Bicas, Juliano Lemos, Ana Paula Dionísio, and Gláucia Maria Pastore. 2009. “Bio-Oxidation of Terpenes: An Approach for the Flavor Industry.” *Chemical Reviews* 109(9):4518–31.

Bloch, Konrad. 1965. “The Biological Synthesis of Cholesterol.” *Science* 150(3692):19 LP – 28.

Bochar, Daniel A., Jona. Freisen, Cynthia V. Stauffacher, and Victor W. Rodwell. 1999. “Biosynthesis of Mevalonic Acid from Acetyl-CoA.” *Comprehensive Natural Products Chemistry* 15–44.

Bohlmann, J., G. Meyer-Gauen, and R. Croteau. 1998. “Plant Terpenoid Synthases: Molecular Biology and Phylogenetic Analysis.” *Proceedings of the National Academy of Sciences* 95(8):4126–33.

Bojin, Mihaela D. and Dean J. Tantillo. 2006. “Nonclassical Carbocations as C-H Hydrogen Bond Donors.” *Journal of Physical Chemistry A* 110(14):4810–16.

Boucher, Yan and W. Ford Doolittle. 2000. “MicroReview The Role of Lateral Gene Transfer in the Evolution of Isoprenoid Biosynthesis Pathways.” 37.

Breitmaier, Eberhard. 2006. *Terpenes : Flavors, Fragrances, Pharmaca, Pheromones*. WILEY-VCH.

Burkert, U. and N. L. Allinger. 1982. “Molecular Mechanics.” *Accurate Molecular Structures Their Determination and Importance* 117:1–10.

Byrd, R., P. Lu, J. Nocedal, and C. Zhu. 1995. “A Limited Memory Algorithm for Bound Constrained Optimization.” *SIAM Journal on Scientific Computing* 16(5):1190–1208.

Cane, David E. 1990. “Enzymatic Formation of Sesquiterpenes.” *Chemical Reviews* 90(7):1089–1103.

Chang, Wei-chen, Heng Song, Hung-wen Liu, and Pinghua Liu. 2013. “Current Development in Isoprenoid Precursor Biosynthesis and Regulation.” *Current Opinion in Chemical Biology* 17(4):571–79.

Chen, Feng, Dorothea Tholl, Jörg Bohlmann, and Eran Pichersky. 2011. “The Family of Terpene Synthases in Plants: A Mid-Size Family of Genes for Specialized Metabolism That Is Highly Diversified throughout the Kingdom.” *The Plant Journal* 66(1):212–29.

Chen, Xinlu, Tobias G. Köllner, Qidong Jia, Ayla Norris, Balaji Santhanam, Patrick Rabe, Jeroen S. Dickschat, Gad Shaulsky, Jonathan Gershenzon, and Feng Chen. 2016. “Terpene Synthase Genes in Eukaryotes beyond Plants and Fungi: Occurrence in Social Amoebae.” *Proceedings of the National Academy of Sciences* 113(43):12132–37.

Christianson, David W. 2006. “Structural Biology and Chemistry of the Terpenoid Cyclases.” *Chemical Reviews* 106(8):3412–42.

Christianson, David W. 2017. “Structural and Chemical Biology of Terpenoid Cyclases.” *Chemical Reviews* 117(17):11570–648.

Croteau, Rodney. 1987. “Biosynthesis and Catabolism of Monoterpenoids.” *Chemical Reviews* 87(5):929–54.

Degenhardt, Jörg, Tobias G. Köllner, and Jonathan Gershenzon. 2009. “Monoterpene and Sesquiterpene Synthases and the Origin of Terpene Skeletal Diversity in Plants.” *Phytochemistry* 70(15–16):1621–37.

Dewick, Paul M. 2002. “The Biosynthesis of C5-C25 Terpenoid Compounds.” *Natural Product Reports* 19(2):181–222.

Dicke, Marcel. 1999. *Are Herbivore-Induced Plant Volatiles Reliable Indicators of Herbivore Identity to Foraging Carnivorous Arthropods? Entomol Exp Appl*. Vol. 91.

Eisenreich, W., A. Bacher, D. Arigoni, and F. Rohdich. 2004. “Biosynthesis of Isoprenoids via the Non-Mevalonate Pathway.” *Cellular and Molecular Life Sciences* 61(12):1401–26.

Feinstein, Wei P. and Michal Brylinski. 2015. “Calculating an Optimal Box Size for Ligand Docking and Virtual Screening against Experimental and Predicted Binding Pockets.” *Journal of Cheminformatics* 7(1).

Filipe, Alexandra, João C. R. Cardoso, Graça Miguel, Liliana Anjos, Helena Trindade, Ana Cristina Figueiredo, José Barroso, Deborah M. Power, and Natália T. Marques. 2017. “Molecular Cloning and Functional Characterization of a Monoterpene Synthase Isolated from the Aromatic Wild Shrub Thymus Albicans.” *Journal of Plant Physiology* 218:35–44.

Forli, Stefano, Ruth Huey, Michael E. Pique, Michel Sanner, David S. Goodsell, and J. Arthur. 2016. “HHS Public Access.” 11(5):905–19.

Friesner, Richard A., Jay L. Banks, Robert B. Murphy, Thomas A. Halgren, Jasna J. Klicic, Daniel T. Mainz, Matthew P. Repasky, Eric H. Knoll, Mee Shelley, Jason K. Perry, David E. Shaw, Perry Francis, and Peter S. Shenkin. 2004. “Glide: A New Approach for Rapid, Accurate Docking and Scoring. 1. Method and Assessment of Docking Accuracy.” *Journal of Medicinal Chemistry* 47(7):1739–49.

Gershenzon, Jonathan and Natalia Dudareva. 2007. “The Function of Terpene Natural Products in the Natural World.” *Nature Chemical Biology* 3(7):408–14.

Gindulyte, Asta, Benjamin A. Shoemaker, Bo Yu, Jia He, Jian Zhang, Jie Chen, Leonid Zaslavsky, Paul A. Thiessen, Qingliang Li, Siqian He, Sunghwan Kim, Tiejun Cheng, and Evan E. Bolton. 2018. “PubChem 2019 Update: Improved Access to Chemical Data.” *Nucleic Acids Research* 47(D1):D1102–9.

Hamberger, B., T. Ohnishi, B. Hamberger, A. Seguin, and J. Bohlmann. 2011. “Evolution of Diterpene Metabolism: Sitka Spruce CYP720B4 Catalyzes Multiple Oxidations in Resin Acid Biosynthesis of Conifer Defense against Insects.” *Plant Physiology* 157(4):1677–95.

Herrera, Jimena María, María Paula Zunino, Jose Sebastian Dambolena, Romina Paola Pizzolitto, Nicolás Alberto Gañan, Enrique Ivan Lucini, and Julio Alberto Zygadlo. 2015. “Terpene Ketones as Natural Insecticides against Sitophilus Zeamais.” *Industrial Crops and Products* 70:435–42.

Hong, Young J. and Dean J. Tantillo. 2010. “Quantum Chemical Dissection of the Classic

Terpinyl/Pinyl/Bornyl/Camphyl Cation Conundrum - The Role of Pyrophosphate in Manipulating Pathways to Monoterpenes." *Organic and Biomolecular Chemistry* 8(20):4589–4600.

Hong, Young J. and Dean J. Tantillo. 2011. "The Taxadiene-Forming Carbocation Cascade." *Journal of the American Chemical Society* 133(45):18249–56.

Hostettmann, Kurt, Shilin Chen, Andrew Marston, and Hermann Stuppner. 2014. *Handbook of Chemical and Biological Plant Analytical Methods, 3 Volume Set*. John Wiley & Sons.

Huang, Sheng-You. 2014. "Search Strategies and Evaluation in Protein–Protein Docking: Principles, Advances and Challenges." *Drug Discovery Today* 19(8):1081–96.

Hyatt, D. C., B. Youn, Y. Zhao, B. Santhamma, R. M. Coates, R. B. Croteau, and C. Kang. 2007. "Structure of Limonene Synthase, a Simple Model for Terpenoid Cyclase Catalysis." *Proceedings of the National Academy of Sciences* 104(13):5360–65.

Iwata, Takeshi, Yoshiki Okeda, and Yoji Hori. 2004. "Process for Producing Isopulegol." *United States PatentS Patent* 6(12):2–9.

Jackson, Ron S. 2008. *Wine Science : Principles and Applications*. Elsevier/Academic Press.

Jia, Qidong, Tobias G. Köllner, Jonathan Gershenzon, and Feng Chen. 2018. "MTPSLs: New Terpene Synthases in Nonseed Plants." *Trends in Plant Science* 23(2):121–28.

Jonathan L. Gross, Jay Yellen. 1999. *Graph Theory and Its Applications, Second Edition*. Second Edi. New York: CRC Press.

Kandi, Sabitha, Vikram Godishala, Pragna Rao, and K. V Ramana. 2015. "Biomedical Significance of Terpenes : An Insight." *Science & Education Publishing* 3(1):8–10.

Kearnes, Steven, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. 2016. "Molecular Graph Convolutions: Moving beyond Fingerprints." *Journal of Computer-Aided Molecular Design* 30(8):595–608.

Kessler, Andre and Ian Baldwin. 2002. *Kessler, A. & Baldwin, I. T. Plant Responses to Insect Herbivory: The Emerging Molecular Analysis. Annu. Rev. Plant Biol. 53, 299-328*. Vol. 53.

Klayman, D. L. 1985. "Qinghaosu (Artemisinin): An Antimalarial Drug from China." *Science* 228(4703):1049 LP – 1055.

Köllner, Tobias G., Jonathan Gershenzon, and Jörg Degenhardt. 2009. "Molecular and Biochemical Evolution of Maize Terpene Synthase 10, an Enzyme of Indirect Defense." *Phytochemistry* 70(9):1139–45.

Köllner, Tobias G., Paul E. O'Maille, Nathalie Gatto, Wilhelm Boland, and Jörg Degenhardt. 2006. "Two Pockets in the Active Site of Maize Sesquiterpene Synthase TPS4 Carry out Sequential Parts of the Reaction Scheme Resulting in Multiple Products." *Archives of Biochemistry and Biophysics* 448(1–

2):83–92.

Kutyna, Dariusz R. and Anthony R. Borneman. 2018. “Heterologous Production of Flavour and Aroma Compounds in Saccharomyces Cerevisiae.” *Genes* 9(7).

Lamberti, Vincent E., Lloyd D. Fosdick, Elizabeth R. Jessup, and Carolyn J. C. Schauble. 2002. “A Hands-On Introduction to Molecular Dynamics.” *Journal of Chemical Education* 79(5):601.

Leffingwell, J. C. and R. E. Shackelford. 1974. “Leavo-Menthol – Syntheses and Organoleptic Properties.” *Cosmetics and Perfumery* 89(6):69–78.

Lipkus, Alan H., Qiong Yuan, Karen A. Lucas, Susan A. Funk, William F. Bartelt, Roger J. Schenck, and Anthony J. Trippe. 2008. “Structural Diversity of Organic Chemistry. A Scaffold Analysis of the CAS Registry.” *The Journal of Organic Chemistry* 73(12):4443–51.

Little, Dawn B. and Rodney B. Croteau. 2002. “Alteration of Product Formation by Directed Mutagenesis and Truncation of the Multiple-Product Sesquiterpene Synthases δ-Selinene Synthase and γ-Humulene Synthase.” *Archives of Biochemistry and Biophysics* 402(1):120–35.

Malouf, Robert. 2002. “A Comparison of Algorithms for Maximum Entropy Parameter Estimation.” Pp. 1–7 in *Proceedings of the 6th Conference on Natural Language Learning - Volume 20, COLING-02*. Stroudsburg, PA, USA: Association for Computational Linguistics.

Marmulla, Robert and Jens Harder. 2014. “Microbial Monoterpene Transformations-a Review.” *Frontiers in Microbiology* 5(JULY):1–14.

Martin, D. M., O. Toub, A. Chiang, B. C. Lo, S. Ohse, S. T. Lund, and J. Bohlmann. 2009. “The Bouquet of Grapevine (Vitis Vinifera L. Cv. Cabernet Sauvignon) Flowers Arises from the Biosynthesis of Sesquiterpene Volatiles in Pollen Grains.” *Proceedings of the National Academy of Sciences* 106(17):7245–50.

McGarvey, D. J. and R. Croteau. 1995. “Terpenoid Metabolism.” *The Plant Cell* 7(7):1015–26.

Mckay, Brendan D. 2013. “Nauty and Traces User ’ s Guide (Version 2 . 5) How to Use This Guide.” 1–94.

McKay, Brendan D. and Adolfo Piperno. 2014. “Practical Graph Isomorphism, II.” *Journal of Symbolic Computation* 60(January 2013):94–112.

Miguel, Maria Graça. 2010. “Antioxidant and Anti-Inflammatory Activities of Essential Oils: A Short Review.” *Molecules* 15(12):9252–87.

Miziorko, H. 2011. “Enzymes of the Mevalonate Pathway of Isopreniod Biosynthesis.” *Archives of Biochemistry and Biophysics* 505(2):131–43.

Morehouse, Benjamin R., Ramasamy P. Kumar, Jason O. Matos, Sarah Naomi Olsen, Sonya Entova, and Daniel D. Oprian. 2017. “Functional and Structural Characterization of a (+)-Limonene Synthase

from Citrus Sinensis." *Biochemistry* 56(12):1706–15.

Mukhopadhyay, Mayukh. 2014. *A Brief Survey on Bio Inspired Optimization Algorithms for Molecular Docking*. Vol. 7.

Nykamp DQ, "Graph definition." From *Math Insight*. http://mathinsight.org accessed 31 January 2019 09:22

O'Brien, Terrence E., Steven J. Bertolani, Yue Zhang, Justin B. Siegel, and Dean J. Tantillo. 2018. "Predicting Productive Binding Modes for Substrates and Carbocation Intermediates in Terpene Synthases - Bornyl Diphosphate Synthase As a Representative Case." *ACS Catalysis* 8(4):3322–30.

Oldfield, Eric and Fu-yang Lin. 2012. "Terpene Bioynthesis: Modularity Rules." *Angewandte Chemie International Edition* 51(5):1124–37.

Österberg, Fredrik, Garrett M. Morris, Michel F. Sanner, Arthur J. Olson, and David S. Goodsell. 2002. "Automated Docking to Multiple Target Structures: Incorporation of Protein Mobility and Structural Water Heterogeneity in Autodock." *Proteins: Structure, Function and Genetics* 46(1):34–40.

Pagadala, Nataraj S., Khajamohiddin Syed, and Jack Tuszynski. 2017. "Software for Molecular Docking: A Review." *Biophysical Reviews* 9(2):91–102.

Paine, R. T. 2018. "Tinker." *Design* (June).

Patrikeev, G. A. 1973. "Macromolecular Mechanics." *Polymer Mechanics* 7(2):183–92.

Pichersky, Eran and Jonathan Gershenzon. 1934. "The Formation and Function of Plant Volatiles : Perfumes for Pollinator Attraction and Defense." 237–43.

Ponder, Jay William. 2018. "Software Tools for Molecular Design Version .0 2FWREHU 20." (February 2018):0–219.

Rackers, Joshua A., Zhi Wang, Chao Lu, Marie L. Laury, Louis Lagardère, Michael J. Schnieders, Jean Philip Piquemal, Pengyu Ren, and Jay W. Ponder. 2018. "Tinker 8: Software Tools for Molecular Design." *Journal of Chemical Theory and Computation* 14(10):5273–89.

Rai, Mahendra and Kateryna Kon. 2013. *Fighting Multidrug Resistance with Herbal Extracts, Essential Oils and Their Components*. Academic Press.

Rocha Caldas, Germana Freire, Alisson Rodrigo da Silva Oliveira, Alice Valença Araújo, Simone Sette Lopes Lafayette, Giwellington Silva Albuquerque, Jacinto da Costa Silva-Neto, João Henrique Costa-Silva, Fabiano Ferreira, José Galberto Martins da Costa, and Almir Gonçalves Wanderley. 2015. "Gastroprotective Mechanisms of the Monoterpene 1,8-Cineole (Eucalyptol)" edited by J. Gracia-Sancho. *PLOS ONE* 10(8):e0134558.

Rohmer, M., M. Knani, P. Simonin, B. Sutter, and H. Sahm. 1993. "Isoprenoid Biosynthesis in Bacteria: A Novel Pathway for the Early Steps Leading to Isopentenyl Diphosphate." *The Biochemical Journal*

295 (Pt 2(Pt 2):517–24.

Ruddigkeit, Lars, Ruud Van Deursen, Lorenz C. Blum, and Jean Louis Reymond. 2012. “Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17.” *Journal of Chemical Information and Modeling* 52(11):2864–75.

Sacchettini, James C. and C. Dale Poulter. 1997. “Creating Isoprenoid Diversity.” *Science* 277(5333):1788 LP – 1789.

Savage, T. J., M. W. Hatch, and R. Croteau. 1994. “Monoterpene Synthases of Pinus Contorta and Related Conifers. A New Class of Terpenoid Cyclase.” *Journal of Biological Chemistry* 269(6):4012–20.

Steele, Christopher L., John Crock, Jörg Bohlmann, and Rodney Croteau. 1998. “Sesquiterpene Synthases from Grand Fir (Abies Grandis).” *The Journal of Biological Chemistry* 273(4):2078–89.

STINSON, STEPHEN C. 1996. “SCIENCE/TECHNOLOGY.” *Chemical & Engineering News Archive* 74(29):35-45,54,57,61.

Thurnher, Martin, Oliver Nussbaumer, and Georg Gruenbacher. 2012. “Novel Aspects of Mevalonate Pathway Inhibitors as Antitumor Agents.” *Clinical Cancer Research* 18(13):3524–31.

Tian, Boxue, C. Dale Poulter, and Matthew P. Jacobson. 2016. “Defining the Product Chemical Space of Monoterpenoid Synthases.” *PLoS Computational Biology* 12(8):1–13.

Tian, Lijun, Chaoqun Liu, and Jianquan Xie. 2012. “A Partition Method for Graph Isomorphism.” *Physics Procedia* 25:1761–68.

Tiwari, Brijesh K., Nigel. Brunton, and Charles S. Brennan. 2013. *Handbook of Plant Food Phytochemicals : Sources, Stability and Extraction*. Wiley-Blackwell.

Tongnuanchan, Phakawat and Soottawat Benjakul. 2014. *Essential Oils: Extraction, Bioactivities, and Their Uses for Food Preservation*. Vol. 79.

Trott, Oleg and Arthur J. Olson. 2009. “Software News and Update AutoDock Vina : Improving the Speed and Accuracy of Docking with a New Scoring Function , Efficient Optimization , and Multithreading.”

Vella, Frank. 2005. “The Organic Chemistry of Biological Pathways: McMurry, John, and Begley, Tadhg.” *Biochemistry and Molecular Biology Education* 33(5):380.

Vickers, Claudia E. and Suriana Sabri. 2015. “Isoprene BT - Biotechnology of Isoprenoids.” Pp. 289–317 in, edited by J. Schrader and J. Bohlmann. Cham: Springer International Publishing.

Wani, M. C., H. L. Taylor, Monroe E. Wall, P. Coggon, and A. T. Mcphail. 1971. “Plant Antitumor Agents.VI.The Isolation and Structure of Taxol, a Novel Antileukemic and Antitumor Agent from Taxus Brevifolia2.” *Journal of the American Chemical Society* 93(9):2325–27.

Van Westen, Thijs, Thijs J. H. Vlugt, and Joachim Gross. 2011. “Determining Force Field Parameters

Using a Physically Based Equation of State." *Journal of Physical Chemistry B* 115(24):7872–80.

Whittington, D. A., M. L. Wise, M. Urbansky, R. M. Coates, R. B. Croteau, and D. W. Christianson. 2002. "Bornyl Diphosphate Synthase: Structure and Strategy for Carbocation Manipulation by a Terpenoid Cyclase." *Proc Natl Acad Sci U S A* 99(24):15375–80.

Williams, Cheryll J. 2011. *Medicinal Plants in Australia. Volume 2, Gums, Resins, Tannin and Essential Oils*.

Xu, Jinkun, Jingwei Xu, Ying Ai, Roba A. Farid, Li Tong, and Dong Yang. 2018. "Mutational Analysis and Dynamic Simulation of S-Limonene Synthase Reveal the Importance of Y573: Insight into the Cyclization Mechanism in Monoterpene Synthases." *Archives of Biochemistry and Biophysics* 638(October 2017):27–34.

Yamada, Yuuki, Tomohisa Kuzuyama, Mamoru Komatsu, Kazuo Shin-ya, Satoshi Omura, David E. Cane, and Haruo Ikeda. 2015. "Terpene Synthases Are Widely Distributed in Bacteria." *Proceedings of the National Academy of Sciences* 112(3):857–62.

Yao, Jianzhuang, Feng Chen, and Hong Guo. 2018. "QM/MM Free Energy Simulations of the Reaction Catalysed by (*4S*)-Limonene Synthase Involving Linalyl Diphosphate (LPP) Substrate." *Molecular Simulation* 44(13–14):1158–67.

Yoshikuni, Yasuo, Thomas E. Ferrin, and Jay D. Keasling. 2006. "Designed Divergent Evolution of Enzyme Function." *Nature* 440(7087):1078–82.

Zebec, Ziga, Jonathan Wilkes, Adrian J. Jervis, Nigel S. Scrutton, Eriko Takano, and Rainer Breitling. 2016. "Towards Synthesis of Monoterpenes and Derivatives Using Synthetic Biology." *Current Opinion in Chemical Biology* 34:37–43.

Zhao, Lishan, Wei-chen Chang, Youli Xiao, Hung-wen Liu, and Pinghua Liu. 2013. "Methylerythritol Phosphate Pathway of Isoprenoid Biosynthesis." *Annual Review of Biochemistry* 82:497–530.

Appendix

1. Script to create and processing graphs and creating molecular models

```
#!/usr/bin/python

import random
import math
import os
import threading
import datetime
import sys



noargs=len(sys.argv)
#print "This is the name of the script: ", sys.argv[0]
#print "Number of arguments: ", noargs
#print "The arguments are: " , str(sys.argv)

if(noargs != 2):
   print "Error - this script"
   print "requires exactly one parameter relating to 10_XX.txt"
   print "./test_Y10_alternate.py 0   will run the calculation on 10_0.txt
and produce logfile_0.txt"
   exit(1)



thefile=sys.argv[1]
file_title="10_"+str(thefile)+".txt"
thelog="logfile_"+str(thefile)+".txt"
os.system("rm "+thelog)
```

```
acceptable =[]                                  #List of acceptable used in
Route1 of the code which uses readlines and not read
needed_graphs_dict = {}                         #Dictionary from route1
graph_title_list = []                           #List  containing  graph  titles
from Route 1
nodes = 10
n1 = nodes + 2
#file_title = '10_0.txt'

#########################################################################
#########################################################################
#########################################################################

#########################################################################
def min_energy_dict(extracted_funct_list_local):
      energy_dict = {}
      for             (index1,             minimum_value_1)             in
enumerate(extracted_funct_list_local):
            if minimum_value_1 in energy_dict:
                  energy_dict[minimum_value_1].append(index1)


        else:
            energy_dict.update({minimum_value_1:[index1]})
            #energy_main_dict.update({minimum_value_1:[index1]})
            print ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>"
            print "preapring energy_dict"
        return energy_dict

      # for key1, value1 in energy_dict.iteritems():
      #   if len(value1) > 1:
      #           #print "this is ", key1, " value", value1
      #           energy_main_dict[key1] = value1
      #           #print "****************************************"
      #           # print "this is the main energy_dict", energy_
```

```
      #           #print "this is the energy_dict", energy_dict
      #           # minimum_value = min(extracted_funct_list)
      #           # min_e = energy_dict[minimum_value]
      #           # print "\n minimum value", minimum_value

      #           # print "\n Index occurances of minimum value", min_e

      #           for e_val in energy_dict:
      #                 #minimum_value = min(extracted_funct_list)
      #                 if float(e_val) == float(minimum_value):
      #                       print "\n Positions minimum duplicates occur:
", energy_dict[minimum_value]


      #                       print
"************************************//////////////////////////////////////
//////////////////////////**************************************************"




########################################################################

def
chirals_screener(energy_dict_local2,minimum_value_local,dict_t,chiral_scale
_list_local):
      for i_index in energy_dict_local2[str(minimum_value_local)]:
            print            "THE            ENERGIES            INDEXES",
energy_dict_local2[minimum_value_local]
            if     float(chiral_scale_list_local[i_index])     >     3     or
float(chiral_scale_list_local[i_index]) < -3:
                  print  "values  greater  than  3  or  less  than  -3",
chiral_scale_list_local[i_index]
                  print "value index", i_index

                  folder_name = str(dict_t)
```

```
                print "folder name ",folder_name
              file_ext = int(i_index)+2
                finalstring2 = dict_t+"_"+str(file_ext)+".pdb"
                if (not os.path.isdir("./Chirals")):
                     os.system("mkdir Chirals")
                if (not os.path.isdir("./"+folder_name)):
                     os.system("mkdir "+folder_name)
                #os.system("cd "+folder_name)
                os.system("cp "+finalstring2+" "+folder_name)
                os.system("cp -r "+folder_name+" "+"Chirals")
                os.system("rm -r "+folder_name)
          else:
                no_chiral = float(chiral_scale_list_local[i_index])
                file_ext = int(i_index)+2
                if (not os.path.isdir("./Chiral_negative")):
                     os.system("mkdir Chiral_negative")
                folder_name = str(dict_t)
                finalstring2 = dict_t+"_"+str(file_ext)+".pdb"
                if (not os.path.isdir("./Chirals")):
                     os.system("mkdir Chirals")
                if (not os.path.isdir("./"+folder_name)):
                     os.system("mkdir "+folder_name)
                os.system("cp "+finalstring2+" "+folder_name)
                os.system("cp -r "+folder_name+" "+"Chiral_negative")
                os.system("rm -r "+folder_name)
##########################################################################
##########################################################################
########

#########################################################

def graph_screener(file_name_local):
      invalids_list =[]
      graphnumber = 0
      with open(file_name_local, 'r') as read_file_local:
```

```
            lines_local          =          read_file_local.readlines()
      #input the file

            print  "The  length  of  the  whole  file  as  a  list",
len(lines_local)
                  #print  "Graph  screener  process  starting  first
loop..............."
            #for line in lines:
                  #if line.startswith('Graph'):
                        #graph_title_list.append(line)
                        #print(len(graph_title_list))
                        #print(lines.index(line))
                        #index1 = lines.index(line)              #identify
where in the lines this particular graph starts
                        #index_list.append(index1)               #record
this index in the index_list
            #print index_list
            #for char in index_list:
                  #my_graphs = lines[char+1:char+n1]              #+1 makes
this block start after the heading line
                  #my_graphs_list.append(my_graphs)               #now   we
know all of the indexes, we can strip out the blocks of lines
            #print(my_graphs_list[852])                          #per
graph
                  #print(my_graphs_list[5])
                  #print(my_graphs)
            #with open("graph_content.txt", 'w') as g_content:
                  #g_content.write(str(my_graphs_list))


            ###B


                  print "Graph screening in progress......"
            for line_local in lines_local:
                        #  print  "Graph screener  process: ..........Looping
through the whole file as a list"
```

```
                line_local = line_local.strip() #remove the newline
                if not line_local:
                        continue                        #this is not necessary
                if line_local.startswith('Graph'):
                        #print(line)
                                list_for_title = line_local.split()
                                the_title = list_for_title[1].split(',')
                                actual_g_number= int(the_title[0])
                        header = line_local
                        graphnumber=graphnumber+1        #extract the headers and counting the number of graphs
                        print "\nGraph screener process < THE LOOP >:.................Now on graph number {}".format(actual_g_number)
                if ":" in line_local:
                        #line2 = lines[3:16]
                        #print(line)
                        words=line_local.split()
                        #print(words)
                        length=len(words)-2
                        #print("number of attachments = "+str(length))
                        #print(line2)
                        if(length>4):
                                #print("Impossible number of attachments")
                                #print("invalid valency on carbon: ", words[0], "on graphnumber", graphnumber)
                                #g_title.append(graphnumber)
                                        #print "\nGraph screener process:.....................Appending invalid graps to list for screening \n"
                                invalids_list.append(actual_g_number)
        #identify the exact graph number that is invalid
                                        print "\nGraph screener process:.............Graph {} appended to invalid list".format(actual_g_number)
```

```
                    #print(invalids_list)
            ###C
                 print                    "Graph                    screener
process:.....................invalids list compiled"
             print "Graph screener process completed........\n\n|||||.....
returning to main code....||||||\n\n.....................\n\n"
             invalids1 = list(set(invalids_list))
             return invalids1
##########################################################################
#########################

def calc_mod_list(graph_list3):
  modified_list=[]
  bonds_number_list = []
  usables_list = []

  print "Calc_mod_list process:.......generating modified list"
  for needed_value in graph_list3:
      if needed_value != graph_list3[0]:##### Handling the empty space
regarded as item in list and skipping it
        needed_line_list = needed_value.split()
        needed_line_list.pop(0)
        needed_line_list.pop(0)
        last_value_list = needed_line_list[-1].strip()####This makes a list
of the last item in the needed line list.['digit', ';'], where digit is any
integer value
      last_value = last_value_list[0]#### This selects the digit only
        needed_line_list2 = needed_line_list
        needed_line_list2.remove(needed_line_list2[-1])#### Removing the
last digit, and handling the "digit;" problem.
      needed_line_list2.append(last_value)#### Adds back the digit to list
this time without the ";".
        bonds_number_list.append(len(needed_line_list2))
      for usable_value in needed_line_list2:
            usables = int(usable_value) + 1
            usables_list.append(usables)
```

```
            new_usables_list=list(usables_list)
            modified_list.append(new_usables_list)
        del usables_list[:]


    print "Calc_mod_list process:..........Modificatio list generated"
    return modified_list

##########################Route2#################################################
#########################################################
def rvector(pointa,pointb):
        #print "rvector point a "
        #print pointa
        returnvector  =  [float(pointb[0])-float(pointa[0]),float(pointb[1])-
float(pointa[1]),float(pointb[2])-float(pointa[2])]
        return returnvector

def crossproduct(u,v):
        #we return uxv
        u1=float(u[0])
        u2=float(u[1])
        u3=float(u[2])
        v1=float(v[0])
        v2=float(v[1])
        v3=float(v[2])

        s=(u2*v3-u3*v2,u3*v1-u1*v3,u1*v2-u2*v1)
        return s

def size(u):
        u1=float(u[0])
        u2=float(u[1])
        u3=float(u[2])
        return math.sqrt(u1*u1+u2*u2+u3*u3)
```

```
def dotproduct(u,v):
      u1=float(u[0])
      u2=float(u[1])
      u3=float(u[2])
      v1=float(v[0])
      v2=float(v[1])
      v3=float(v[2])
      dot=u1*v1+u2*v2+u3*v3
      return dot


def calc_chirality(inputpdb):
            pdbfile=open(inputpdb,"r")
            lines=pdbfile.readlines()
            atomcount=0;
            atomarray = [[0]*3 for i in range(12)]#only goes up to 12 atoms
with 3 coordinates

            for line in lines:
            if "HETATM" in line:
                  #print line
                   words=line.split()
                  #print   str(words[5])+   "   "+str(words[6])   +   "   "   +
str(words[7])
                   atomarray[atomcount][0]=words[5]
                   atomarray[atomcount][1]=words[6]
                   atomarray[atomcount][2]=words[7]
                   atomcount=atomcount+1
            n=2
            m=1   #powers that make the index dimensionless
            prefactor=float(math.factorial(4))/float(((atomcount**4)   * 3))
# 4!
            weight=12.0*12.0*12.0*12.0 #wi * wj * wk * wl where all atoms
are carbon with mass 12
```

```
            factor=prefactor*weight
            totalsum=0.0

            for i in range(0,atomcount):
            for j in range(0,atomcount):
                  for k in range(0,atomcount):
                        for l in range(0,atomcount):
                              if((i != j) and (i != k) and (i != l) and (j
!= k) and (j != l) and (k != l)):
                                    #print  str(i)+"  "+str(j)+"  "+str(k)+"
"+str(l)

                                    rij=rvector(atomarray[i],atomarray[j])
                                    rkl=rvector(atomarray[k],atomarray[l])
                                    ril=rvector(atomarray[i],atomarray[l])
                                    rjk=rvector(atomarray[j],atomarray[k])

                                     cp1 = crossproduct(rij,rkl)
                                    dot1 = dotproduct(cp1,ril)
                                    dot2 = dotproduct(rij,rjk)
                                    dot3 = dotproduct(rjk,rkl)

                                    numerator=dot1*dot2*dot3

                                      denominator                           =
(((size(rij)*size(rjk)*size(rkl))**n))*((size(ril))**m)

                                    final=numerator/denominator

                                       totalsum=totalsum+final


            pdbfile.close();

            totalsum=factor*totalsum
```

```
            totalsum = round(totalsum,3)
            return totalsum

#############################################################################
########
def modify_line(modified_list_local):
  counter=1
  final_line_list_local=[]
  print "\nModify_line process:.....Generating final_line_list"
  for modified_line in modified_list_local:
            line=str(counter)+'\t'                +                "C"+'\t'+
str((round(((random.uniform(1,2)+random.uniform(1,4))/2), 3))+0.001) + '\t'
+  str((round(((random.uniform(1,5)+random.uniform(1,4))/2),  3))+0.001)  +
'\t' + str((round(((random.uniform(1,5)+random.uniform(1,4))/2), 3))+0.001)
+ '\t' + "  1"
              counter=counter+1
            for char in modified_line:
                line = line + "  " + str(char)
              final_line_list_local.append(str(line))
  print "\nModify_line process.....final_line_list generated\n"
  return final_line_list_local


#############################################################################
######
def write_xyzfile(xyz_file_name,final_line_list_local,nodes_local):
      with open(xyz_file_name, 'w') as my_atoms:
                my_atoms.write(str(nodes_local) +'\n')
                  for xyz_string in final_line_list_local:
                  my_atoms.write(str(xyz_string)+'\n')
                my_atoms.close()

                # my_atoms.write(str(final_line_list_local[0]) + '\n')
                  # my_atoms.write(str(final_line_list_local[1]) + '\n')
                  # my_atoms.write(str(final_line_list_local[2]) + '\n')
```

```
                # my_atoms.write(str(final_line_list_local[3]) + '\n')
                # my_atoms.write(str(final_line_list_local[4]) + '\n')
                # my_atoms.write(str(final_line_list_local[5]) + '\n')
                # my_atoms.write(str(final_line_list_local[6]) + '\n')
                # my_atoms.write(str(final_line_list_local[7]) + '\n')
                # my_atoms.write(str(final_line_list_local[8]) + '\n')
                # my_atoms.write(str(final_line_list_local[9]) + '\n')


##########################################################################
######
def                                    write_min_pdb(current_xyz_name_local,
converted_pdb_name_local,enantiomer_name_local):
      xyz_to_pdb(current_xyz_name_local,converted_pdb_name_local,enantiomer
_name_local)
   # sentence_string_local = ''
   # header_local=''
   # with open(current_xyz_name_local) as read_in_file:
   #        header1 = read_in_file.readline()
   #          header_local = header1.strip()
   #        for line in read_in_file:
   #               sentence_string_local += str(line[8:48])+'\n'

   # with open("temporary.xyz", 'w') as xyz_output:
   #          xyz_output.write(str(header_local)+'\n')
   #
xyz_output.write("graph_"+str(header_local)+"_molecules"+'\n')
   #       xyz_output.write(sentence_string_local)
   #          xyz_output.close()

      #os.system("babel       -ixyz       temporary.xyz       -h       -opdb
"+converted_pdb_name_local)
      #xyz_to_pdb("temporary.xyz",converted_pdb_name_local)
##########################################################################
#######
```

```
#########################################################################
######
def  minimize(xyz_to_minimize,output_xyz,temporary_name):
 #print "minimize xyz "+xyz_to_minimize+" temp "+temporary_name
 if os.path.isfile("./"+temporary_name):
      os.system("rm "+temporary_name)#+" "+temporary_name+"_2")
 if os.path.isfile("./"+temporary_name+"_2"):
      os.system("rm "+temporary_name+"_2")
 os.system("cp "+xyz_to_minimize+" "+temporary_name)
 #logfile=temporary_name[:-4]
 #logfile=logfile+".log"
 #print "logfile ",logfile
 #os.system("cat "+temporary_name)
 result=os.popen("/opt/chemistry/Tinker/minimize "+temporary_name+" 0.01 |
grep Function").readline()
 #line=""
 #result=os.system("/opt/chemistry/Tinker/minimize "+temporary_name+" 0.01
> "+logfile)
 #print("/opt/chemistry/Tinker/minimize "+temporary_name+" 0.01")
 #with open(logfile) as file_temp:
 #    lines=file_temp.readlines()
 #    for line in lines:
 #    if "Function" in line:
 #             result=line

 os.system("mv "+temporary_name+ "_2 "+output_xyz)
 if os.path.isfile("./"+temporary_name):
      os.system("rm "+temporary_name)#+" "+temporary_name+"_2")
 if os.path.isfile("./"+temporary_name+"_2"):
      os.system("rm "+temporary_name+"_2")
```

```
 result=result[:-1]
 return result

#########################################################################
######
def
write_energy_values(logfilename,result_array_local,chirality_array_local,ne
eded_g_name_local):
      logfile=open(logfilename,'a')
      logfile.write("Graph"+"\t"+str(needed_g_name_local)+"\t"+"----------
----------------------------\n")
      for repeat in range(2,10):
      #print "repeat is :"+str(repeat)
      #print "the ", result_array_local[repeat]
      logfile.write("repeat                is                :"+str(repeat)+"
"+str(result_array_local[repeat])+"                    :                    CI
"+str(chirality_array_local[repeat])+"\n")
      logfile.close()

#########################################################################
######
def full_calc(needed_g_name, repeat, nodes,final_line_list_local):
  graph_xyz_file_name = str(needed_g_name)+'_'+str(repeat)+'.xyz'
  current_xyz_name=graph_xyz_file_name+"_"+str(repeat)
  temp_file_name= "temporary_"+str(repeat)+".xyz"
  converted_pdb_name=str(needed_g_name)+"_"+str(repeat)+"_a.pdb"
  enantiomer_name=str(needed_g_name)+"_"+str(repeat)+"_b.pdb"
  #print        "graph_xyz        "+graph_xyz_file_name+"        current_xyz
"+current_xyz_name+"     temp_file_name     "+temp_file_name+"     converted
"+converted_pdb_name
  write_xyzfile(graph_xyz_file_name,final_line_list_local,nodes)
  result=minimize(graph_xyz_file_name,current_xyz_name,temp_file_name)
  write_min_pdb(current_xyz_name, converted_pdb_name,enantiomer_name)
  #print "result in full_calc "+str(result)
  return result
```

```
def full_calc_name(needed_g_name, repeat):
  converted_pdb_name=str(needed_g_name)+"_"+str(repeat)+"_a.pdb"
  return converted_pdb_name

#########################################################################
######
#########################################################################
######
class myThread (threading.Thread):
   def __init__(self, threadID, name,
needed_g_name,repeat,nodes,final_line_list):
      print "myThread process:.............Running threads in classes"
      threading.Thread.__init__(self)
      self.threadID = threadID
      self.name = name
      self.result=""
      self.needed_g_name=needed_g_name
      self.repeat=repeat
      self.nodes=nodes
      self.final_line_list=final_line_list
      self.chirality=0.0
   def getresult(self):
      return self.result
   def getid(self):
      return self.repeat
   def getchirality(self):
      return self.chirality
   def run(self):
      #print "Starting " + self.name
      self.result=str(self.repeat)+":
"+full_calc(self.needed_g_name,self.repeat,self.nodes,self.final_line_list)

      final_file=full_calc_name(self.needed_g_name,self.repeat)
      self.chirality=calc_chirality(final_file)
```

```
      print    "\n    myThread    process:.....thread    end    of
function.....\n "
##########################################################################
######
##########################################################################
######
##########################################################################
######
##########################################################################
######



def xyz_to_pdb(file_to_open,pdb_output,enantiomer_output):
  writing_cont_list = []#List containing the HETATM section
  writing_enant_list=[]
  connect_list =[]#List containing the connectivity section
  conect_output=[]
  with open(file_to_open, 'r') as read_file:
      lines = read_file.readlines()
      for line in lines:
                  if line != lines[0]:
                        line_cont_list = line.split()
                        n=int(line_cont_list[0])
                        x=float(line_cont_list[2])
                        y=float(line_cont_list[3])
                        z=float(line_cont_list[4])
                        xn=-x
                        yn=-y
                        zn=-z
                        line_string    =    "HETATM%5d       C           LIG
      1%4s%8.3f%8.3f%8.3f%6.2f%6.2f%11sC" % (n,"",x,y,z,0.0,0.0,"")
                        enantiomer_string   =   "HETATM%5d      C         LIG
1%4s%8.3f%8.3f%8.3f%6.2f%6.2f%11sC" % (n,"",xn,yn,zn,0.0,0.0,"")
                        writing_cont_list.append(line_string)
```

```
                        writing_enant_list.append(line_string)
                        con_string="CONECT%5d" % n
                        for con in line_cont_list[6:]:
                              con_string += "%5d" % int(con)
                        connect_list.append(con_string)

    with open(pdb_output, 'w') as write_146:
        write_146.write("COMPND graph_10_molecules "+'\n')
        write_146.write("AUTHOR W. DENDERA CUSTOM GENERATION"+'\n')
        for needed_string in writing_cont_list:
                  write_146.write(str(needed_string)+'\n')
        for connect in connect_list:
                  write_146.write(str(connect)+'\n')
        write_146.write("END\n")
        write_146.close()
    with open(enantiomer_output, 'w') as write_146:
        write_146.write("COMPND graph_10_molecules "+'\n')
        write_146.write("AUTHOR W. DENDERA CUSTOM GENERATION"+'\n')
        for needed_string in writing_enant_list:
                  write_146.write(str(needed_string)+'\n')
        for connect in connect_list:
                  write_146.write(str(connect)+'\n')
        write_146.write("END\n")
        write_146.close()
```

```
##########################################################################
######
##########################################################################
######
##########################################################################
######
```

```
#########################################################################
######

graph_dict={}
#########################################################################
#############################################################

with          open(file_title,          'r')          as          order_10:
                                #Opening the file required as a read file
      lines                          =                          order_10.read()
                                 #Reading in the whole document
      lines_split_on_g                =                lines.split('Graph')
                          #Creating  a  list  of  graphs  by  spliting  graph
content on the line containing "Graph"
      #for i in lines_split_on_g:
      #for i in range(0,len(lines_split_on_g)):
          #   print "#####################################################"
      #   print lines_split_on_g[i]
      #print "Number of graphs", len(lines_split_on_g)-1

      #graph_dict = {}
      print "\n\nInitializing Graph screener>>>>> >>>>>>>>>> >>>>>>>>>>>>>
>>>>>>>>>>>\n\n"
      invalids_list_set             =             graph_screener(file_title)
                          #calling the graph screening function
      print                          "\n\nGraph                          screening
completed<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<\n\n"

      for          g_raph          in          range(0,len(lines_split_on_g)):
                                #Range for all graph names to be called
                    print       "we       are       now       at       position
"+str(g_raph)+".....................Graphs       left       to       proces
      "+str(len(lines_split_on_g)-int(g_raph))

            if g_raph not in invalids_list_set:
                                #Selecting  the  valid  graph  basing  on  the
condition that it is not in the invalids list
```

```
                    #if "order" in g_raph:
                    graph_dict[str(g_raph)]     =     lines_split_on_g[g_raph]
                          #Adding valid graphs to the graph disctionary
                       print    "\nGraph    "+str(g_raph)+"    added    to    the
dictionary"


print     "\nnow     have     graph_dict\nInitializing     arrays     and
threads.............\n"
triplesarray=[]
counter=0
triple=[]
for needed_g_name in graph_dict:
   print "setting ", counter," ", needed_g_name
   triple.append(needed_g_name)
   if(counter==2):
      counter=0
      triplesarray.append(triple)
      triple=[]
   else:
      counter=counter+1
triplesarray.append(triple)

for triple in triplesarray:
  print len(triple)
  for i in range(0,-1):
      print i," ",triple[i]
      ####################
      ####################
      ####################
      ####################
      ####################
      ####################
      ####################
      ####################
```

```
full_progress=len(graph_dict)
actual_progress=0
for needed_g_name in graph_dict:
   graph_select                    =                    graph_dict[needed_g_name]
                        #Calling   individual   needed   graphs   from   the
dictionary
   if not graph_select == "": #prevents proceeding with empty string
         actual_progress+=1
      print
"#############################################################################
####Working through graph # ",actual_progress," of ",full_progress
      resultarray=['','','','','','','','','','','']
      chiralityarray=['','','','','','','','','','','']
      threadLock = threading.Lock()
      threads = []

      #graph_select                =                graph_dict[needed_g_name]
                        #Calling   individual   needed   graphs   from   the
dictionary
      graph_select_split              =              graph_select.split('.')
                        #spliting the whole graph at the full stop. Since
there is only one full stop the whole graph becomes a list of two elements
      print "graph_select ",graph_select
      print "graph_select_split[0]",graph_select_split[0]
      print "***",needed_g_name,"***"
      graph_select_lines_split      =      graph_select_split[1].split('\n')
                  #Making a list of lines from by splitting on new lines
      graph_select_lines_split.pop()
                                    #Removing unwanted empty spaces at the
end of the list treated as elements of the list
      graph_select_lines_split.pop()
                                    #Removing unwanted empty spaces at the
end of the list treated as elements of the list
      print "aaaaaaaaaaa ",datetime.datetime.now()
      for repeat in range(2,10):
            modified_list = calc_mod_list(graph_select_lines_split)
```

```
            final_line_list=modify_line(modified_list)
            thread      =      myThread(repeat,      "Thread_"+str(repeat),
needed_g_name,repeat,nodes,final_line_list)
            threads.append(thread)
                thread.start()

      print "bbbbbbbbbbb ",datetime.datetime.now()
      #for t in threads:
      #     t.start()
      print "ccccccccccc ",datetime.datetime.now()
      for t in threads:
            t.join()
      print "ddddddddddd ",datetime.datetime.now()

      extracted_funct_list =[]
      chiral_scale_list =[]
      for t in threads:
            theid=t.getid()
            theresult=t.getresult()
            #print "id "+str(theid)+" result "+str(theresult)
            resultarray[theid]=theresult
            chiralityarray[theid]=t.getchirality()
                  print
"*************************************************************************
****"
            print "the result "+str(theresult)
            print "the id "+str(theid)
            print "chirality "+str(chiralityarray[theid])
                  print
"*************************************************************************
****"


            #for minimized_value in funct_values:
            minimized_value_line_list = theresult.split()
```

```
            actual_minimized_value = minimized_value_line_list[-1]
            #print "actual minimized value ", actual_minimized_value
            extracted_funct_list.append(float(actual_minimized_value))
            chiral_scale_list.append(chiralityarray[theid])

      #print extracted_funct_list

      #indx, minimum_value = min((enumerate(extracted_funct_list)))
      extracted_funct_list = map(float, extracted_funct_list)
      #minimum_value = min(extracted_funct_list)
        minimum_value=100000
        minimum_index=0
        current_index=0
      for cvalue in extracted_funct_list:
            print current_index," : ",cvalue
            if cvalue<minimum_value:
               minimum_value=cvalue
               minimum_index=current_index
           current_index+=1
        minimum_chirality=chiral_scale_list[minimum_index]      #not      min
chirality but chirality of lowest energy conformer
      print "minimum_value***********", minimum_value
      print "minimum_index***********", minimum_index
      print "minimum_chirality***********", minimum_chirality
      actual_index=minimum_index+2  #0 is _2 1 is _3
        actual_enantiomer_a=needed_g_name+"_"+str(actual_index)+"_a.pdb"
      actual_enantiomer_b=needed_g_name+"_"+str(actual_index)+"_b.pdb"

      print "enantiomer a ",actual_enantiomer_a
      print "enantiomer b ",actual_enantiomer_b
      if minimum_chirality<0.0:
            minimum_chirality=-minimum_chirality
      if minimum_chirality>3.0:
            print "definitely chiral compound"
          folder_name = str(needed_g_name)
```

```
        print "copying folder:...", folder_name
        if (not os.path.isdir("./Chirals")):
              os.system("mkdir Chirals")
        if (not os.path.isdir("./"+folder_name)):
            os.system("mkdir "+folder_name)
            #os.system("cd "+folder_name)
            os.system("cp "+actual_enantiomer_a+" "+folder_name)
            os.system("cp "+actual_enantiomer_b+" "+folder_name)
        os.system("cp -r "+folder_name+" "+"Chirals")
        os.system("rm -r "+folder_name)
                  ######## copy both files into a folder
    else:
            print "not a chiral compound"
           folder_name = str(needed_g_name)
            print "copying folder:...", folder_name
            if (not os.path.isdir("./Chiral_negative")):
             os.system("mkdir Chiral_negative")
            if (not os.path.isdir("./Chirals")):
             os.system("mkdir Chirals")
            if (not os.path.isdir("./"+folder_name)):
             os.system("mkdir "+folder_name)
           os.system("cp "+actual_enantiomer_a+" "+folder_name)
           #os.system("cp "+actual_enantiomer_b+" "+folder_name)
           os.system("cp -r "+folder_name+" "+"Chiral_negative")
           os.system("rm -r "+folder_name)

            ###### copy one file into a folder


    energy_dict = min_energy_dict(extracted_funct_list)
    print energy_dict
    #
chirals_screener(energy_dict,minimum_value,needed_g_name,chiral_scale_list)
```

```
      #print "Exiting Main Thread"
  # print resultarray
  # print "chiralityarray", chiralityarray
  # print " needed g name", needed_g_name

      write_energy_values(thelog,resultarray,chiralityarray,needed_g_name)


##########################################################**********************
ROUTE1*******************************#########################################
###
#   invalids_list_set   =   graph_screener(file_title)#calling   the   graph
screening function
# print "lines lens", len(graph_title_list)

# for valid in range(1,len(graph_title_list)+1):
# if valid not in invalids_list_set:
#           acceptable.append(valid)
# acceptable_set = list(set(acceptable))
# print len(acceptable)
# print len(invalids_list_set)
# print len(acceptable_set)

# graphs_dict = {}
# for acceptable_g_title in acceptable_set:
#     graphs_dict[str(acceptable_g_title)]=
needed_graphs_dict[str(acceptable_g_title)]

# print graphs_dict['146']
#########################################################################
#########################################################################
##3
```