Machine Learning Towards Automated Discovery of Organic Molecules as Active Materials in Non-Aqueous Redox Flow Batteries

by

E. Claire Anderson

Bachelor of Science in Chemistry with Honours,
University of New Brunswick, 2021

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

In the Graduate Academic Unit of Chemistry

Supervisor:	Stijn De Baerdemacker, PhD, Department of Chemistry
Examining Board:	Anna Ignaszak, PhD, Department of Chemistry
	Gobinda Saha, PhD, Department of Mechanical Engineering

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

June 2024

Abstract

With increasing demand for energy and the resources needed to provide this energy, redox flow batteries (RFBs) have shown potential as large-scale electrochemical energy storage systems. In this project, interest lies in the automated discovery of organic redox-active materials that undergo both oxidation and reduction reactions for symmetric non-aqueous RFBs. Machine learning methods were applied to automate the generation of organic molecules followed by the application of a genetic algorithm (GA) to improve the generated population.

A set of molecules were constructed through a series of random choices under set structural parameters. Multiple GA generations were run on a selected population where two randomly chosen molecules combine their structural features to generate new molecules. All molecules were characterized computationally to determine their cell potential, stability, and solubility values used to assess their capabilities as redox-active materials. A set of top-ranking molecules have been proposed as potential candidates for non-aqueous RFBs.

Acknowledgements

First of all, I would like to acknowledge and thank my supervisor, Dr. Stijn De Baerdemacker, for first taking me on as an undergraduate student for my Honours thesis and then to continue into graduate studies as a Master's student. I would also like to thank the members of the De Baerdemacker group for their support during these past four years, especially Amer El Samman for all of his help throughout these projects. I have learned so much during my time with this group and it has been an honour to be part of such a great team of people. I would also like to thank Dr. Adam Dyker and his group for their helpful discussions on the experimental aspect of this work. My gratitude extends to the faculty and staff involved in the many steps of completing this degree, including but not limited to my advisory and examining committees, Dr. Sara Eisler, Dr. Anna Ignaszak, and Dr. Gobinda Saha. Last but not least, I would like to thank my family and friends for their continued support during my time at UNB. They have never failed to show their confidence in me and for that I am forever grateful; I would not have been able to do this without them.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Attentive FP Attentive Fingerprints

B88 Becke-88

BPY Bipyridine

BPYME [2,2']-bispyridinylidene-4,4'-dimethyl ester

cc-pVDZ Correlation consistent polarized double-zeta

DFT Density functional theory

DMF Dimethylformamide

EDG Electron donating group

EV Electric vehicle

EWG Electron withdrawing group

GA Genetic algorithm

GAMESS General Atomic and Molecular Electronic Structure System

HF Hartree-Fock

HOMO Highest occupied molecular orbital

LYP Lee-Yang-Parr

LiTFSI Lithium bis(trifluoromethanesulfonyl)

LUMO Lowest unoccupied molecular orbital

ML Machine learning

MMFF94 Merck molecular force field

`.mol` **file** MDL Molfile

MO Molecular orbital

obabel Open Babel

PCM Polarizable continuum model

PHES Pumped hydroelectric storage

RHF Restricted Hartree-Fock

RFB Redox flow battery

SCF Self-consistent field

SMD Solvational model density

$TBAPF_6$ Tetrabutylammonium hexafluorophosphate

UPHES Underground pumped hydroelectric storage

Xgboost eXtreme Gradient Boosting

List of Symbols

E Reduction potential (V)

E_{cell} Cell potential (V)

$E_{HOMO-LUMO}$ HOMO-LUMO energy gap (eV)

F Faraday's constant (96.485 kC/mol)

G Gibb's free energy (kJ/mol)

ΔG Change in Gibb's free energy (kJ/mol)

$\Delta G_{solvation}$ Solvation free energy (kJ/mol)

$\Delta G_{stability}$ Stability/free energy of stability (kJ/mol)

n Number of electrons transferred

S Score

σ Standard deviation

Chapter 1

Introduction

1.1 Energy and Redox Flow Batteries

With the increasing demand for energy and the resources needed to provide this energy, researchers have been looking into more environmentally friendly renewable energy sources and storage methods. Fossil fuels are among some of the most commonly used energy sources however their detrimental effects on the environment[1,2] have led to the search for renewable energy sources to minimize environmental impacts. Wind and solar power are among conventionally used renewable energy sources however they can become unreliable due to their dependence on weather and the time of day to be able to provide energy.[3] This is what has led to the development of energy storage systems that are able to store excess energy from renewable energy sources in order to keep up with the increasing energy demands.[4–6] Redox flow batteries (RFBs) have now been proposed as a large-scale electrochemical energy storage system.[7,8] A typical RFB structure is made up of two external electrolyte tanks connected to a fuel cell through which electrolytes are pumped to drive the redox processes that convert electrochemical energy in these battery systems.[9]

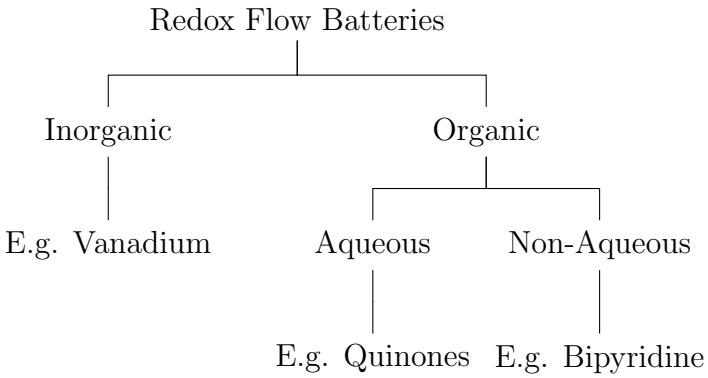


Figure 1.1: Categorization of RFBs.

RFBs can be separated into two general categories: inorganic and organic RFBs (Figure 1.1). The most commonly researched type of RFB is the vanadium RFB which falls into the inorganic category. They were initially proposed as ideal battery systems for use in electric vehicles (EVs) because of their ability to be charged by conventional methods or mechanical refuelling,[10] but their focus has since shifted to large-scale energy storage. Vanadium RFBs have shown issues with low power and energy density, which has subsequently led researchers to investigate organic RFBs as an attempt to improve the power and energy density of RFB systems. There are two type of organic RFBs that differ by the solvent used; aqueous RFBs use water as a solvent whereas non-aqueous RFBs make use of a solvent other than water.[11] These two types of organic RFBs also differ by their relative energy densities. While aqueous RFBs are considered low-energy density systems because of the low potential window of water (1.23 V), non-aqueous RFBs are high energy density systems because they are able to expand their electric potential window and therefore expand the operational voltage of the cell, allowing for more choices of redox couples as redox-active materials for the RFB system.[12] Much of the research on organic RFBs has been toward discovering new electrolytes and redox couples for use in these RFB systems.[11] Comparisons can be made between RFBs and conventional energy storage methods such as Li-ion batteries, fuel cells, and gravity energy storage, however

several disadvantages have led to favouring RFBs for large-scale energy storage over these conventional methods. More details on RFBs are provided in Chapter 2.

1.2 Project Goals

The main goal of this project was to identify a potential set of organic molecules that are the best active materials for use as a redox couple in non-aqueous RFBs. As a result, a method for discovering these materials was also developed. The followed approach makes use of computational chemistry techniques to compute the redox properties of potential candidate active materials, such as cell potential and solvation free energy, and makes use of machine learning (ML) and genetic algorithms (GAs) in order to automate and optimize the search for a potential RFB active material.

1.3 Computational Chemistry

The type of RFB of interest for this project were non-aqueous RFBs based on the experimental study of bispyridinylidenes by the Dyker group at the University of New Brunswick.[13] This work supports their study by computationally assessing a large search area of potential active material candidates based on a general bipyridine (BPY) backbone through the use of ML methods. Computational chemistry is a powerful tool that can be used to accompany and aid in experimental chemistry methods. Both computational and experimental studies have their own advantages, however they may be paired together to support the same studies.

To identify organic molecules as potential redox-active materials for use in non-aqueous RFBs, calculations were run to determine the cell potential (E_{cell}) based on the Nernst equation. Furthermore, stability ($\Delta G_{stability}$) and solubility ($\Delta G_{solvation}$) values for each molecule were computed as the Gibb's free energies of bond-breaking and solvation, respectively, in order to determine their success as redox-active mate-

rial candidates (see Chapter 4).

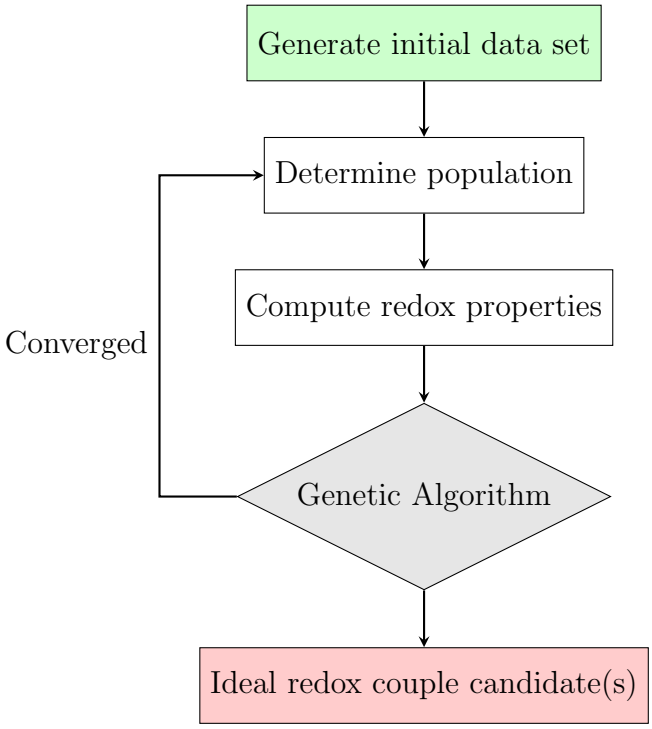


Figure 1.2: General steps involved in the search for redox-active material candidates for use in non-aqueous RFBs.

The organic molecules involved in this project were generated using a Python script that was developed for this project to automate the generation of molecules and subsequently compute the redox properties of each molecule. Figure 1.2 shows the general steps involved in the search for these active material candidates. Each organic molecule was designed to have a two ring backbone connected by a symmetric linker where the two rings are mirror images of each other. A randomly chosen number of up to six nitrogen atoms and double bonds are present throughout each ring as well as a randomly chosen functional group substituted on each ring to keep the overall symmetry of the molecule. The construction of these molecules is discussed in detail in Chapter 4.7.

Each computation on an individual molecule can take anywhere from two to twelve hours depending on the size of the molecules and computational resources

available. Because of this, an exhaustive high-throughput search of all possible molecules is not viable due to time restrictions with an estimated total of 10^{10} possible derivatives that could be computed in such a high-throughput study. Therefore, the search through all possible molecules was optimized using ML and GAs.

1.4 Machine Learning and Genetic Algorithms

One of the most important aspects of this project was automating as much of the described processes as possible. This was done by applying machine learning (ML) which is able to learn observed patterns in data to make predictions about the data. There are a few different types of ML; supervised, unsupervised, semi-supervised, and reinforcement learning. The choice of learning algorithm depends on the specific application of ML. In this project, reinforcement learning was used through the application of a GA. The basis of reinforcement learning is that it works by applying a reward or penalty based on the outcome of the algorithm[14] and is able to use the environment of the algorithm to change its behaviour and improve the overall efficiency of the algorithm.[15]

As mentioned, ML was applied in this project through an initial random generation of organic molecules followed by the application of a GA to generate new molecules whose structures are combinations of those from the initial population. GAs are a type of reinforcement learning algorithm that make use of evolutionary theories such as natural selection, genetics, and survival of the fittest to solve optimization problems. There are three main features of a GA; selection, crossover, and mutation. Selection is a method used to select the molecules that will make up the population for the GA while crossover is the operation that is applied to two randomly chosen parent molecules from that population in order to generate new molecules by combining structural features of the parent molecules. Mutations help

to drive evolution by allowing for new molecules that may not have otherwise been obtained through crossover. They were used in this project as means to fix newly generated molecules that may not follow valency rules due to the random crossover event so that all features of the molecules follow valency rules.[16]

Each molecule is ranked using a scoring function based on the combined performance of cell potential, stability, and solvation free energy values, and a population is selected from the top ranked molecules to be run through a GA. The GA used was designed to randomly choose two molecules from the population to undergo a crossover event and generate a new molecule that is a combination of the two randomly selected molecules with the goal of generating molecules that are better candidates for use as redox-active materials in non-aqueous RFBs than those in the GA population from previous generations (see Chapter 4.8 for more details).

1.5 Results

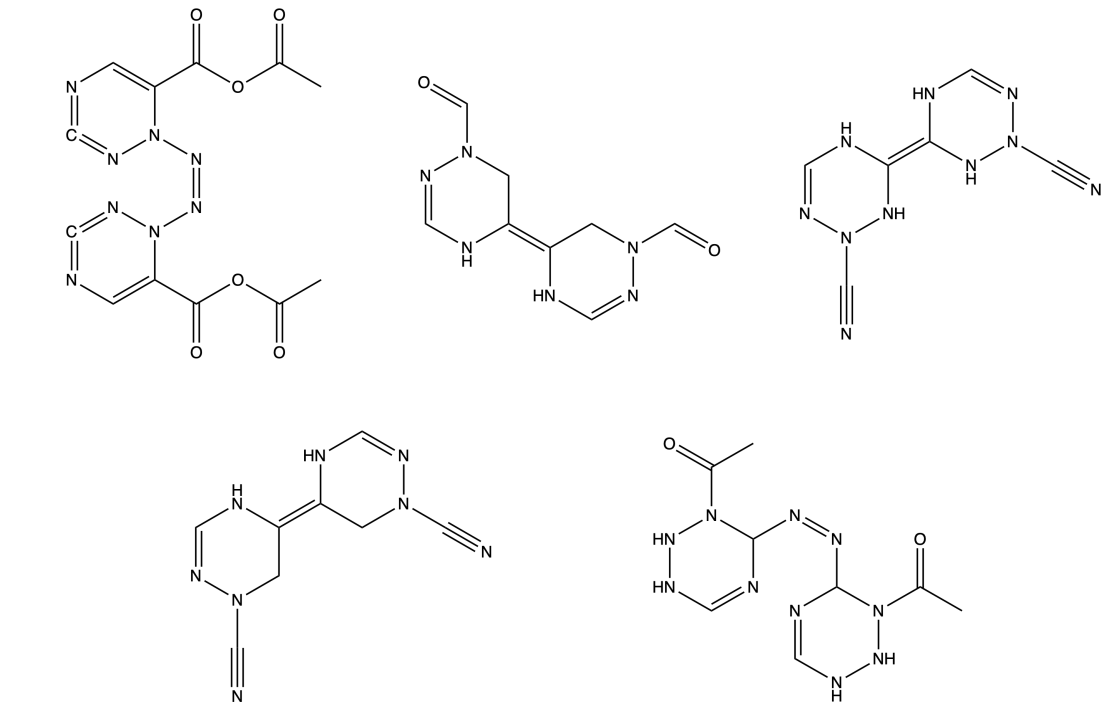
Overall, ML methods were applied to automate the random generation of organic molecules followed by the use of a GA to improve upon the population. These improvements were evaluated through the changes in the distributions of cell potential, stability, solubility, and relative score values of each molecule in the population before the first generation and after the last GA generation. The initial population of randomly generated molecules was made up of 360 molecules, which was further reduced to 156 molecules that fell within the desired windows for cell potential, stability, and solubility (see Chapters 5.4, 5.5, and 5.6). From this reduced population, the molecules were ranked using a scoring function to choose the highest ranked molecules that make up the population for the GA based on their respective cell potential, stability, and solvation free energy values. Two GAs were run; the first used unsubstituted BPY as a reference to compare the population to and the second

used the average values of the randomly generated population as a reference. Both GAs showed improvements of the population throughout multiple generations and from these final populations, a set of organic molecule candidates for non-aqueous RFBs could be proposed.

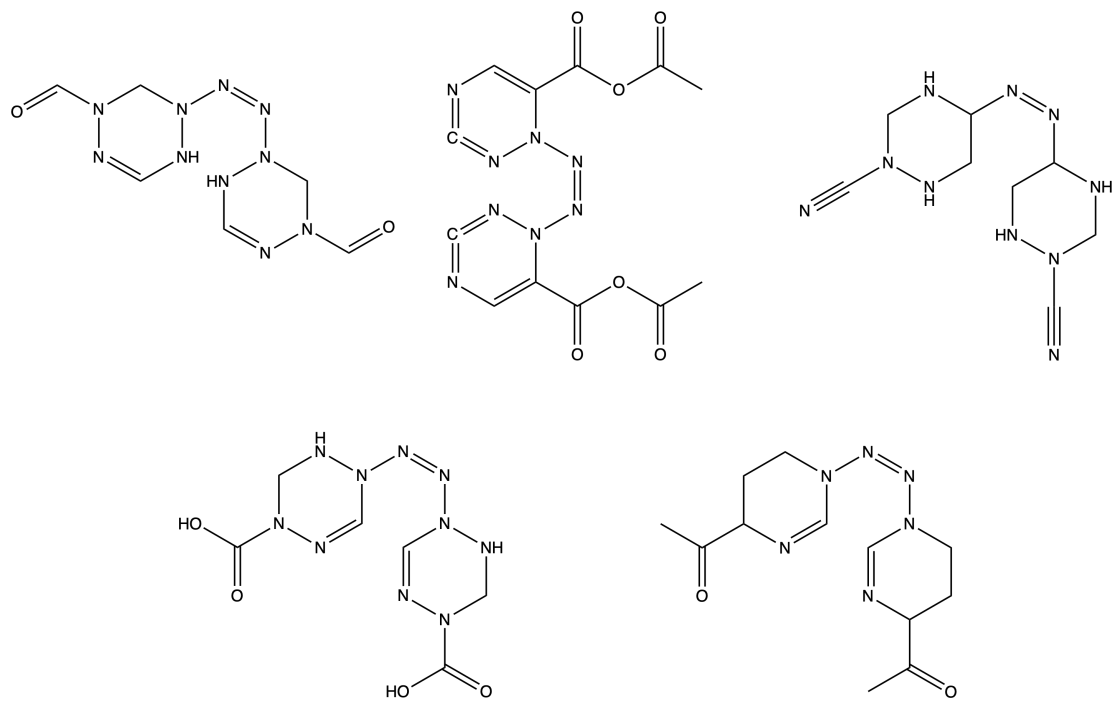
Throughout the course of this project some challenges were encountered. When designing the script for the random automated generation of organic molecules, it was quickly discovered that generating these molecules following constraints and chemical rules is not trivial. Therefore multiple versions of the script were generated before finally converging to an optimal method for randomly generating molecules with chemically viable structures.

1.6 Conclusions

A GA was developed for identifying potential organic RFB candidates based on a scoring function including chemical cell potential, solvability, and stability. Figure 1.3 shows the top five molecules as a result of the two GAs that are potential candidates for use as active materials in non-aqueous RFBs. Future improvements in the process have been identified.



(a) First GA



(b) Second GA

Figure 1.3: Possible active material candidates for use in non-aqueous RFBs as a result of two GAs.

Chapter 2

Redox Flow Batteries

2.1 Introduction

Over the years, the demand for energy and the resources needed to provide this energy has continually increased. Fossil fuel energy has become a popular energy source due to its large reserves of low-cost raw materials, however, this is not a sustainable source of energy in the long-term since these resources are being burned in short amounts of time, leading to major impacts on the environment due to the associated CO_2 emissions.[1,2] It is these detrimental environmental impacts that have led to the search for more environmentally friendly renewable energy sources. While wind and solar power are some of the most commonly used conventional renewable energy sources, they are heavily dependent on the time of day and weather to be able to provide their energy, causing them to be unreliable renewable energy sources.[3]

Traditional energy grids have not been able to keep up with the integration of primary energy sources to the grid, which must take into account the design, control, and management of the electric grid and all of its different features.[17] These traditional energy grids have shown to become unstable when the power from the primary energy source exceeds 20% of the generated power without a suitable energy storage

measure, leading them to perform poorly.[18,19] These are the types of problems that have led to the combination of energy generation from renewable energy sources with energy storage systems that are able to store excess quantities of energy when it is not needed in order to keep up with higher energy demands when needed.[4–6]

A leading development in electrochemical energy storage systems that have shown much potential as a solution to the ongoing energy storage issue is the redox flow battery (RFB).[7,8] Flow battery systems make use of redox processes to convert electric energy into electrochemical energy by using species that are stored in solution in external electrolyte tanks and moved through the flow battery system as needed. RFBs are highly appealing as energy storage devices because of their scalability to the size needed for each application, decoupling of their power and energy, and durability throughout their use.[20]

2.2 Redox Flow Battery Structure

In order to describe the electrochemistry behind the function of RFBs, it is important to first understand how they are assembled. Figure 2.1 shows a typical RFB structure. The flow battery system is made up of two electrolyte tanks connected to a fuel cell by two individual pumps. The external electrolyte tanks are shown as the anolyte and catholyte, which contain the negative and positive electrolyte solutions of the active species with low and high redox potentials, respectively. The fuel cell that connects these electrolyte tanks is made up of two half-cells located between the negative and positive electrodes, labelled as the anode and cathode, respectively. Separating these half cells is an ion-selective membrane. The purpose of this membrane is to allow the passage of counter ions between the two solutions without allowing the electrolyte or active species themselves to pass through. These counter ions are a necessary addition to the electrolytes in order to keep the charge

balance between the electrodes as well as to maintain the overall neutral charge of the system. The electrodes are then each connected to a common power source to drive the redox reactions during the charging and discharging cycles.[9]

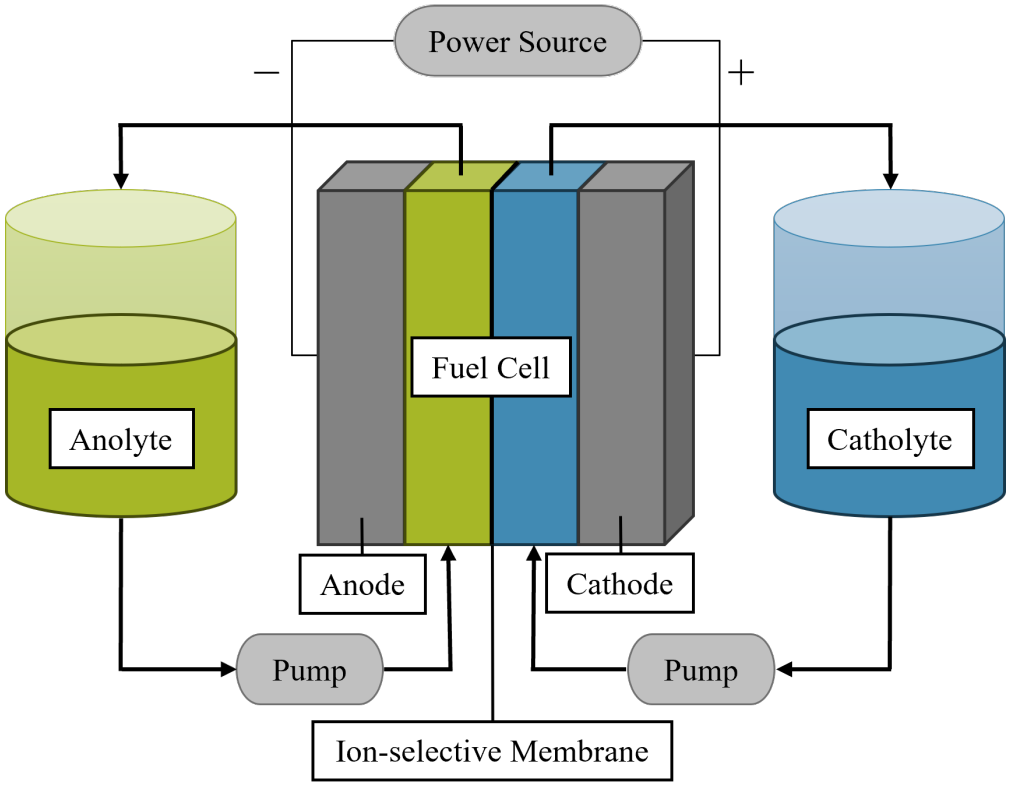


Figure 2.1: Typical RFB structure.

The fuel cells of the RFBs make use of electrochemical principles in order to couple the reduction and oxidation reactions that are occurring in the electrolyte solutions of the RFB system. An electrochemical conversion between the electrical and chemical energy occurs as the pumps move electrolytes in their liquid anolyte and catholyte forms from their electrolyte tanks to flow through the respective electrodes providing an electrochemically active chemical surface on which redox reactions can occur.[11] To keep the electrolyte solutions in the liquid phase, the RFB cell must operate at around room temperature. During the charging/discharging processes, ions in the electrolytes will flow between the electrodes where a reduction half-reaction occurring at one electrode will require electrons that are gained from the oxidation

half-reaction at the other electrode.[3] The respective reduction and oxidation half reactions are

$$A + e^- \longrightarrow A^-$$

$$B^- \longrightarrow B + e^-$$

where A is reduced to A^- by gaining an electron and B^- is oxidized to B by losing an electron.

2.3 Advantages and Disadvantages of RFBs

2.3.1 Advantages

RFBs have many advantages in their application as electrochemical storage systems. First of all, their energy storage system is completely separate from their power conversion, meaning the magnitude of both energy and power can be manipulated on their own. This power and energy separation allows for their storage capacity to be essentially unlimited since in order to increase the capacity, only the size of the storage tanks must be increased.[20] The reactions that occur in RFBs are fully reversible, therefore one RFB cell is able to convert electricity into chemical energy as well as perform the reverse process of converting chemical energy into electrical energy. RFBs also have the advantage of a long life cycle since they are able to charge the ions of the electrodes and electrolytes without actually consuming the ions. Furthermore, due to the external storage of the electrolytes, the handling and storing of liquids in RFBs is easy and the circulation of electrolytes between the electrolyte tanks and electrodes of the fuel cell can be executed simply by using the two attached pumps. As mentioned previously, temperature is an important condition for flow batteries, which can be controlled by regulating the flow of electrolytes through the cell. Lastly, because the two electrolytes are stored externally in separate tanks, self-discharge of the RFB does not occur and this separation allows for

the cell to remain in a discharged state for long periods of time without any effect on the cell itself.[3]

2.3.2 Disadvantages

While RFBs have many clear advantages, they do exhibit some general setbacks as well. Although they have the advantage of their power and energy density being separate from each other, RFBs can exhibit low power and energy density, which makes them poor candidates for use in small electronic devices such as laptops and cell phones. The active areas of the cell are fairly large, which increases the size of the battery itself, causing high transverse gradients of the solutions that are flowing inside of the electrodes and in turn lowering the average current and current density of the cell. Although it is easy to control the temperature of the RFB cell, the temperature of some electrolytes must be monitored closely to avoid precipitation of the electrolyte solution. The ideal temperature for solutions prone to precipitation is between 15 to 35 °C as any temperature above or below this range will cause the solution to precipitate.[3]

2.4 Organic and Inorganic Redox Flow Batteries

There are two main categories of RFBs: inorganic and organic RFBs. The most common type of RFB is the inorganic RFB. While many types of inorganic RFBs have been researched, the vanadium RFB has become the most developed.[21] Whereas other types of inorganic RFBs have shown issues such as the cross-contamination of electrolytes as seen in Fe/Cr, Fe/Ti, and polysulfide bromine RFBs due to the use of inadequate membranes,[22] the vanadium RFB is able to avoid this cross-contamination issue by using the same electrolyte system in both of its half-cells. This type of symmetric RFB helps to avoid the problems of electrolyte

maintenance and degradation that non-symmetric RFB systems face.[10]

Organic RFBs can be further categorized as either aqueous or non-aqueous, each with their own advantages and disadvantages. The main difference between these two types of organic RFBs are the solvents that they each use. For aqueous RFBs, the main component of the system uses water as a solvent, whereas non-aqueous RFBs have a liquid other than water as the solvent for their system. In both organic and inorganic RFBs, the liquids of the RFB are stored separately from their reactive materials, making them fairly safe battery systems. Another inherent safety feature implemented into RFB systems is that the flow of electrolytes during the redox reactions carries the generated heat away from the reaction, helping to dissipate heat which is an issue seen in conventional batteries with solid state active materials.[11]

There are some characteristic features of RFBs to be discussed that make up the most important differences between aqueous and non-aqueous RFBs.

1. Energy density
    (a) Cell potential
    (b) Number of electrons
    (c) Solubility
2. Stability
3. Cost

Energy density is an important feature of RFBs because it quantifies the amount of energy a battery can hold proportional to its size or weight, making high-energy density systems more favourable over low-energy density systems as they are able to store more energy in smaller battery systems. The energy density of RFB systems can be increased by making use of a battery system whose active materials are

capable of producing a larger potential difference and in turn a higher voltage.[23] Aqueous and non-aqueous RFBs themselves differ in their relative energy densities. Aqueous RFBs are considered low-energy density systems because they are restricted by the concentration of their active materials, as well as by their water electrolysis. Non-aqueous RFBs on the other hand are considered high-energy density systems because of their ability to expand their *cell potential* window defined as the voltage window in which electrochemical reactions start to occur in the solvent rather than the electroactive material. This leads to a significant increase in the number of redox couples that may be possible as active materials. Furthermore, it allows for additional advantages such as *multi-electron* transfer redox reactions, which help to further increase the energy density of non-aqueous RFBs.[12] Molecules that are able to undergo multi-electron redox reactions are able to increase the energy of the system by nature of the Nernst equation (See Chapter 4.4 on redox properties).

Attempts to make use of multiple redox couples for aqueous RFBs have been made,[12] however, they are limited by their small potential window in comparison to their non-aqueous counterparts that exhibit a larger potential window, allowing for the possibility of high energy density systems.[11,24–26] This has led the redox couples of non-aqueous solutions in general to exhibit higher *solubility* and *stability* in the RFB system. On the flip side, aqueous RFBs are typically cycled at higher current densities than their non-aqueous counterparts and the low current density of non-aqueous RFBs is partially associated with lack of energy efficiency. Non-aqueous RFBs are also subject to degradation throughout their charge-discharge cycles, which can account for some issues with stability. However, overall non-aqueous RFBs are at an advantage in the flexibility of their system's design, made possible by having a more expansive potential window as well as having more options for possible redox-active redox couples and the ability to implement multi-electron transfer redox couples into their design. Much of RFB research has been in the area of discover-

ing new electrolytes and the search for ideal redox couples and their complimentary electrolytes. Less research has been conducted on the electrode and ion-selective membrane components of the flow battery system for non-aqueous RFBs. Limitations in the further development of non-aqueous RFBs are related to low electrolyte conductivity, the need for temperature management throughout the battery system, and the *high costs* of implementing non-aqueous RFBs into flow battery chemistry.[11] However, these topics do not form the direct subject of this work.

2.5 Recent Research in RFB Chemistry

2.5.1 Vanadium and Zinc-Based Inorganic RFBs

Before their application as large-scale energy storage systems, vanadium RFBs were developed for use in electric vehicles (EVs). They were proposed as ideal battery systems for EV applications due to their ability to be recharged by conventional methods or by mechanical refuelling.[10] The vanadium RFB that was used in EVs was considered as the "Generation 1" vanadium RFB where both half-cells contain a solution of vanadium in sulfuric acid where the cathode operates by the $VO^{2+}/VO_2{}^{+}$ redox couples while the anode operates by the V^{2+}/V^{3+} redox couple.[22] There have been studies performed on the reactions of the vanadium redox couples in different electrolytes and electrode surfaces. Electrolytes are an important determining factor of the stability of the ions during the charge-discharge cycles of vanadium RFBs. Sulfuric acid showed to be the best electrolyte for these RFBs as it allows for adequate solubility of the different vanadium ions.[27] A variety of electrode materials have been proposed, such as different types of graphite and carbon electrodes. These electrodes have shown to function well under typical conditions for charge-discharge cycling, however, they have also shown to undergo slow oxidation as a result of the cell being overcharged and the production of oxygen. This slow oxidation has caused

the carbon and graphite surfaces to slowly disintegrate over time in the case of the cathode but not in the case of the anode. Metal electrodes have also been studied using Pb, Au, Pt, platinized Ti, and IrO_2 as stable electrode materials, however it has been shown that the graphite and carbon electrodes are much better suited for both the anode and cathode materials of vanadium RFBs, given the cell is not overcharged and is relatively stable.[28]

The vanadium RFBs as described along with zinc-based RFBs have been shown to be close to commercialization, however, they do have some setbacks with their low power and energy densities that prevent them from achieving this. The main research focus has been on improving the power and energy densities of the flow battery systems while maintaining a low cost, for instance by the design of stacks, or development of highly soluble electrolytes however with their own potential drawbacks.[29]

Due to the disadvantages associated with the high cost of vanadium and other materials, researchers have shifted some focus towards organic non-aqueous RFBs as new energy storage systems.[30,31] The research on non-aqueous RFBs is currently mostly focused on improving the battery voltage, however they do have the setback of having low power densities and inadequate charge/discharge cycling performance. On the other hand, aqueous RFBs have been mainly researched to find methods of improving their energy density while also decreasing the costs associated with them.[29] Various redox-active organic molecules have been studied in non-aqueous solvents, however despite achieving more expansive electrochemical potential windows, there still remain issues with the stability, solubility, and energy density of these organic molecules as active materials for RFBs.[31]

2.5.2 Quinones for Aqueous Organic RFBs

One type of organic redox-active species that has been proposed for use in aqueous RFBs are quinones.[32] The backbone structure of a quinone molecule can be seen in Figure 2.2 below.

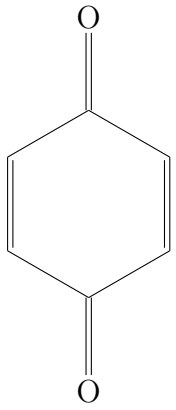
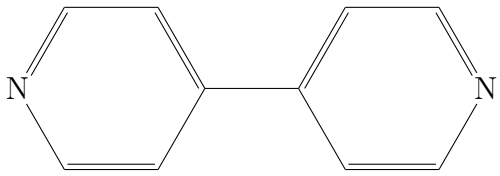


Figure 2.2: Structure of a quinone molecule.

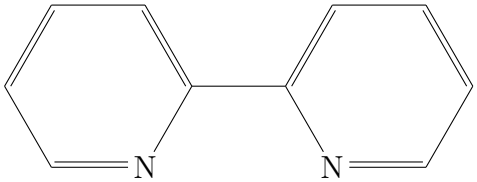
Quinone-based organic RFBs make use of the oxidized and reduced forms of two non-necessarily equal quinone molecules as the respective electrolytes. The two quinone molecules undergo a charging/discharging cycle between the two forms of quinone where the difference of the reduction potentials are used to determine the overall cell potential of the RFB. Quinones exhibit favourable electrochemical reversibility and reaction rates and are able to be designed as low-cost RFBs due to being metal-free organic compounds consisting of earth abundant elements. While there are advantages to the tunability of quinone compounds for RFBs, they are also subject to issues of stability and low solubility as well as experiencing limits of their operating voltages. The low solubility of quinones is related to their pH dependence, where they typically exhibit low solubility at neutral pH. The solubility of quinones may be increased by adding acid or base, however, this in turn decreases their stability as electroactive materials.[32]

2.5.3 2,2'-bipyridine as Electro-active Compounds

Throughout the ongoing study of organic redox couples as new low-cost electrolytes for RFBs, the use of computational techniques to aid in this exploration has been proposed by Sánchez-Castellanos *et. al.*[33] Various molecular parameters such as redox potential, solubility, and electrochemical stability have been studied computationally through the use of high-throughput calculations, which can help accelerate the search for new redox-active compounds.[34,35] Some research has been carried out to computationally study various quinone derivatives such as anthraquinone,[34,36,37] benzoquinone,[38] and thiophenoquinone[35] as well as 4,4'-bypyridinium[34,39] derivatives for use as active materials in RFBs. Most research performed on bipyridine (BPY) derivatives for RFBs has focused on the methyl viologen derivative, which is a di-substituted derivative of 4,4'-BPY shown in Figure 2.3a. In the study by Sánchez-Castellanos *et. al*,[33] focus was shifted to 2,2'-BPY as an electro-active compound for RFBs because it has shown good electrochemical reversibility.[40,41] The structure of 2,2'-BPY is shown in Figure 2.3b where addition of a functional group can occur at any position on the rings.



(a) 4,4'-BPY



(b) 2,2'-BPY

Figure 2.3: Structures of two BPY derivatives.

In the work by Sánchez-Castellanos *et. al*,[33] the molecular energies of 156 BPY

derivatives with substituted nitro- and carbonyl-based functional groups were calculated and their redox potentials were determined for the first and second electron transfers since these 2,2'-BPY derivatives are capable of two reduction and two oxidation reactions. The polarizable continuum model (PCM)[42] and solvational model density (SMD)[43] implicit solvation models were included to account for solvent effects. Through the results of these computations, the effects of functional groups were studied, where it was observed that the reduction potential of 2,2'-BPY derivatives shifted towards more positive values when substituted with electron withdrawing groups (EWGs) as compared to the unsubstituted BPY molecule, due to the stabilizing effect that EWGs have on the aromatic rings. The opposite is true for the addition of electron donating groups (EDGs) where the 2,2'-BPY derivatives saw a decrease in the reduction potential values that is much less significant than the shift seen for the addition of EWGs. EDGs make it harder for electrons to enter the aromatic ring, causing this decrease in reduction potential. EWGs themselves may also allow for the increased reversibility of BPY derivatives because of their ability to stabilize electrons during the reduction reaction. Overall, Sánchez-Castellanos *et. al*[33] were able to determine that BPY derivatives that are substituted with nitro-based and carbonyl-based functional groups may be most successful as electro-active materials for use in organic RFBs.[33]

2.5.4 Ester-Substituted Bispyridinylidenes for Organic RFBs

Redox-active organic molecules have been studied in asymmetric non-aqueous RFB systems as the anolyte and catholyte, however, crossover of the active materials has been observed, affecting the overall lifetime and capacity of the cell.[44–46] Instead, symmetric organic RFBs have been proposed as potential solutions to this crossover issue. Symmetric RFB systems make use of one redox-active molecule that is able to perform in multiple oxidation states and therefore exists as both the anolyte

and catholyte as two different redox couples.[47,48] Researchers have now identified ester-substituted bispyridinylidenes as double concerted two-electron molecules for symmetric organic RFB systems. The molecule of interest is [2,2']-bispyridinylidene-4,4'-dimethyl ester (BPYME) whose structure is shown in Figure 2.4 below, with a similar backbone to the BPY molecule seen in Figure 2.3b.[13] There already exists a sequential two-electron redox process between the $BPYME^{2+}$/BPYME redox couple, proposed by Zhang *et. al*,[49] and because of this, it has been shown that the two-electron BPYME/$BPYME^{2-}$ redox couple also exists through the use of electron withdrawing ester substituents.[13]

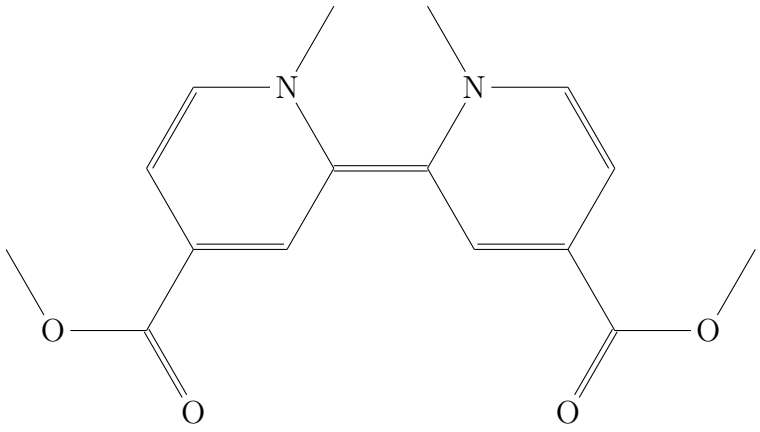


Figure 2.4: Structure of BPYME.

Dimethylformamide (DMF) has been proposed as a solvent that allows for concerted two-electron redox reactions, as seen with other BPY derivatives.[50,51] Through the use of cyclic voltammetry, the redox reactions of BPYME and its charged forms were able to be identified. For these reactions, DMF was used as the solvent and paired with either tetrabutylammonium hexafluorophosphate ($TBAPF_6$) or lithium bis(trifluoromethanesulfonyl) (LiTFSI) as the supporting electrolyte/salt. When pairing DMF with $TBAPF_6$, the reduction of BPYME to its dianion state, $BPYME^{2-}$, occurred as two sequential one-electron reduction reactions at two different potentials. As for the pairing of DMF with LiTFSI, a concerted two-electron oxidation reaction occurred from BPYME to its dication state, $BPYME^{2+}$, and for the reduc-

tion reaction of BPYME to $BPYME^{2-}$, a single two-electron process was observed. Overall, it was shown that ester-substituted bispyridinylidenes are able to exhibit concerted two-electron redox processes as seen with BPYME through the use of DMF solvent and LiTFSI as a supporting salt. Through the study of these materials for two-electron processes, higher energy density systems may be achieved.[13]

2.6 Conventional Energy Storage Methods

The alternative technology of RFBs must be compared to other more conventional energy methods.

2.6.1 Li-ion Batteries

Many battery systems are designed for multiple uses, which can range from small electronics to large-scale energy storage capabilities. One of the most advanced and widely known type of battery is the Li-ion battery. On a large scale, Li-ion batteries can be used in renewable energy sources such as solar and wind power. In the case of solar power, excess solar energy that is generated during daylight hours is stored in Li-ion batteries, which is then used during the night. In order to use Li-ion batteries for grid energy storage applications, they must be produced as large-scale Li-ion batteries at low cost.[52]

Li-ion batteries have a flexible design, which allows them to be modelled in a variety of shapes and sizes to accommodate the size needed for specific applications.[53] A basic Li-ion cell can be seen in Figure 2.5 below, made up of a cathode and anode, which are the positive and negative electrodes, respectively. These electrodes are connected by an electrolyte that contains Li-ions. A separation membrane, usually a microporous polymer membrane, keeps the electrodes isolated from each other so that there is an exchange of Li-ions between the cathode and anode but not an

exchange of electrons, similarly to the ion-selective membrane of the RFB.[52]

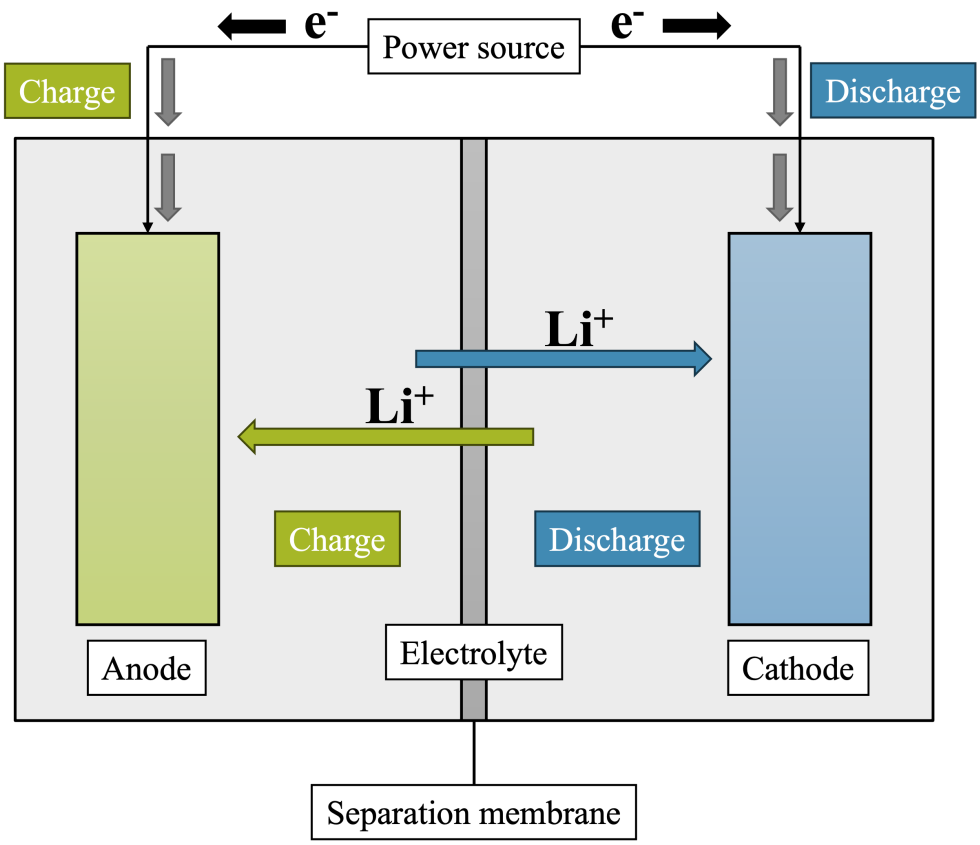
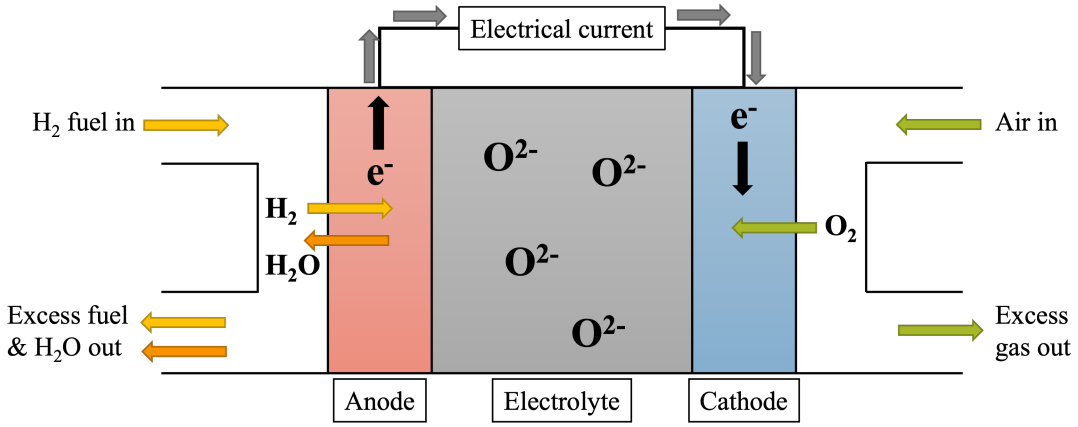


Figure 2.5: Typical Li-ion battery cell for a charge-discharge cycle.

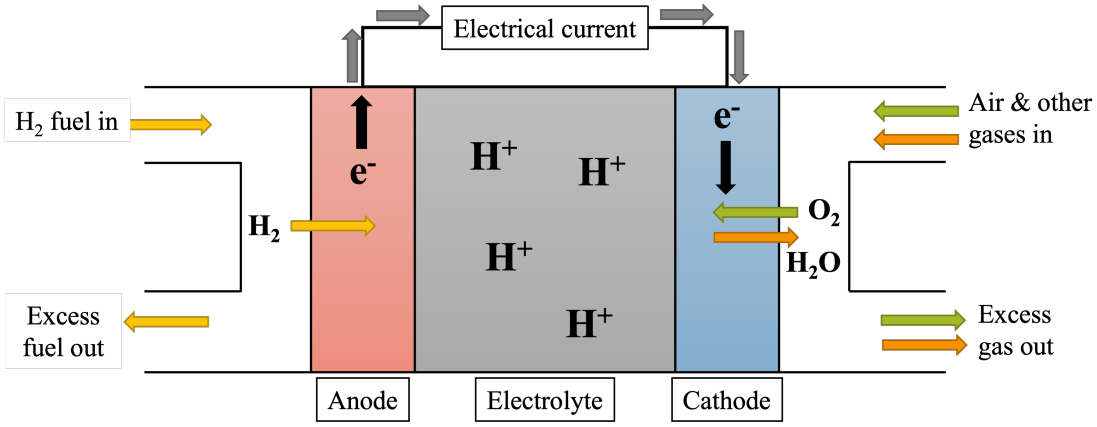
Li-ion batteries have been considered as an option for use in sustainable energy grids where they could store the energy generated from various renewable energy sources, however, this would require the cost of the Li-ion battery system to be greatly reduced.[54] Li-ion batteries may be one of the most common and advanced batteries, however, they still leave much room for improvement in areas where other battery types may be at an advantage. While they are ideal for use in small electronics such as phones and laptops due to their high energy density, they have not yet been optimized for large-scale grid energy storage as the solvents used in the electrolyte of Li-ion batteries are flammable, leading to a risk of fire in their applications.[55]

2.6.2 Fuel Cells

Another conventional energy storage system are fuel cells. As an energy storage method, fuel cells are able to efficiently convert chemical energy into electrical energy by means of electrical currents.[56] There are different types of fuel cells that all exhibit different properties and requirements for energy storage. Some types of fuel cells require a supply of pure hydrogen at the anode of their fuel cell stack. In order to incorporate hydrocarbon or alcohol as fuel, these fuel cells require an external fuel processor, making them not as ideal since this raises the complexity of the system, which in turn raises its cost and lowers its efficiency. It is possible, however, to raise the efficiency of these energy storage systems by having the fuel-processing reaction occur inside the fuel cell stack instead of externally. This makes the fuel cell more efficient by being able to manage the thermal features of the fuel stack itself.[57]



(a) Solid Oxide Fuel Cell



(b) Hydrogen Fuel Cell

Figure 2.6: Structures of two types of fuel cells.

Other types of fuel cells are solid-oxide fuel cells (Figure 2.6a) and molten carbon fuel cells. These fuel cells have high operating temperatures, making them advantageous since both CO and H_2 gas at the anode are able to be electrochemically oxidized.[57] Hydrogen has been looked at as an attractive fuel source for these fuel cells (Figure 2.6b), however, they do pose more technical and economic issues. For example, in order to achieve the lowest costs of hydrogen production for the amounts necessary for large-scale fuel cells, steam reforming of natural gas would need to be applied, creating large greenhouse gas emissions that should be avoided.[58] Unfortunately because of this, it is not likely that hydrogen will be a plausible fuel source

for energy storage in fuel cells unless new methods for the production of hydrogen are developed, leading to the need for hydrocarbons and alcohol fuels for use in fuel cell technologies instead.[57]

2.6.3 Gravity Energy Storage

One last widely used method for storing energy is a type of mechanical energy storage known as gravity energy storage. There are two types of gravity energy storage: wet gravitational energy storage and dry gravitational energy storage. The most used bulk energy storage method is pumped hydroelectric storage (PHES), which is a type of wet gravitational energy storage. PHES is especially useful for long-term and large-scale energy storage. This method involves using two reservoirs where one reservoir is located above the other. The higher reservoir contains water that is used to store the energy in the form of potential energy. During non-peak time, the water is pumped into the upper reservoir and released into the lower reservoir as demand increases to drive the generation of electricity when needed. PHES has some advantages, one of which is its ability to be scaled based on energy demand. Some other advantages are that it is a good long-term energy storage system as well as being highly efficient throughout the whole process. This method does however face some potential setbacks. PHES exhibits low energy density due to the gravity of Earth being a relatively weak force that in turn requires PHES systems to have a large height difference between the two reservoirs or a large body of water in which large quantities of energy may be stored.[59]

A second type of wet gravitational energy storage is underground pumped hydroelectric storage (UPHES), which is typically used when the topography is not ideal for employing PHES. With UPHES, the higher reservoir is placed above ground whereas the lower reservoir is placed underground, directly below the higher reservoir. UPHES is able to ensure that there is a large enough height difference between

the reservoirs to allow for less surface area requirements.[60]

The second type of gravity energy storage method is dry gravitational energy storage that uses the same principles as wet gravitational energy storage, but is able to do so without the use of water. One method is gravitricity, which involves using vertical height in order to raise or lower a mass through a shaft where a system of cables is used to lift the mass. Gravitricity may be used in combination with a compressed air energy storage component that seals the shaft to make use of pressure and increases the amount of energy that can be stored.[61]

2.6.4 Comparison with RFBs

Researchers have been looking into large-scale energy sources and their methods of energy storage as an alternative to the use of fossil fuels as an energy source. While conventional energy storage methods have their advantages in some applications, their disadvantages have led researchers to explore more suitable alternative energy storage systems.[62–64] RFBs exhibit many advantages as they are able to effectively combine energy efficiency, capital costs, and life cycle costs when compared to conventional energy storage systems. Gravitational energy storage is comparable to RFBs since they are both shown to exhibit comparable power and energy densities, which allow them to be used in similar applications. Overall, RFBs and conventional energy storage systems share some similar features however different types of large scale energy storage systems can be advantageous for different applications since there is no known universal solution for large scale energy storage as of yet.

Chapter 3

Machine Learning and Genetic Algorithms

3.1 Introduction

Machine learning (ML) is a type of algorithm used to make predictions about data by learning the patterns that exists within this data. It is a large development in many areas of research since it is a technique that is powerful enough to perform tasks on a standard computer, which may require an exhaustive amount of computational power otherwise in a more throughput approach. One method of applying ML as seen in this project is through the use of a genetic algorithm (GA).

3.2 Machine Learning Theory

With ML, the effectiveness of the learning algorithm is often reliant on both the nature and characteristics of the available data and the overall performance of the learning algorithm itself. There are different types of learning algorithms, and choosing the right one can depend on the specific application the ML algorithm is being used for.[65]

There are four types of ML algorithms:

1. **Supervised learning** uses labelled training data in order to learn or train a function that is able to map an input to an output.[66] This type of ML algorithm is used when there are specific goals identified from a given set of inputs.[67] Most often supervised learning is used for classification tasks to separate data as well as for regression tasks to fit data.

2. **Unsupervised learning** is applied to analyze data without any outside human assistance.[66] Unsupervised learning is mainly used to identify trends and results among data and is typically used in tasks that involve clustering and association of data.

3. **Semi-supervised learning** is a hybrid learning algorithm of both supervised and unsupervised learning. It combines both methods by using labelled and unlabelled data[66,67] in order to operate under supervised and unsupervised conditions. The main goal of using semi-supervised learning as a ML algorithm is to provide better prediction results using both labelled and unlabelled data than results produced from using only labelled data.

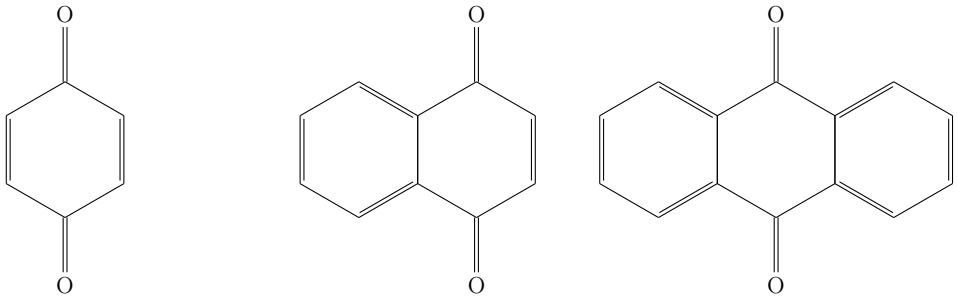
4. **Reinforcement learning** works on the principle of applying a reward or penalty depending on the outcome of the algorithm.[14] It is able to evaluate the behaviour of the algorithm automatically depending on its environment with the goal of improved efficiency of the algorithm.[15]

Overall, these four different types of ML algorithms can be classified by what drives their method of learning. Supervised learning is a *task*-driven method, unsupervised learning is a *data*-driven method, semi-supervised learning works with both labelled and unlabelled data,[68] and reinforcement learning is an *environment*-driven method due to its ability to assess its environment and adjust its behaviour to opti-

mize the efficiency of the algorithm.[15] In order to effectively use each of these methods, it is important to understand how each method works and for which application each is best suited for.[68] Supervised ML is an appropriate method for predicting features of molecules such as the energy gaps between their highest occupied molecular orbital (HOMO) and lowest unoccupied molecular orbital (LUMO),[69] however, reinforcement ML is best suited for finding the best solution to a problem which in this project is the search for the best organic molecule candidates for use as active materials in non-aqueous RFBs.

3.3 Machine Learning Applications for RFBs

There are many ways that ML can be applied to a given problem. In this project, ML was applied through the automated generation of molecules followed by the use of a GA to attempt to further improve the set of molecules. While ML can be useful in the discovery of compounds for use in RFBs as observed in this project, Wang *et. al*[70] made use of ML in a different way while studying RFBs. They applied supervised ML algorithms in order to make predictions on the redox potential difference and solubility of quinone active materials for RFBs by using the positive correlation that these parameters have with the HOMO-LUMO gap and solvation free energy, respectively. They first used quantum mechanics and density functional theory (DFT) to calculate the HOMO-LUMO energy gaps and solvation free energies of various substituted quinone molecules constructed from benzoquinone, napthoquinone, and anthraquinone backbones as shown in Figure 3.1.



(a) Benzoquinone (b) Napthoquinone (c) Anthraquinone

Figure 3.1: Structures of three quinone backbones.

Next, a database was created from the computationally predicted HOMO-LUMO gaps and solvation free energies to be used for training two prediction models, eXtreme Gradient Boosting (Xgboost)[71] and Attentive Fingerprints (Attentive FP).[72] As a prediction model, Xgboost makes use of an ensemble of multiple simple base learners (or weak supervised ML models) that are able to minimize the difference between the overall predicted values and the actual values. It does this by continuously generating new base learners that are able to learn the differences in the predicted values of the previous learner and the desired target. Attentive FP on the other hand adds an attention mechanism at the atomic level while still conserving the intrinsic structure of the molecule to learn the features of the molecule and then it will add an attention mechanism at the level of the entire molecule where it will learn the features of the entire molecule's structure.[72] After applying both of these prediction models, the researchers found that Attentive FP gave poor predictions of HOMO-LUMO gaps, leaving Xgboost as the better method for predicting HOMO-LUMO gaps. They attribute the better performance of Xgboost to its feature selection since it will assign lower weight to repetitive features. As for solvation free energies, Attentive FP was slightly better at making predictions than it was for HOMO-LUMO gaps, however, Xgboost was still the better predictor. Overall, they concluded that Xgboost can be used as a prediction model for determining HOMO-LUMO gaps and solvation free energies instead of computationally expensive DFT

computations.[70]

3.4 Genetic Algorithms

Genetic algorithms (GAs) are a type of reinforcement ML algorithm that makes use of the principles of evolutionary theory to solve optimization problems for data generation. The nature of GAs is to randomly direct search algorithms toward the best possible solutions by utilizing evolutionary mechanisms such as natural selection, natural genetics, and survival of the fittest. The general purpose of a GA is to create a population of potential solutions to a problem that is updated generation to generation so that after each generation, the population should get closer to the optimal solution of the problem that is being studied. Because supervised ML only predicts properties of molecules but not which molecule has the best properties, reinforcement ML is applied instead for this project through the use of a GA. GAs offer many advantages over other possible search algorithms. They can be applied to an extensive list of tasks due to its easy implementation and are fairly resistant to changes in the conditions of the optimization problem before it reaches a solution, allowing the GA to adapt dynamically to the conditions over time. Lastly, one of the more preferable advantages to using GAs is their ability to solve optimization problems that have no preexisting solution.[16]

GAs implement natural evolution by making use of the principles of selection, crossover, and mutation.

1. **Selection** is a genetic operator that is performed on a set of data to select the individual data points that will be used as the population for the crossover operation.

2. **Crossover** is used to create new data points by combining properties of two individuals to generate a new set of data that ideally has been improved from

the original population.

3. **Mutations** help to drive evolution by allowing for the generation of new solutions that may not have otherwise been discovered by only random selection and crossover.[16]

To understand more about how selection, crossover, and mutation operators may be applied to GAs, the typical flow of a GA is shown in Figure 3.2 below.[16]

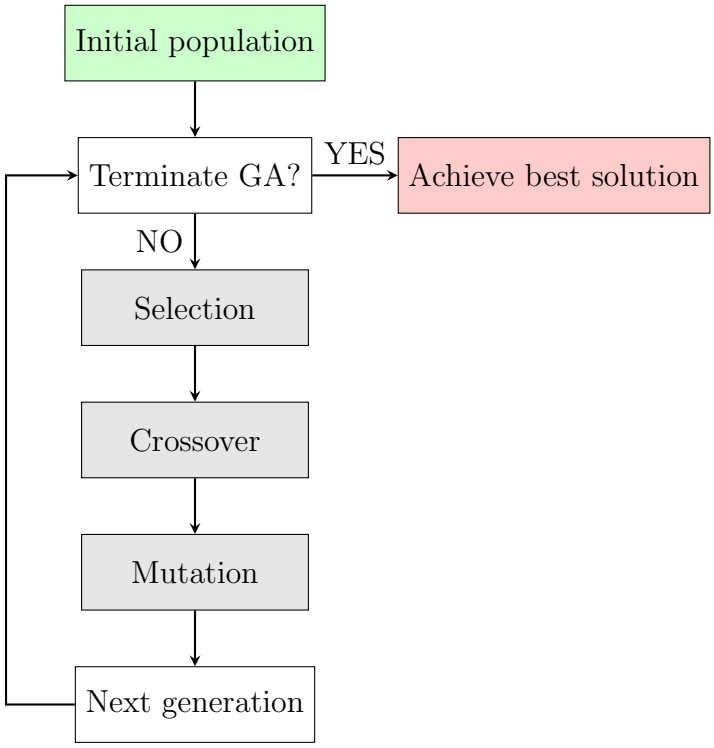


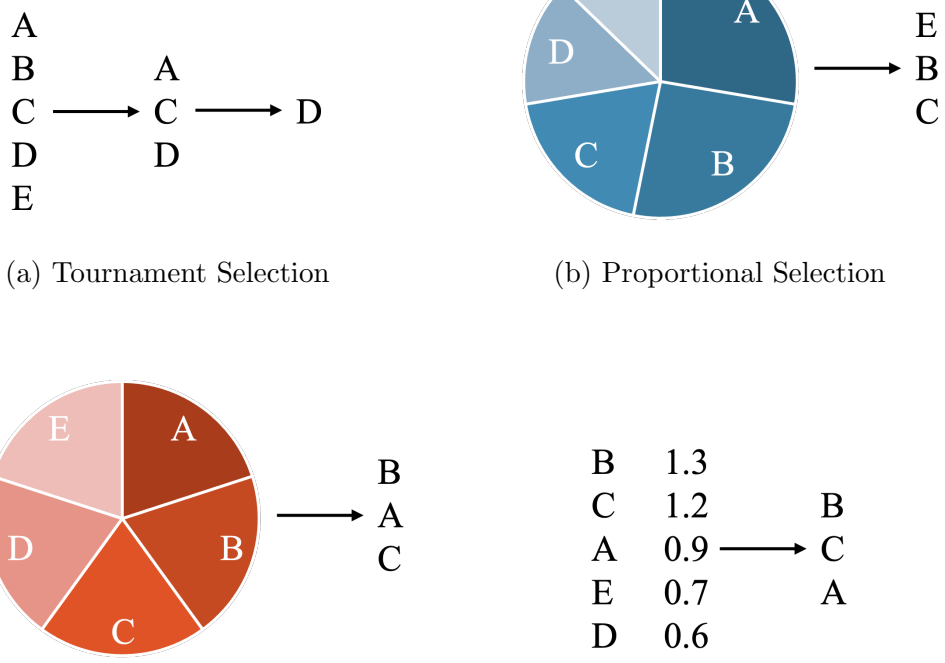
Figure 3.2: Typical flow of a GA.

This GA flow will continue for multiple generations until an optimal solution is achieved or it is decided to stop the GA.

3.4.1 Selection

The typical flow of a GA starts with the selection of a random initial population. This initial population is made up of individuals that represent possible solutions

to the problem that is being optimized. There are a few methods for determining which individuals will be chosen for the population. Typically, a fitness function, or "score" function is applied to determine the best individuals based on the idea of survival of the fittest where the individuals that are best suited to their given constraints are able to maximize or minimize their fitness function. Selection is then applied using the fitness function to determine the best individuals for the initial population. There exists multiple methods of selection for GAs, as seen in Figure 3.3, all of which have different advantages for different applications.[16]



(a) Tournament Selection

(b) Proportional Selection

(c) Stochastic Universal Sampling

(d) Rank Selection

Figure 3.3: Examples of different selection methods for GAs.

1. **Tournament selection** selects a subgroup, typically made up of two to three individuals from a population, then selects the best individual from that subgroup.

2. **Proportional selection** assigns each individual a section of a roulette wheel

based on their relative fitness values so individuals with higher fitness will have a larger section of the wheel and therefore a higher likelihood of being selected for the population.

3. **Stochastic universal sampling** is a similar selection method to proportional selection in that it also makes use of the roulette wheel, however, it divides the wheel into equal sections instead, independent from the relative fitness values. This method of selection helps to even out a level of bias that proportional selection may have.

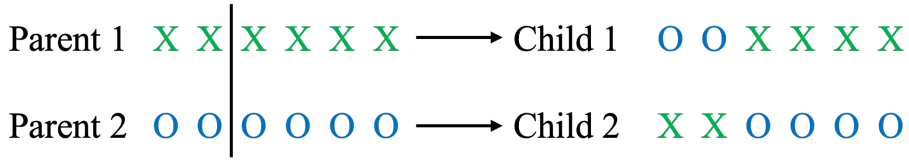
4. **Rank selection** makes use of the same principles as proportional selection and ranks individuals by their fitness values from most to least fit, assigning a number to each individual, placing it in between the highest and lowest fitness value individuals in a list. This method helps to level out the large differences that exists between individual high and low fitness values.

5. **Elite selection** can make use of the above mentioned methods, however it guarantees that the best individual will always be selected for a given population. While it is not entirely necessary to apply elite selection, it may become useful when fitness values decrease over generations as a result of crossover or mutations.[16]

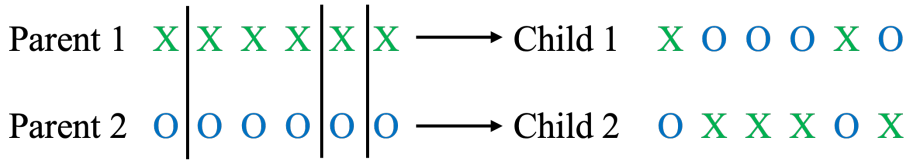
In this project, two of these selection methods were used for two separate steps of the GA. The first was the use of rank selection (Figure 3.3d) to select the molecules that make up the population for the GA. The second was stochastic universal sampling (Figure 3.3c) to randomly choose the parent molecules to be crossed over in the GA.

3.4.2 Crossover

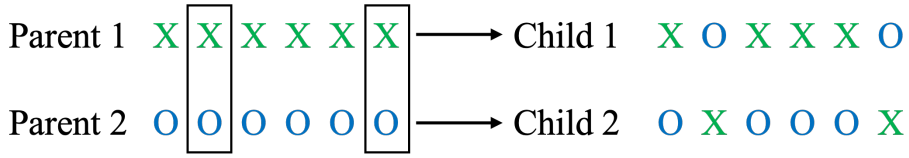
Crossover is the first step that creates new individuals in a GA. Crossover allows for the combination of traits of two individuals from the current generation, called "parents", to create new individuals for the next generation, called "children" or "offspring", while still maintaining these traits throughout. Three crossover methods have been shown in Figure 3.4.[16]



(a) One-point Crossover



(b) N-point Crossover



(c) Uniform Crossover

Figure 3.4: Examples of different crossover methods for GAs.

1. **One-point crossover** divides a gene sequence into two parts and creates new individuals by crossing over these divided sequences, swapping the genes of one part of the sequence.

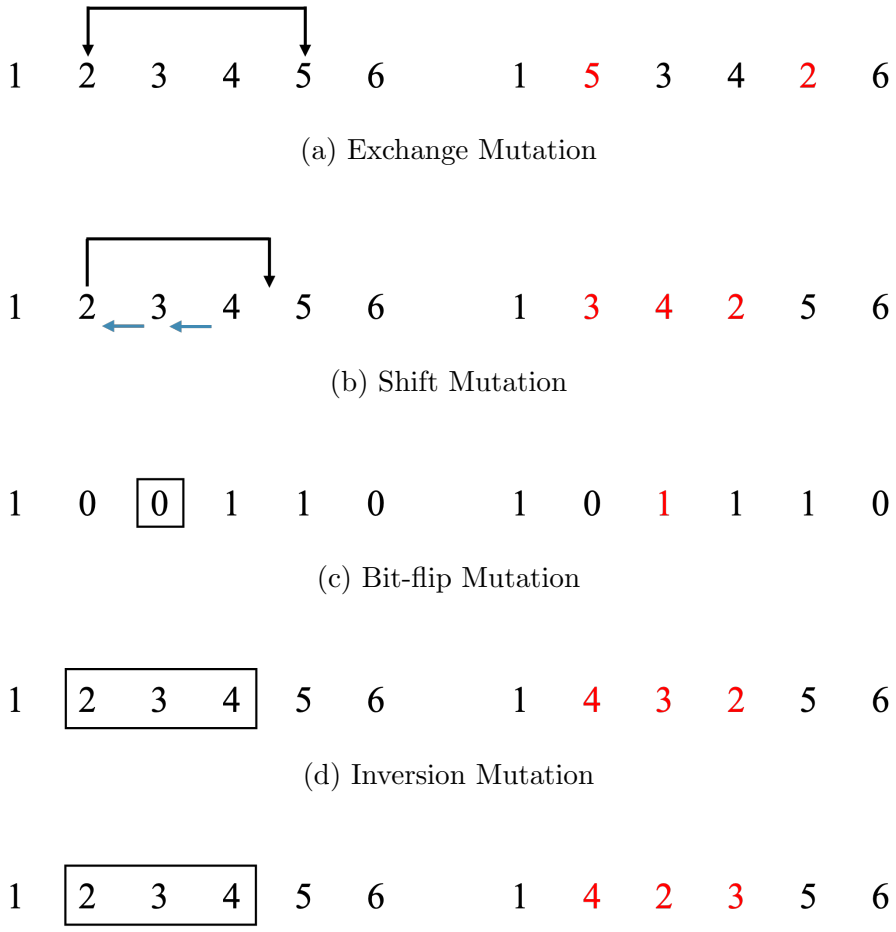
2. **N-point crossover** follows the same principles as one-point crossover except the gene sequence is divided into N parts and the genes are crossed over in a criss-cross manner.

3. **Uniform crossover** will randomly select points along the gene sequence of parent individuals and then swap the genes only at these points.[16]

In this projects, uniform crossover was used with the added caveat that the offspring may not be chemically meaningful. To fix this, mutations were applied.

3.4.3 Mutations

Mutations are a driving force of evolution. While they are typically assumed to alter genes in a negative way, mutations may actually give a chance for the GA to introduce positive changes to an individual to help improve this individual and provide it with new features that may help it to adapt to its environment. There are a few different types of mutations, each with a different application depending on the type of genes that are being presented, examples of which are shown in Figure 3.5.[16]



(a) Exchange Mutation

(b) Shift Mutation

(c) Bit-flip Mutation

(d) Inversion Mutation

(e) Shuffle Mutation

Figure 3.5: Examples of different mutation methods for GAs.

1. **Exchange mutation** may be applied to binary or ordered sets of genes. With this mutation method, a pair of genes is randomly selected to exchange positions.

2. **Shift mutation** allows for a gene to be moved to a randomly selected position in the sequence of genes, which will shift genes to either the left or the right, depending on where in the sequence the gene has been moved to.

3. **Bit-flip mutation** may only be applied to a binary set of genes because it will flip a random binary gene from either 0 to 1 or 1 to 0.

4. **Inversion mutation** may be applied to binary or ordered set of genes where

a random sub-range is chosen and the order of genes within it is inverted.

5. **Shuffle mutation** may be applied to binary or ordered set of genes and a random sub-range is also chosen, however, the order of genes within it is randomly shuffled.[16]

In this project, mutations were not explicitly considered but instead employed to fix chemically invalid structures.

3.4.4 Tuning GA Hyperparameters

When designing a GA, there is more to consider than just the selection process, crossover, and mutations. One might also consider how these different processes may be tuned to better suit the GA design or given application. Some hyperparameters that may be modified in the GA design are population size, crossover probability, and mutation probability. It is known that larger population sizes are better for GAs as they lead to a higher probability of finding the best solution to a given problem, however, maintaining a larger population can become difficult as the population size increases. Therefore, it becomes beneficial to determine the smallest population size that will give the best solution(s). Unfortunately there is no clear method for determining the ideal population size, but there are some ways to help narrow it down. One way is through the idea that at the start, individuals in a population will be randomly distributed and as more individuals drift toward the ideal or best solutions, the entire population will try to shift toward that point as well. This is why a smaller population size may be beneficial at times since smaller populations can move much faster than larger population sizes would be able to.[16]

Crossover probability can be understood by referencing Figure 3.6. Once individuals make it to the crossover process, there are two probable events. Two individuals with crossover probability p will undergo crossover where they will generate two new

individuals that are hopefully even closer to the desired solution. With crossover probability $1 - p$, individuals will self-clone instead of crossing over, leading to identical offspring as the original individuals. Self-cloning can be useful for allowing good individuals to make it to further generations without the possibility of losing features to crossover. Overall, crossover probability defines the degree to which a population may vary, where if the crossover probability is high, the population has more desire to create new solutions.[16]

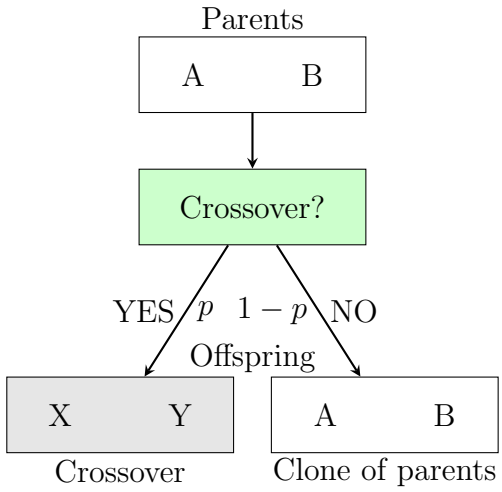


Figure 3.6: Scheme depicting the flow of crossover probability.

Mutation probability is described similarly as seen in Figure 3.7 where it describes the probability at which an individual may be mutated in its new population and the degree to which the population will deviate from the original. An individual with mutation probability p will have a mutation applied to it while an individual with mutation probability $1 - p$ will be a copy of the original individual. Too few mutations will not help the GA to find the best solution to complex problems while having too many mutations in a population will lead to disruption of the evolution process.[16]

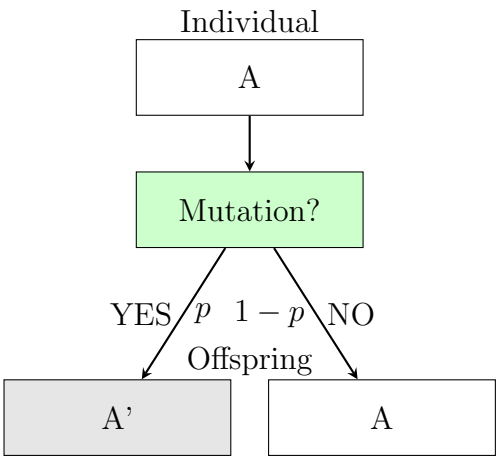


Figure 3.7: Scheme depicting the flow of mutation probability.

Overall, GAs require the best suited combination of the hyperparameters of population size, crossover probability, and mutation probability to best perform its search for the optimal solutions to the given problem, which are often obtained through experimentation.

Chapter 4

Computational Methods

4.1 Introduction

This project combined computational chemistry and ML methods for generating and identifying a set of organic molecules as potential redox-active materials for use in non-aqueous organic RFBs. The random generation of molecules for this data set was automated using a GA and DFT calculations to compute the redox capabilities of the generated molecules to discover a set of organic molecules with the highest cell potential, stability, and solubility. For each molecule, the necessary Gibb's free energies were computed to extract the reduction and cell potential, stability, and solvation free energy. The molecules are then ranked based on these values using an appropriate scoring function to determine the population for the GA. The described processes were automated in a continuous workflow, minimizing human intervention. As a result, the project required a large amount of scripting, ensuring that all file formats were compatible moving from one task to another (see Appendices A and B).

4.2 Bipyridine

In a first, initial stage of this project, different substituted BPY derivatives were studied by substituting the rings of the BPY backbone as shown in Figure 4.1 where RR'BPY^{2-}, RR'BPY, and RR'BPY^{2+} are the dianion, neutral, and dication states, respectively. Although substitutions can be made at any position on the rings, all substitutions made to this BPY backbone in the initial stage of the project were at the *para* position to the nitrogen atom of each ring to allow for the least amount of steric hindrance from the rings. This feature was later changed to any position of the ring to allow for a larger search area of molecules.

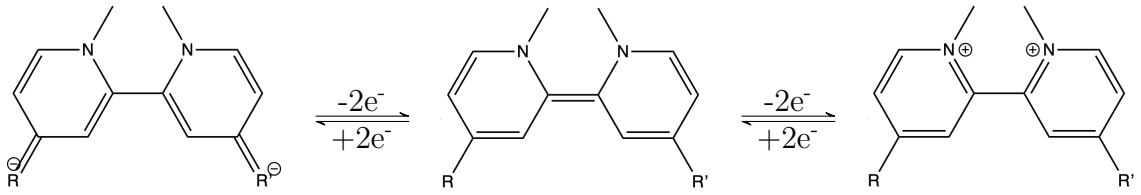


Figure 4.1: Reaction scheme of a BPY derivative where electrons are transferred between its neutral and charged states.

After first studying substituted BPY derivatives as shown in Figure 4.1, this project was then shifted towards a more expansive search area by allowing for more freedom in the construction of the backbone itself. Overall, the initial study of BPY gave a good foundation to lead into the expansion of the search area of organic redox-active materials. The backbone structures of the randomly generated molecules described in Chapter 4.7 were based on the general structure of BPY since BPY derivatives have shown to be good candidates for use in RFBs.[13]

4.3 Computational Set-Up

The properties of interest for determining the redox capabilities of the generated organic molecules as redox-active materials are cell potential, stability, and solubil-

ity. The key for calculating each of these parameters are thermodynamic properties, such as Gibb's free energy (G), which is computed for the neutral molecule in the gas phase and in solvent (DMF), as well as for the dication and dianion states in DMF. After the structure of each molecule was generated in the form of a MDL Molfile (`.mol` file) and its geometry optimized with the MMFF94 force field (Merck molecular force field)[73] using RDKit,[74] the `.mol` file for each molecule is then converted to a GAMESS (General Atomic and Molecular Electronic Structure System)[75] input file using Open Babel (obabel)[76,77] in Python. These file are then run through Compute Canada to compute the Gibb's free energy of each molecule for each of the above described states. From there the necessary change in Gibb's free energy (ΔG) can be calculated to be used to calculate the reduction potentials ($E_{2+/0}$, $E_{0/2-}$), cell potential (E_{cell}), stability ($\Delta G_{stability}$), and solubility through solvation free energy ($\Delta G_{solvation}$).

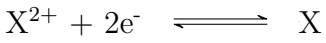
4.4 Redox Properties

A Hessian calculation was performed for each of the neutral, dication, and dianion states at the B3LYP/cc-pVDZ DFT level to calculate the Gibb's free energy of each molecule in its different states. This method was chosen as it is a relatively fast method of performing DFT calculations for all states of interest and includes electronic and molecular energies (such as zero-point vibrational energies). The B3LYP functional is an example of a hybrid method which includes Becke-88 (B88) and Hartree-Fock (HF) exchange, the Lee-Yang-Parr (LYP) correlation functionals, and three parameters that are determined by fitting them to experimental data.[78] The accuracy of the B3LYP functional has been debated,[79] however, it must be taken in mind that in this project, relative measures of performances are being used to rank molecules, not to evaluate values to compare with experiments for which more

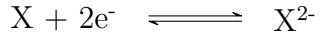
focused studies are needed. For the calculations performed on the neutral molecule in the gas phase and all states in DMF as the solvent, the following specifications were used.

The self-consistent field (SCF) type was set to perform a Restricted Hartree-Fock (RHF) calculation since there is an even number of electrons in these molecules and therefore no unpaired electrons, leading the multiplicity of the electronic state to be set to 1, a singlet. The basis set used was the doubly polarized Dunning-Type cc-pVDZ[80] (correlation consistent polarized double-zeta) basis set. To perform calculations in solvent, additional solvent parameters are required. As mentioned, the solvent used for these calculations was DMF, which is incorporated in the PCM[42] implicit solvation model. For DMF, the SMD[43] implicit solvation model must also be applied, which adds corrections for cavitation, dispersion, and solvent structure. The computation of the Hessian matrix for the zero-point energy corrections is done semi-numerical, i.e. numerical differentiation of the analytically computed first derivatives. All computations were run on ACENET.

(a)



(b)



(c)

Figure 4.2: Generalization of the redox reactions observed for the generated organic molecules. (a) Overall reversible redox reaction (b) Reduction reaction of the dication (X^{2+}) to the neutral (X) molecule (c) Reduction reaction of the neutral molecule to the dianion (X^{2-}).

Figure 4.2 shows the redox reactions that the generated molecules may undergo. Each state is assumed to undergo concerted two-electron transfer reduction reactions.

In the overall reaction seen in Figure 4.2a, the neutral molecule (X) undergoes a reduction reaction to the dianion state (X^{2-}) by gaining two electrons, therefore X is reduced to X^{2-}. On the opposite side, X is oxidized to the dication state (X^{2+}) by losing two electrons. To calculate the reduction potentials, both reactions must be written as reduction reactions as seen in Figure 4.2b and 4.2c which show the reduction reactions for the dication to neutral state, and the neutral to dianion state, respectively. The reduction potential associated with the redox reaction is given by the Nernst equation (Eq. 4.1) with ΔG the change in Gibb's free energy of the cell, n the number of electrons transferred during the reduction reaction, Faraday's constant (F) given as 96.485 kC/mol, and E the reduction potential given in V.

$$\Delta G = -nFE \tag{4.1}$$

The DFT computation yields the final Gibb's free energy of the molecules. This value contains four components: one dominant electronic component ($E_{elec.}$), and three smaller temperature dependent components, vibrational ($E_{vib.}$), rotational ($E_{rot.}$), and translational ($E_{trans.}$). The following equations, Eqs. 4.2 and 4.3, were used for calculating ΔG of the reduction reaction from the reduced species to the oxidized species in kJ/mol.

$$\Delta G_{2+/0} = G_0 - G_{2+} \tag{4.2}$$

$$\Delta G_{0/2-} = G_{2-} - G_0 \tag{4.3}$$

Once ΔG has been calculated, it is then substituted back into Eq. 4.1 to give Eqs. 4.4 and 4.5 below to determine the reduction potential for the two reduction reactions. Since $n = 2$ electrons are transferred and introducing Faraday's constant, they can be written as a combined constant in Eqs. 4.4 and 4.5. For the molecules

of this project, $E_{2+/0}$ is desired to be as large as possible while $E_{0/2-}$ as small but positive as possible.

$$E_{2+/0} = \frac{-\Delta G_{2+/0}}{192.97 \text{ kC/mol}} \tag{4.4}$$

$$E_{0/2-} = \frac{-\Delta G_{0/2-}}{192.97 \text{ kC/mol}} \tag{4.5}$$

From the reduction potentials, the overall cell potential of the molecule may be calculated as

$$E_{cell} = |E_{2+/0} - E_{0/2-}| \tag{4.6}$$

By having $E_{2+/0}$ as large as possible and $E_{0/2-}$ as small as possible, E_{cell} can be maximized.

4.4.1 Solvent Window

It is important to consider the electrochemical properties of the solvent, as the applied voltage during charging of the cell may charge/discharge the solvent if it is too high. Therefore, the electroactive material must have have an electrochemical potential that is within the window of solvent charging/discharging.[81] The solvent being used in this case was DMF (Figure 4.3).

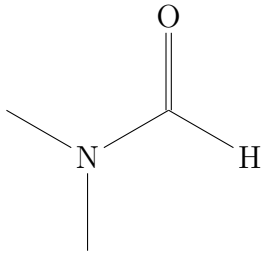


Figure 4.3: Structure of DMF.

The potential window for DMF was calculated similarly as described in Chapter

4.4, where the redox properties are computed for the reduction and oxidation of DMF to the respective positive and negative states.

For DMF, ΔG is calculated similarly to Eqs. 4.2 and 4.3 except only a single electron transfer is involved during the reduction reactions of DMF, which is accounted for in Eqs. 4.7 and 4.8 with $n = 1$.

$$E_{+/0,DMF} = \frac{-\Delta G_{+/0,DMF}}{96.485 \text{ kC/mol}} = 6.335 \text{ V} \tag{4.7}$$

$$E_{0/-,DMF} = \frac{-\Delta G_{0/-,DMF}}{96.485 \text{ kC/mol}} = -0.797 \text{ V} \tag{4.8}$$

The reduction potentials of DMF are 6.335 V and -0.797 V for the reduction of the neutral DMF molecule to the cation and the anion to the neutral molecule, respectively. These reduction potentials are then used to calculate the overall cell potential of DMF (Eq. 4.9).

$$E_{cell,DMF} = |E_{+/0,DMF} - E_{0/-,DMF}| = 7.132V \tag{4.9}$$

The calculated cell potential for DMF is equal to 7.132 V therefore no calculated value of cell potential may be above this value. Overall, given the calculated values for the solvent window, the accepted values for $E_{2+/0}$ must be less than 6.335 V and the accepted values for $E_{0/2-}$ must be greater than -0.797 V. For simplicity, only $E_{0/2-}$ values greater than 0.000 V will be considered inside the window.

4.5 Stability

An important factor for determining ideal candidates for use as active materials in RFBs is stability. If a compound is not stable for a desired length of time, it will not be a good active material for use in RFBs. Assessing the stability of a compound

in a systematic, automated, and computational manner is very challenging; multiple approaches can be envisioned. One method that was used to assess the stability of the molecules in this project was the removal of substituents from one ring of the molecule to determine if it was favourable for that substituent to be lost or not. In other words, seeing how easy it is for the substituents of these molecules to be lost or removed. There are other methods of determining stability that are possible such as looking at the stability of the ring itself or the linkers that connect the two rings, however these can become difficult to automate therefore the automated assessment of stability of the randomly generated molecules was limited to the ΔG of the removal of FG^+ as shown in Eq. 4.10 where FG is the substituent and R is the ring backbone.

$$\mathrm{FGRRFG} \longrightarrow \mathrm{FGRR^- + FG^+} \tag{4.10}$$

To automate this process, the script used to generate the `.mol` file for each molecule also created a second `.mol` file for the molecule encoding the loss of one of its substituents. Since these are symmetric molecules that are being studied, substituents were only removed off of one ring of the molecule. If the molecule had more than one substituent on each ring, then a `.mol` file is created for the loss of each of the substituents, one at a time. These molecules then undergo geometry optimization using the MMFF94 force field with RDKit[74] and conversion to a GAMESS[75] input file for the neutral molecule in DMF using obabel[76,77] in Python. While the stability should be determined for the dication and dianion states in addition to the neutral state, this was not done for this project due to the extra computational time that would be needed to perform these calculations with the assumption that the neutral state would be the most stable in general.

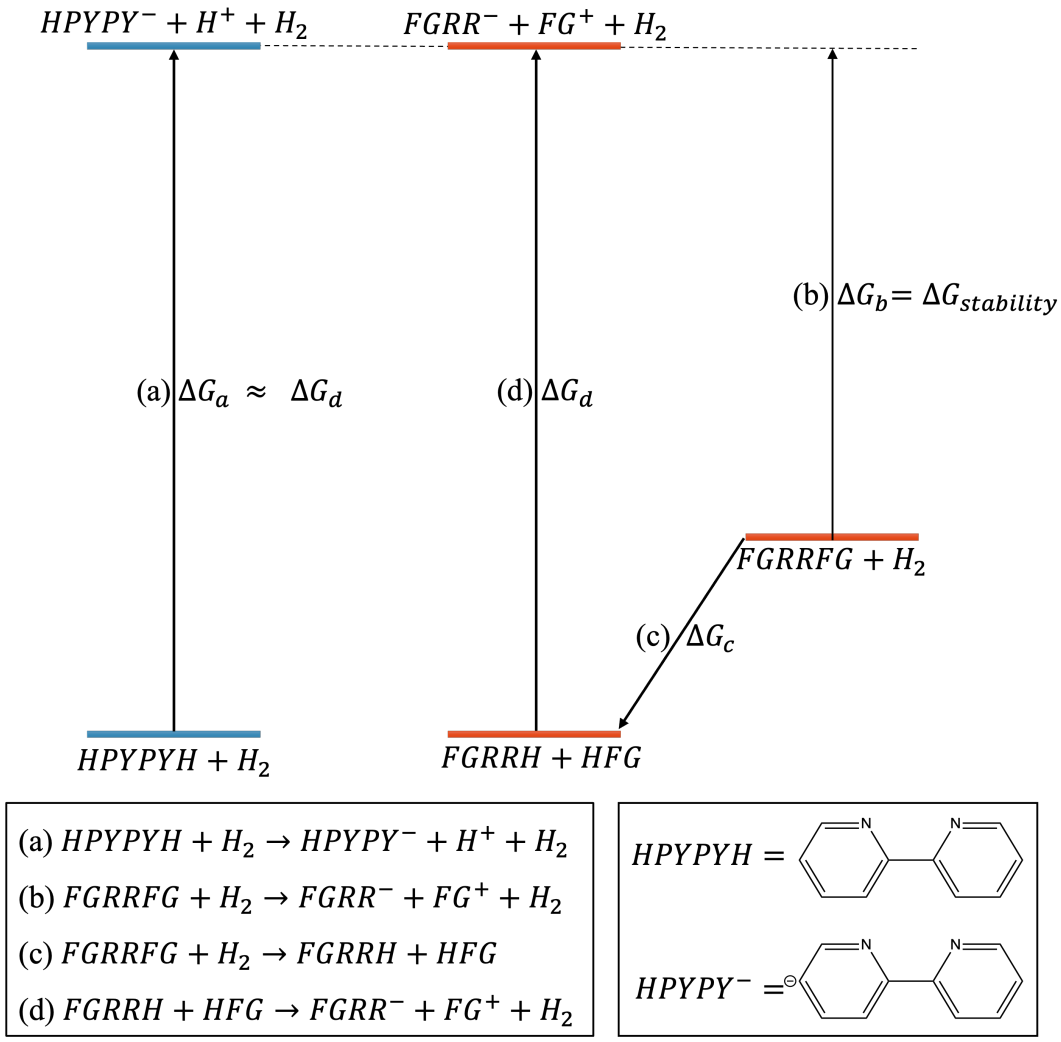


Figure 4.4: Scheme showing the processes for the loss of substituents to determine stability where R = ring backbone of the generated molecule and FG = substituent lost.

Computing ΔG for Eq. 4.10 on an automated level is complicated within RDKit[74] as it works exclusively with `.mol` files, which ensures that all atoms are saturated with hydrogen. During the geometry optimization on the molecule after the loss of a substituent, RDKit[74] automatically replaces that group with a hydrogen atom as seen in reaction (c) (Eq. 4.11) of Figure 4.4.

$$\mathrm{FGRRFG} + \mathrm{H_2} \longrightarrow \mathrm{FGRRH} + \mathrm{HFG} \tag{4.11}$$

This is not favourable since there should be an intermediate step in this process where the ring that has lost the substituent is negatively charged at that position, leaving the lost substituent positively charged as seen in reaction (b) (Eq. 4.12).

$$\mathrm{FGRRFG} + \mathrm{H_2} \longrightarrow \mathrm{FGRR^-} + \mathrm{FG^+} + \mathrm{H_2} \tag{4.12}$$

Therefore with the protonation of both the ring backbone and substituent that occurs during the geometry optimization step for the loss of a substituent, reaction (b) cannot be computed directly. This is where reaction (a) (Eq. 4.13) for the loss of a hydrogen atom from BPY becomes useful.

$$\mathrm{HPYPYH} + \mathrm{H_2} \longrightarrow \mathrm{HPYPY^-} + \mathrm{H^+} + \mathrm{H_2} \tag{4.13}$$

This reaction was used under the assumption that it would be roughly equal to the difference in Gibb's free energy of reaction (d) (Eq. 4.14) in order to determine the ΔG for the loss of a substituent shown as reaction (b) in Figure 4.4.

$$\mathrm{FGRRH} + \mathrm{HFG} \longrightarrow \mathrm{FGRR^-} + \mathrm{FG^+} + \mathrm{H_2} \tag{4.14}$$

The advantage of using this reaction is that it is a one-shot computation that can be carefully computed up front and used as a correction factor. While it is not a direct substitution for being able to calculate reaction (d) directly to determine the energy of reaction (b), and ultimately the stability of the molecule, these reactions allow for a qualitative analysis of the stability of the molecules with respect to the reference BPY molecule. To determine the stability of each molecule ($\Delta G_{stability}$, Eqs. 4.15 and 4.16), ΔG of reactions (a) and (c) of Figure 4.4 were calculated where ΔG_a is the change in Gibb's free energy for the deprotonation of BPY from the neutral to the anion state and ΔG_c is the change in Gibb's free energy for the loss of a substituent followed by the substitution of a hydrogen atom in its place.

$$\Delta G_b = \Delta G_d + \Delta G_c \approx \Delta G_a + \Delta G_c = \Delta G_{stability} \tag{4.15}$$

$$\Delta G_{stability} = 864.277\ kJ/mol + \Delta G_c \tag{4.16}$$

Since the reaction used to determine stability is the loss of a substituent, a negative value for $\Delta G_{stability}$ would indicate that the molecule would undergo the loss of a substituent, making it unstable and unfavourable. It is therefore desired for a molecule to be considered stable, the calculated value of $\Delta G_{stability}$ must be positive. Reversely, the more positive $\Delta G_{stability}$ is, the less tendency the compound will have to lose their functional group.

4.6 Solvation Free Energy

Cell potential is not the only factor determining the redox capabilities of these molecules for use as active materials in RFBs; solubility and stability are also important factors. While solubility is not directly determined in this project, solvation free energy has been calculated which has a generally positive correlation with the solubility of organic molecules.[70] To determine the solvation free energy, calculations were performed for the neutral molecules in the gas phase and DMF. From these calculations, the Gibb's free energy was extracted to calculate ΔG from the gas phase to the solvent.

$$\Delta G_{solvation} = G_{DMF} - G_{gas} \tag{4.17}$$

When looking at the change in Gibb's free energy, specifically the solvation free energy, a negative value will indicate that the reaction will occur spontaneously, which in this case shows that the solvation reaction of a molecule from the gas phase

to solvent will occur. Therefore, for $\Delta G_{solvation}$, a value that is negative is desired in order to show that the molecule will be soluble in the given solvent, while a positive value shows that the molecule is not soluble.

4.7 Automated Generation of Organic Molecules

In order to automate the random generation of organic molecules for use in non-aqueous RFBs, a Python script was written to build the structures of these molecules. The full Python script used can be found in Appendix A. The parameters chosen to build the backbones of these molecules were based on the BPY backbone structure (Chapter 4.2) however there were fewer constraints given to allow for a larger search area and more diverse set of molecules. In brief, the molecules were generated by modifying a `.mol` file for cyclohexane based on the following successive (random) choices made by the Python script (Figure 4.5), generating a unique `.mol` file for each randomly generated molecule.

1. Random number and position(s) of nitrogen atoms in each ring
2. Random number, position, and choice of substituents (1-2)
3. Random number and position(s) of double bonds in each ring
4. Random choice of symmetric linker

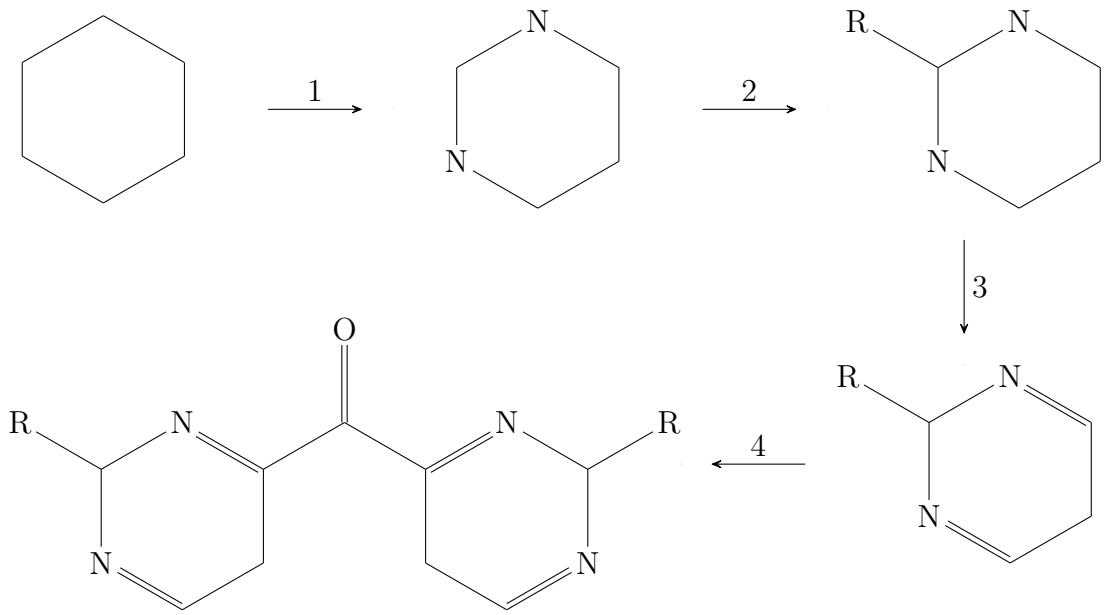


Figure 4.5: Example of the construction of a molecule following the random choices made by the Python script.

A `.mol` file was used to generate the structures of each molecule since this type of file specifies the atoms and their coordinates, as well as the types of bonds that connect them, making the individual structural features easily manipulated for generating the necessary structural files of each molecule.

Once the cyclohexane ring has been modified given the above described parameters, it is duplicated and flipped to be a mirror image of the first generated ring in order to keep the symmetry throughout the entire molecule. The two rings are then connected by a randomly selected symmetric linker. This is just a coarse-grained overview of the flow of choices made to automate the generation of organic molecules, however, fine-grained decisions and modifications were made to this flow of decisions to find the ideal method for building the desired molecules.

4.7.1 Choice of Substituent Functional Groups

The structures of possible functional groups used as substituents are shown in Figure 4.6.

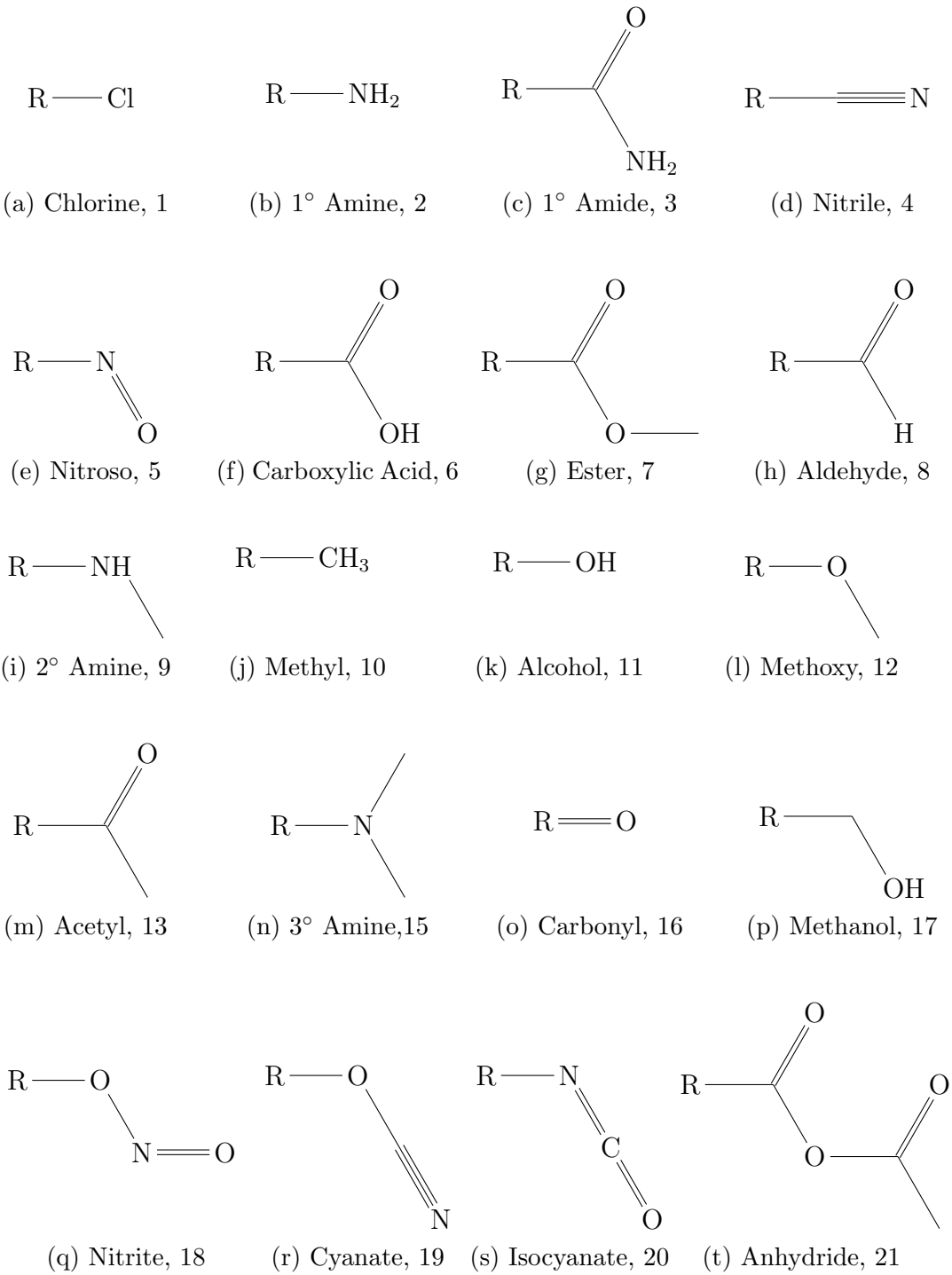


Figure 4.6: Substituents used in the random generation of organic molecules with their corresponding number labels read by a Python script where R = ring backbone.

While these substituents were chosen at random, they all offer different properties which may alter the redox chemistry of the generated molecules. Chlorine

(1) was chosen as a potential substituent to exhibit the electronegativity differences seen from halogen groups. Other substituents were chosen because of their electron withdrawing and electron donating effects. Both EWGs and EDGs have an effect on the redox properties and therefore on the reduction potentials themselves. EWGs have stabilizing effects on aromatic rings, causing them to shift the reduction potential of organic molecules to higher values. On the other hand, EDGs interfere with the entry of electrons to the ring due to the charge density on the aromatic ring. Overall, EWGs will have more of an effect on the reduction potential of molecules than EDGs[33] however it is difficult to predict whether substituting EWGs or EDGs will result in a higher cell potential. The groups shown in Figure 4.6 can also be categorized by their activating and deactivating properties where EWGs are generally deactivating and EDGs are generally activating. Halogens (1) are typically EWGs due to their high electronegativity. Groups like amides (3), nitrile (4), carboxylic acid (6), and aldehyde (8) are moderately deactivating EWGs due to the presence of a polar π bond that is conjugated to the aromatic ring while nitro groups (5) are the most strongly deactivating EWGs due to their strong inductive effects. As for EDGs, groups like amines (2, 9, 15) and alcohols (11) have lone pairs on their heteroatoms, which makes them strongly activating while acetate groups are slightly less activating due to their lone pair that is electron delocalized. Alkyl groups (10) are weakly activating EDGs because of their weak electron delocalization and donation via hyperconjugation.[82]

4.7.2 Structural Constraints

In the Python script designed to automate the generation of organic molecules, the maximum number of nitrogen atoms that were allowed in the initial cyclohexane ring was set to four. This was because it was unlikely that molecules with more than four nitrogen atoms on each ring would be stable. A limit was imposed on the number

of substituents that could be added to each ring because the geometry of molecules with many functional groups proved difficult to optimize. Too many substituents would cause large amounts of steric interactions throughout the molecules therefore a constraint was added to allow only a maximum of two substituents on each ring of the generated molecules. The last constraint that was added was changing the maximum number of double bonds allowed in each ring from six bonds to three. Any number of double bonds in a six-membered ring greater than three leads to at least one *sp*-hybridized carbon atom in the ring caused by the carbon atom being connected to two double bonds, giving a highly unstable molecule as a result. To avoid *sp*-hybridized carbon atoms, the maximum number of double bonds was capped at three bonds. While this constraint did not entirely rule out the possibility of *sp*-hybridized carbon atoms in the generated molecules, it did decrease the chances of them occurring.

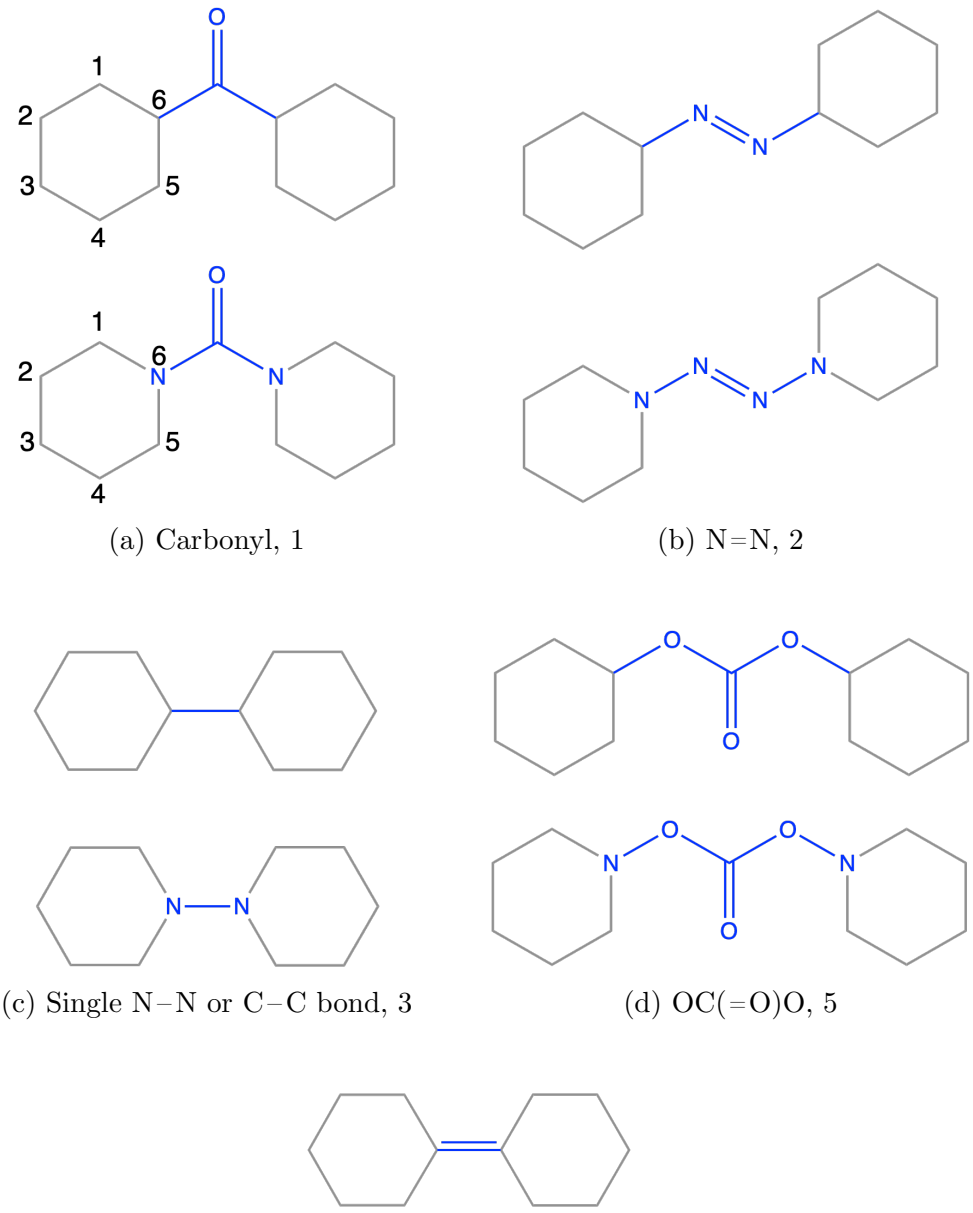
4.7.3 Valency

While building these molecules, the most significant parameter that must be considered in the script is valency. Each atom has a set number of bonds it can make, and a non-constrained script is not able to account for that limit on its own; the script will randomly choose where the nitrogen atoms, double bonds, and substituents will be placed throughout the rings without having chemical knowledge as to whether it is generating viable molecules or not. Therefore, a valency check is added after each step of the molecule generation. This valency check counts the valency at each atom, either three for nitrogen or four for carbon, and then keep track of the valency of each atom in the ring for the subsequent steps as more features are added to the molecular structure. If the valency check identifies any issues with the valency, it will redo that step of the script until valency rules are being followed for the entire structure. This may limit the modifications that can be made to the backbone after each step, but

it ensures that all valency rules are being followed and viable structures are being generated.

4.7.4 Choice of Linker

Figure 4.7 shows the possible linkers that may be chosen to connect the rings of each molecule. These linkers were randomly chosen but all have the feature that they will maintain the overall symmetry of the molecules.



(a) Carbonyl, 1

(b) N=N, 2

(c) Single N−N or C−C bond, 3

(d) OC(=O)O, 5

(e) Double C=C bond, 100

Figure 4.7: Possible choices of symmetric linkers as highlighted in blue with their corresponding number labels read by the Python script.

For the randomly generated molecules, the linkers are always joined at position 6 of the first ring and the corresponding atom of the second ring as shown in Figure 4.7a. Linkers 1, 2, 3, and 5 may be chosen whether the molecule has a carbon or nitrogen atom at position 6 however linker 100 may only be chosen if there is a carbon atom at position 6 since the valency of a nitrogen atom should prefer four

bonds at position 6.

4.7.5 Geometry Optimization

After the `.mol` file for each new molecule is generated, the geometry of these molecules must be optimized. There are a few options for optimizing the geometry of these molecules. One of the more accurate methods would be performing a geometry optimization calculation at the DFT level, however this can become prohibitively time consuming, especially when generating a large data set of molecules as it can take upwards of a few hours per molecule to reach the optimal geometry depending on the size of the molecule. To greatly reduce the time requirements for geometry optimization from hours to seconds, the MMFF94 force field[73] was applied in Python using RDKit.[74]

The flow of decisions made throughout the automated random generation of these organic molecules as described in this section has been summarized in Figure 4.8.

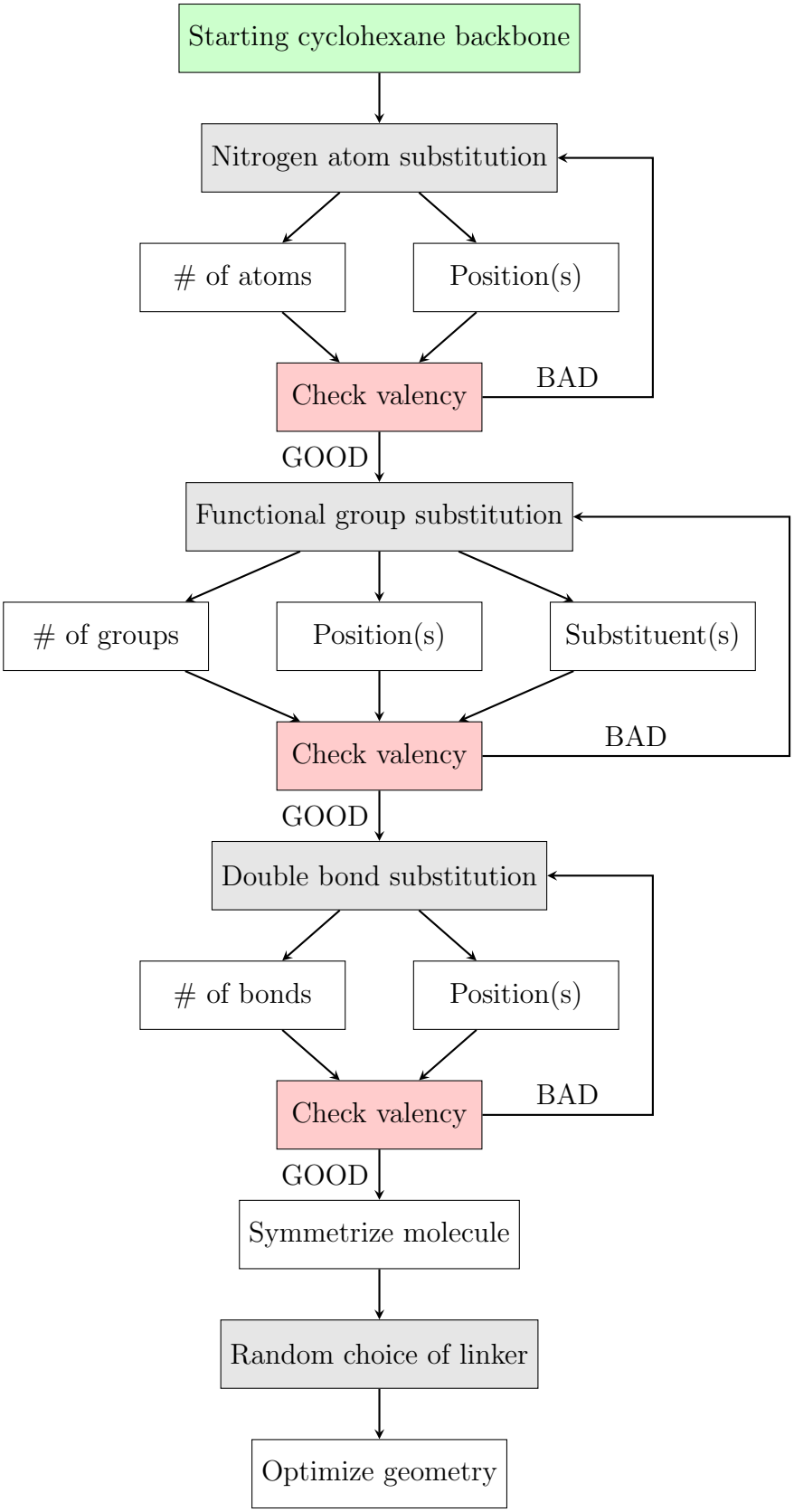


Figure 4.8: Flow of decisions made during the random initial generation of organic molecules.

Throughout the random generation of these organic molecules, a label was created for each molecule which describes the position of nitrogen atoms (XX), the position of double bonds (YY), the position (V) and choice of substituent(s) (W), and the chosen linker (Z) based on their respective number labels in Figures 4.6 and 4.7.

$$\text{N-}XX\text{-DB-}YY\text{-FG–}V,W\text{-L}Z$$

This enables the identification of the structural features of the molecules without having to visualize the full structure itself. Furthermore, the array-like labelling allows for a structural focused application of the GA, as opposed to the more intricate `.mol` file format, or equivalent.

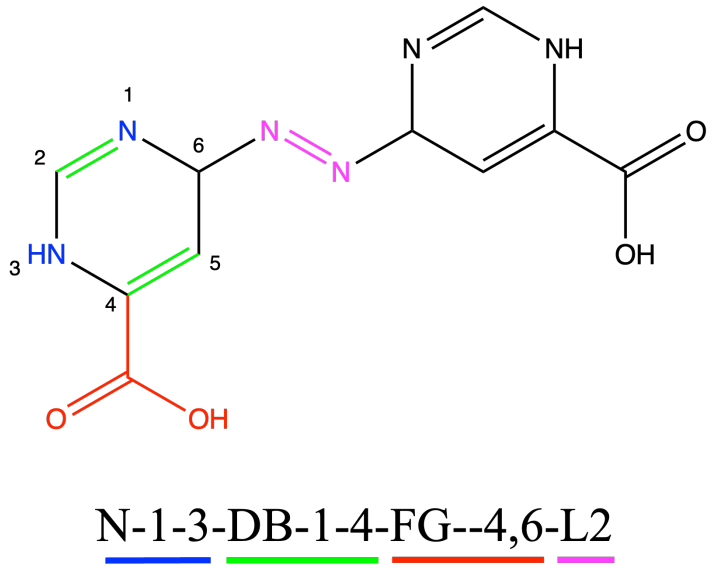


Figure 4.9: Example of label formatting for identifying the structural features of the generated molecules.

Figure 4.9 shows an example of how these molecular labels are constructed. First, it states the positions of the nitrogen atoms in the first generated ring. For this example molecule, the nitrogen atoms are in positions 1 and 3 as shown in blue. As mentioned, the label only specifies the construction of one ring since each molecule is

symmetric so the second ring will be a mirror image of the ring specified in the label. Next, the double bonds are specified by stating the atom that each double bond is connected to in a counterclockwise sense. For this molecule, there are double bonds connected to atoms 1 and 4, which implies that there is a double bond between atoms 1 and 2 as well as atoms 4 and 5 as shown in green. For the substituents, the label will specify the position of the substituent and the functional group number from the list of given functional groups, separated by a comma. For this molecule, there is a substituent at position 4 that is specified as functional group 6, corresponding to a carboxylic acid in Figure 4.6f. Lastly, the label will state the linker that connects the two rings and in this example that is linker 2 which corresponds to N=N in Figure 4.7b.

4.8 Applying Genetic Algorithms

Chapter 3.4 described the principles that drive GAs and the selection, crossover, and mutation methods that may be applied. In this project, rank selection was applied to select individuals to make up the GA population by computing their scoring function and only choosing the top scoring molecules to be included in the population. Stochastic universal sampling was then applied to choose the two parent molecules that will be crossed over to generate the offspring molecules. Uniform crossover (see Chapter 3.4.2) was applied to the chosen parent molecules by randomly swapping traits of these individuals. There was no specific mutation method applied in this GA. Instead, mutations were applied only if there were issues with the valency of the molecule so that the mutations had the purpose of fixing the valency issue. For example, if a choice was made during the crossover step to add a substituent to an atom in the ring that already has a full valency, a mutation is applied to change the position of substitution to an atom that does not have a full valency.

For this GA, a Python script was generated to first randomly select the two parent molecules from a set of top ranked molecules, then it applies the GA by using uniform crossover to generate offspring molecules from the two parent molecules. Lastly the script builds the newly generated molecules similarly to the methods described in Chapter 4.7 using the generated label for the offspring molecule. This Python script for the identification and selection of parent molecules followed by crossover can be seen in Appendix B, which was inserted into the script shown in Appendix A in order to build the offspring molecules generated from the crossover with some minor modifications to the script.

Rank selection was used to choose the GA population by applying a suitable scoring function (detailed in Chapter 4.10). For as many new molecules that are added to the new population from the GA, the equivalent number of lowest ranked molecules are removed to keep the population size the same throughout each GA generation. This allows for the higher ranked molecules to continue to be chosen for crossover and potentially pass on their more ideal traits while molecules that are poor candidates are removed from the population. Two parent molecules are randomly chosen from the population using stochastic universal sampling. In order to identify the properties of each parental structure, the unique labels for the molecules were used. The label can be broken down into its identifying parts that describe the number and position of nitrogen atoms and double bonds throughout the rings, the position and choice of substituent(s), and the linker chosen to connect the two rings. Once the labels are read and the structural components of the parent molecules are identified, uniform crossover is applied label-wise to generate the offspring molecules. An example of this is shown in Figure 4.10 where the offspring molecule received its ring backbone nitrogen atoms and linker from Parent 1 and its substituent and ring double bonds from Parent 2 during the crossover event.

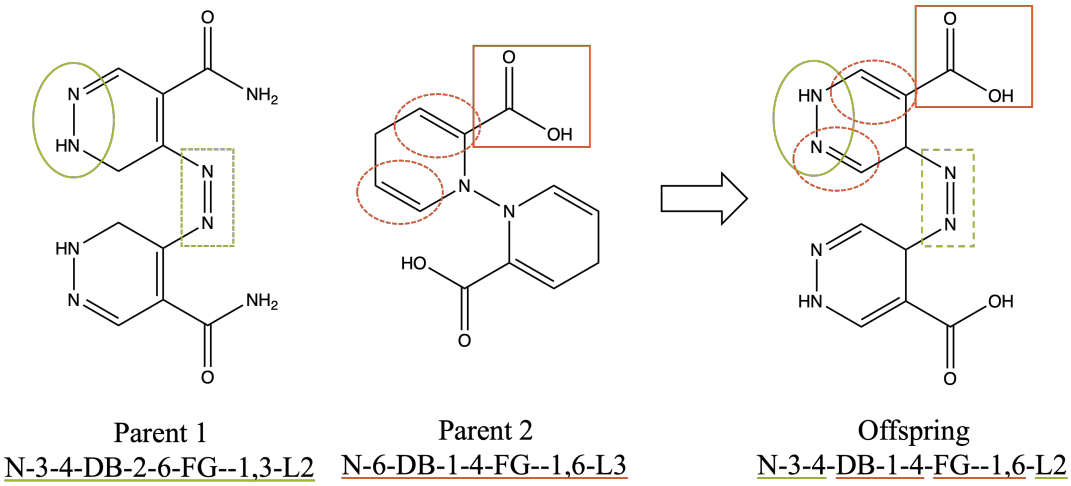
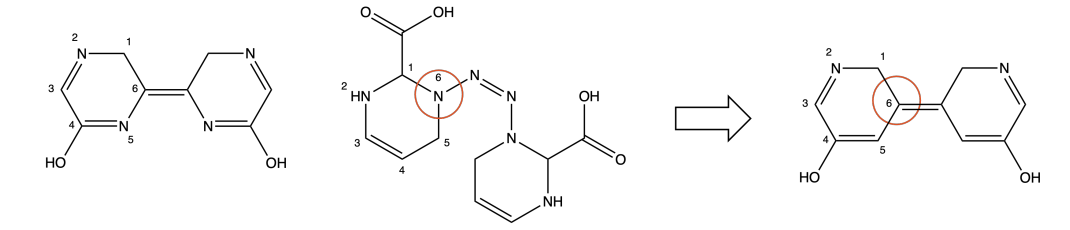


Figure 4.10: Example of crossover between two randomly chosen parent molecules to generate an offspring molecule.

With uniform crossover, label components along both parental gene sequences are randomly selected and swapped, generating unique offspring molecules. The script then reads the individual components of the new molecule from its label to generate the `.mol` files for these structures and the input files to perform the necessary computational calculations. Since uniform crossover doesn't take into account whether the crossover step will generate a chemically valid molecule, mutations are needed to ensure that viable structures are being generated. There are a few cases where mutations may be applied. An example would be if the chosen linker for connecting the rings of the offspring molecule is a double bond (Figure 4.11a). In this case, a nitrogen atom would not be able to be positioned in the ring where the double bond linker is attached so it would need to be substituted for a carbon atom instead as shown in the offspring molecule of Figure 4.11a. Here, the double bond linker from Parent 1 is chosen and the nitrogen atoms from Parent 2, however since the linker is a double bond, the nitrogen atom at position 6 has been replaced by a carbon atom instead to allow for the proper valency. A functional group would also not be able to be substituted at this position either since a carbon atom would already have a full valency. If that is the case, a mutation would be applied to choose

another position for that substituent instead. This also remains true for any other atom in the ring that already has a full valency. The other case where a mutation may need to be applied is in the case that one or both of the two atoms for which a double bond has been selected already have a full valency; this double bond would simply be omitted from the molecule entirely instead of finding another position for it in the ring. In the example shown in Figure 4.11b, since the double bonds for the rings of the offspring molecule are from Parent 1 and the substituent is from Parent 2, the valency at position 3 is already saturated, therefore no double bond is possible between atoms 2 and 3, and a single bond is chosen for the offspring molecule between these two atoms.

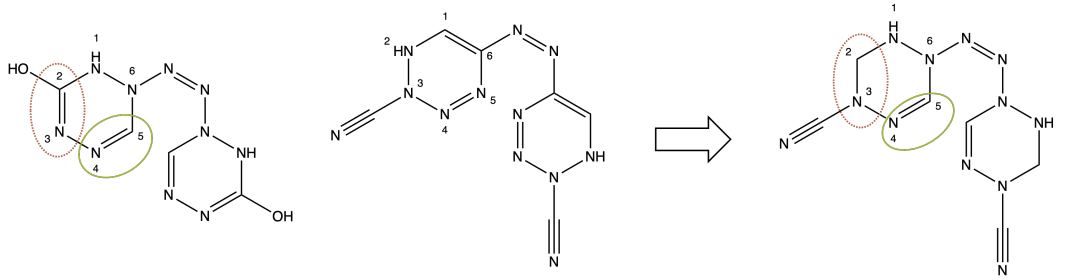


Parent 1
N-2-5-DB-2-4-FG--4,11-L100

Parent 2
N-2-6-DB-3-FG--1,6-L2

Offspring
N-2-DB-2-4-FG--4,11-L100

(a)



Parent 1
N-1-3-4-6-DB-2-4-FG-2,11-L2

Parent 2
N-2-3-4-5-DB-4-6-FG--3,4-L2

Offspring
N-1-3-4-6-DB-4-FG--3,4-L2

(b)

Figure 4.11: Examples of two possible mutation operations.

Initially, the number of molecules, or offspring, generated in each generation was 25, however, this was reduced to five molecules per generation as an attempt to speed

up the GA. While fewer molecules being generated per generation can lead to fewer new molecules being added to the population for the next generation, it can also allow for new populations to be generated faster as the population is being updated with new molecules more frequently than if more offspring molecules are being generated each generation. Once the molecules were generated for the GA generation, DFT calculations were run to determine their cell potential, stability, and solvation free energy and the scoring function (Chapter 4.10) for each molecule was then calculated from these values. Once these values were determined, the molecules with values that fall within all of the desired windows are ranked among the molecules of the previous GA generation. For each new molecule that is added to the GA population, the equivalent number of lowest ranking molecules are removed from the population, keeping the overall population number the same. This population is then used for the next GA generation and this process will be repeated as many times as desired.

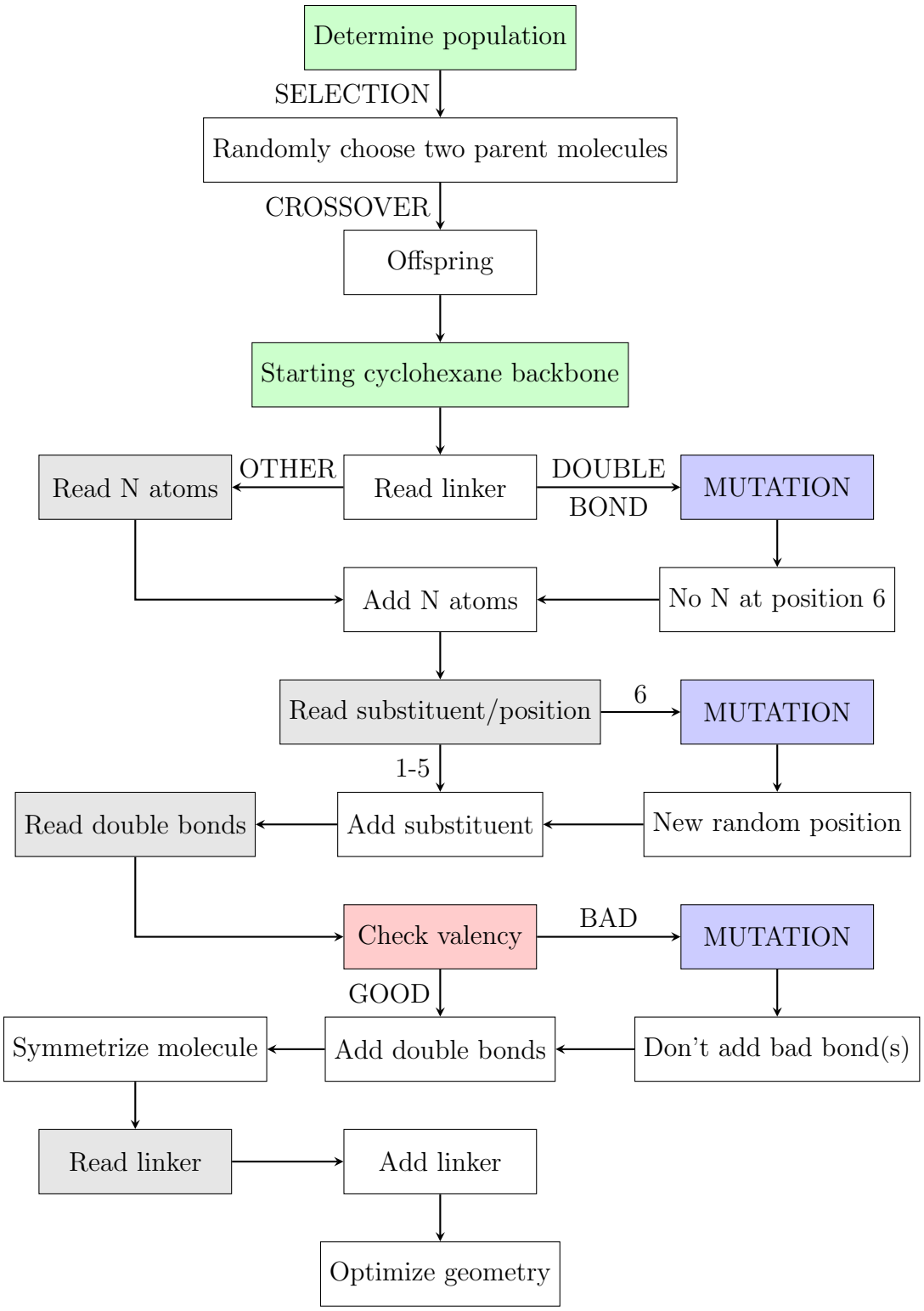


Figure 4.12: Flow of decisions made during the GA process to generate the structure of the offspring molecule.

Figure 4.12 shows the full flow of decisions made throughout the GA process for

each generation. Each new molecule generated by crossover events will go through these steps to determine if they are structurally valid molecules and mutations are performed as needed.

4.9 HOMO-LUMO Energy Gaps

For each generated molecule, the energy gap between the HOMO and LUMO were extracted as they have been reported to have a positive correlation with cell potential.[70]

$$E_{HOMO-LUMO\ gap} = |E_{LUMO} - E_{HOMO}| \tag{4.18}$$

The HOMO-LUMO energy gap was calculated for each molecule to be compared with the cell potential to cross check the positive correlation between these two variables.[70] This required no computational additions to the workflow as HOMO-LUMO gaps are byproducts of a DFT computation.

4.10 Scoring Function

In order to assess the viability of a compound as a redox-active material, all factors of E_{cell}, $\Delta G_{stability}$, and $\Delta G_{solvation}$ must be optimized concertedly. This can be achieved by means of a scoring function which allows molecules to be ranked by their overall redox capabilities. These functions are typically established for the unique scenario of the given problem and designed to be either minimized or maximized. For this project, a filter function was first applied to cut out any molecules that are immediately known not to be good candidates based on their given values for reduction potential, cell potential, solvation free energy, and stability. These molecules would be any that fall outside of the given solvent potential window, are not soluble,

or are not stable. Once the molecules that fell outside the given windows have been removed from the data set as potential candidates for RFB active materials, a scoring function is used to rank the remaining molecules by their success as potential candidates. This is done by evaluating the cell potential, stability, and solvation free energy of each molecule as these are the three factors that will determine the success of each molecule as redox-active materials and a score (S) is assigned to each molecule based on the combined influence of these values.

$$S = \alpha_{E_{cell}} S_{E_{cell}} + \alpha_{\Delta G_{stab.}} S_{\Delta G_{stab.}} + \alpha_{\Delta G_{solv.}} S_{\Delta G_{solv.}} \tag{4.19}$$

In equation 4.19, S is the score value, and $\alpha_{E_{cell}}$, $\alpha_{\Delta G_{stability}}$, and $\alpha_{\Delta G_{solvation}}$ are weights that must be determined by strategic choice. It was decided to weight the cell potential, stability, and solvation free energy equally, therefore the α values are chosen to be $\frac{1}{3}$. There are two ways to determine score, which were both explored.

1. Compare with a reference value

2. Compare with a data sample

In this project, the first method was applied using the BPY derivative in Figure 4.13 as a reference. This scoring function is a *chemistry*-driven method because comparisons are being made to the reference BPY molecule.

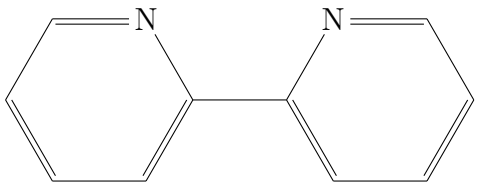


Figure 4.13: BPY molecule used to determine reference values for the scoring function.

$$
\begin{aligned}
S_{BPYref.} &= \frac{1}{3} \cdot \frac{E_{cell}}{E_{cell,BPY}} + \frac{1}{3} \cdot \frac{\Delta G_{stability}}{\Delta G_{stability,BPY}} + \frac{1}{3} \cdot \frac{\Delta G_{solvation}}{\Delta G_{solvation,BPY}} \\
&= \frac{1}{3} \cdot \frac{E_{cell}}{6.205 \text{ V}} + \frac{1}{3} \cdot \frac{\Delta G_{stability}}{864.277 \text{ kJ/mol}} + \frac{1}{3} \cdot \frac{\Delta G_{solvation}}{-36.344 \text{ kJ/mol}}
\end{aligned} \tag{4.20}
$$

BPY is a good base reference for determining the success of these molecules as redox-active materials as it has been experimentally characterized and is the simplest molecule in the chemical space. This means that any molecules with a score higher than 1.000 are likely more successful candidates than unsubstituted BPY since they have higher values in at least one of cell potential, stability, and solvation free energy.

While the above described method of determining a scoring function is a plausible method, it was found that it was strongly biased towards $\Delta G_{solvation}$. The value of solvation free energy for the reference BPY molecule was much less negative than many of the randomly generated molecules, leaving solvation free energy to have a much higher influence on the score than the other properties, biasing the ranking. Therefore, a second scoring function was determined to help keep the influence of each of the three properties on the overall score of each molecule more equal as well as to prevent potential bias from BPY. Instead of comparing the generated molecules with a reference BPY molecule, they were instead compared to the average values of the randomly generated population of molecules that fell within the windows for reduction and cell potential, stability, and solvation free energy, making this a *data*-driven method.

$$
S_{avg.ref.} = \frac{1}{3} \left[\left(\frac{\langle \Delta G \rangle - \Delta G}{\sigma(\Delta G)} \right)_{solv.} + \left(\frac{E - \langle E \rangle}{\sigma(E)} \right)_{cell} + \left(\frac{\Delta G - \langle \Delta G \rangle}{\sigma(\Delta G)} \right)_{stab.} \right] \tag{4.21}
$$

For this scoring function, the average values for solvation free energy ($\langle \Delta G \rangle_{solv.}$), cell potential ($\langle E \rangle_{cell}$), and stability ($\langle \Delta G \rangle_{stab.}$) as well as their standard deviation (σ)

were calculated for the initial random population of molecules that fell within the desired windows and then substituted into Eq. 4.21. The average and standard deviation values of the original randomly generated population were kept constant throughout the calculation of the scoring function for each generation of the GA as they provided a prior to the correct mean and standard distribution. This new scoring function allowed for the generated molecules to instead be compared to the original population of molecules within the desired windows to see if molecules generated throughout the GA improved from the original population or not.

As mentioned, the population for each generation was chosen using rank selection that made use of the scoring function of each molecule. Two scoring functions were explored to generate two different initial populations. The first scoring function compares the generated molecules to a reference BPY molecule where a score greater than 1.000 would indicate that the molecule should be a better candidate for use in non-aqueous RFBs that the reference BPY molecule itself. For the second scoring function, which compared the generated molecules to the average value of a randomly generated population, molecules with a score greater than 0.000 are performing better than the average. Following each generation, the newly generated molecules are ranked using their respective scoring function once their cell potential, stability, and solvation free energy have been computed. Molecules that are potential candidates are then added to the next population and those with the lowest scores are removed from the population, keeping the total population size the same throughout each generation.

Chapter 5

Results and Discussion

5.1 Introduction

The project involved two phases; first, the generation of a random set of organic molecules as an initial population for a GA and second, running the GA itself. There was also what could be considered a "zeroth" phase of the project, which focused on making assessments of BPY itself and some of its derivatives as means to test the computational methods of this project before using these methods on the generated molecules of this project. Without going into further discussion, the conclusions of this zeroth phase was that all tests were successful. The first sections will discuss the results of the original set of randomly generated molecules followed by those generated through applying a GA.

5.2 Characteristics of Randomly Generated Molecules

While more molecules were originally generated, the data set that was produced from the initial random generation contained 397 randomly generated molecules.

Molecules not included in this total are those for which the DFT energy computations in GAMESS[75] did not converge and were therefore removed from the data set under the assumption they would be unstable and/or poor candidates for use in RFBs. The molecules that were able to be fully computed and therefore make up this data set showed a variety of structural characteristics. This set of randomly generated molecules can be separated by the number of substitutents that are present on each ring. The maximum number of substituents allowed per ring was two since it was discovered that having any more substituents would not allow for the geometry of the molecule to be optimized due to steric hindrance. In some cases, there were molecules generated that had no substituents. These molecules were removed from the data set since a large part of this project involves looking at how much of an influence substituents have on the overall structure and redox properties of the generated molecules. While 397 molecules were generated in total with either one or two substituents per ring, only the 360 molecules that had one substituent per ring were further characterized and retained in the data set. This is due to complications coding the stability measure of the molecules when there is more than one substituent per ring which led to removing the 37 molecules with two substituents per ring from the data set. One of the issues with determining the stability of these molecules was identifying individual substituents when there are two different substituents on a ring. The data set was further reduced using a filter function to the 156 molecules that fell within the windows for reduction and cell potential, stability, and solvation free energy. From this point forward, the initial population being referenced constitutes the 360 molecules with one substituent per ring, unless specified otherwise.

5.2.1 Nitrogen Content

Figure 5.1 shows the distribution of the 360 molecules in the randomly generated data set by the number of nitrogen atoms that are present in each ring of the

structure.

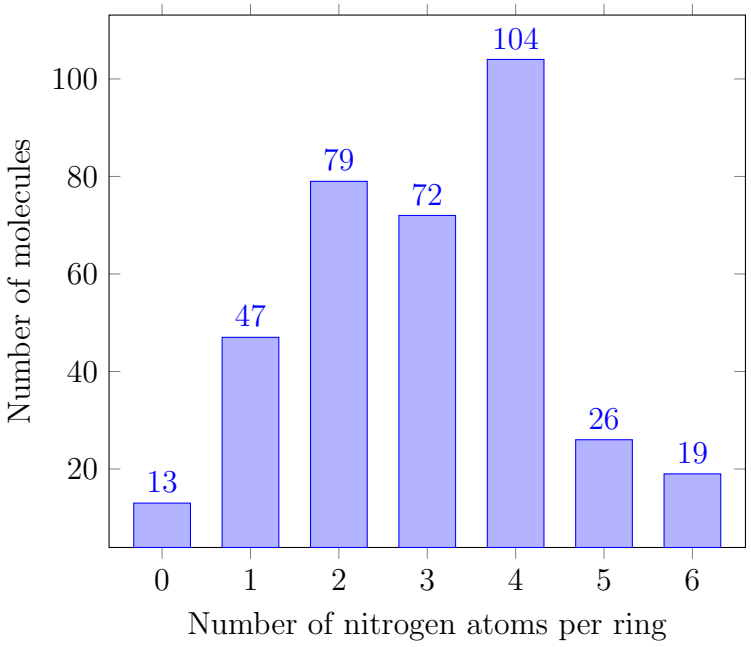


Figure 5.1: Distribution of randomly generated structures by the number of nitrogen atoms present in each ring for molecules with one substituent per ring. Population size = 360 molecules.

For the generation of these molecules, a weight factor was applied to molecules with three nitrogen atoms or less per ring to allow for higher chances of 1-3 nitrogen atoms to be placed in each ring. While four nitrogen atoms per ring was chosen more times than any other number of nitrogen atoms with 104 molecules total, altogether one, two, and three nitrogen atoms per ring make up around 55% of the overall distribution for these molecules. This leaves four, five, and six nitrogen atoms per ring making up 41% of the distribution and molecules with only carbon atoms and no nitrogen atoms in their rings, making up 4% of the total distribution.

5.2.2 Substituents

Figure 5.2 shows the distribution of the substituents on the first ring constructed where position 6 is the position of the linker and the second ring is the mirror image of the first ring since the entire molecule is generated symmetrically.

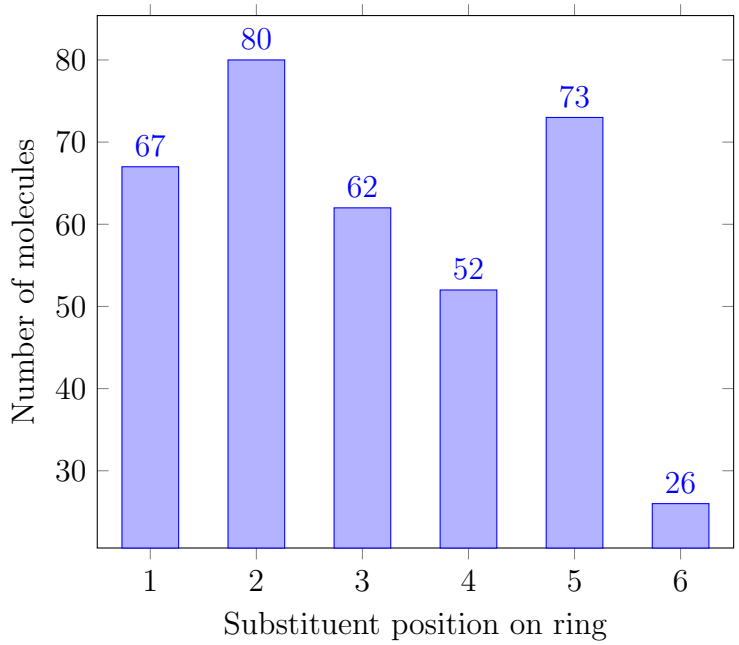


Figure 5.2: Distribution of randomly generated molecules by the position of the substituent for molecules with one substituent per ring. Population size = 360 molecules.

Valency is a factor that influences which positions functional groups may be substituted. Position 6 in particular is influenced by the valency because if the atom at this position is a nitrogen atom or the chosen linker is a double bond, a substituent cannot be added at this position since the valency would not allow for it. Because of this, the fewest substitutions were made at position 6, making up 7% of the total distribution due to the limited valency of this position, depending on the chosen linker. For the rest of the positions, a non-uniform distribution is observed.

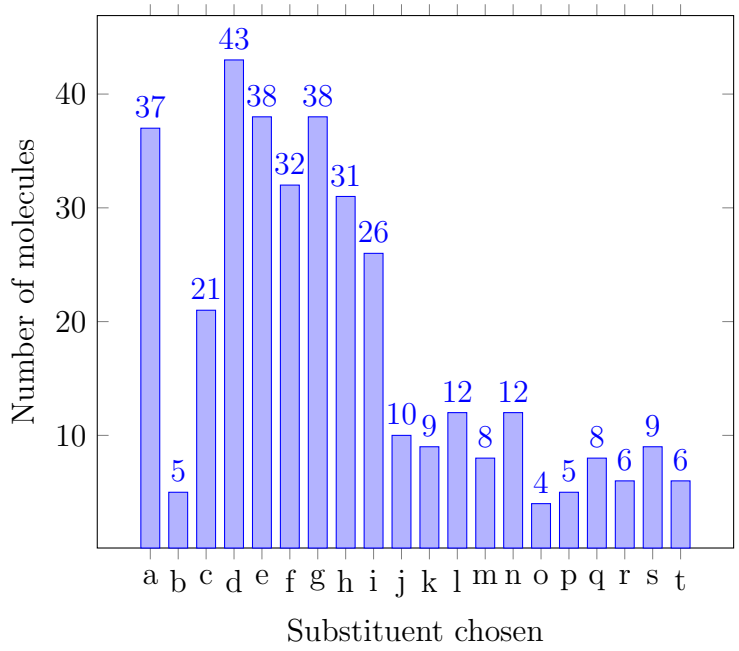


Figure 5.3: Distribution of randomly generated molecules by the number of times each substituent was chosen for molecules with one substituent per ring. Population size = 360 molecules.

The 360 randomly generated molecules have been organized in Figure 5.3 by the number of molecules that contains each of the given functional groups as their substituent. The subtituents (a) through (t) correspond to those seen in Figure 4.6. Initially, the first nine substituents (a) through (i) were the only functional groups used as potential substituents before the remaining 11 groups, (j) through (t), were added to the set of potential substituents. This explains why most of the distribution of substituents seen in Figure 5.3 is made up of these nine substituents, and is not uniform.

5.2.3 Double Bonds

The generated molecules were also characterized by the number of double bonds present throughout each ring as shown in Figure 5.4.

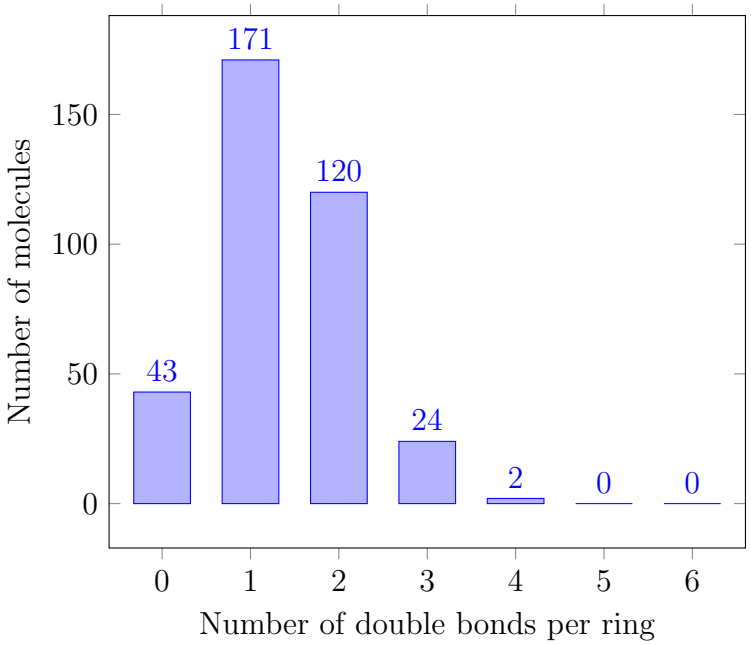


Figure 5.4: Distribution of randomly generated molecules by the number of double bonds present in each ring for molecules with one substituent per ring. Population size = 360 molecules.

At first, up to six double bonds were allowed throughout each ring, however, due to valency and the order in which the molecules were constructed, molecules with five or six double bonds per ring were not seen since the valency of the molecule did not allow for that many double bonds in the rings after the nitrogen atoms and substituents are already placed throughout the rings. While molecules with four double bonds per ring were allowed initially, only two molecules that were randomly generated with four double bonds per ring converged during the energy calculations therefore these molecules only make up 0.5% of the distribution. The number of double bonds per ring was eventually limited to a maximum of three since any more double bonds would always lead to *sp*-hybridized carbon atoms present in the rings. While this did not fully remove the possibility of *sp*-hybridized carbon atoms appearing, it did help to prevent them. More adjustments to the script for randomly generating these molecules would be needed to fully remove the possibility of *sp*-hybridized carbon atoms, however, from a GA point of view, it may be interesting

to retain them in the set to diversify the population. Most structures generated had only one double bond per ring, making up 48% of the total distribution. Next are molecules with two double bonds making up 33% of the distribution and those with three double bonds per ring making up almost 7% of the total molecules. This percentage is the lowest due to some of these molecules having two of the three double bonds connected to the same atom, leading to an *sp*-hybridized carbon atom. As for molecules with no double bonds throughout their rings, they make up 12% of the total distribution and in some cases are most likely due to these molecules already having full valency after the substitution of nitrogen atoms and substituents throughout the rings therefore leaving no atoms with free valency for double bonds to be susbtituted.

5.2.4 Linkers

Figure 5.5 shows the distribution of linkers chosen for each molecule for the data set of 360 molecules where linkers (a) through (e) correspond to those seen in Figure 4.7.

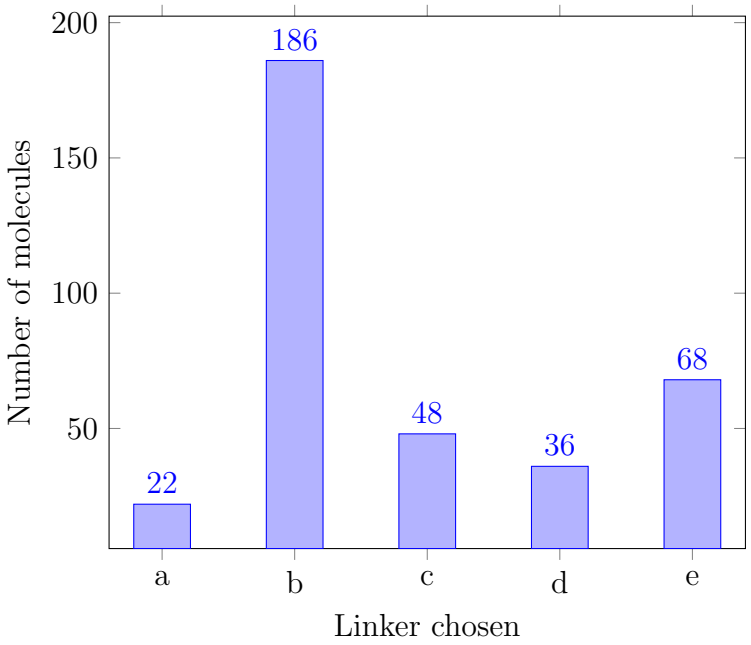


Figure 5.5: Distribution of randomly generated molecules by linkers chosen to connect the two rings for molecules with one substituent per ring. Population size = 360 molecules.

Linker (b) was chosen as a linker most often with 52% of the total distribution. Linkers (c) and (d) made up 13% and 10% of the distribution, respectively, while linker (a) only made up around 6% of the total distribution. For the generation of some of these molecules, not all of the linkers were available to be chosen due to valency requirements at position 6 which shows the large discrepancy of linker (b) being chosen over the other linkers.

Overall, the initial data set of randomly generated molecules varies greatly in their structures. By minimizing the constraints on these molecules only to valency requirements, a wider variety of structures was able to be obtained to keep the search area of these randomly generated organic molecules as large as possible.

5.3 Data Correlation Scatter Plots

Values of $\Delta G_{solvation}$, $E_{2+/0}$, $E_{0/2-}$, E_{cell}, $E_{HOMO-LUMO}$, and $\Delta G_{stability}$ have been computed for the initially generated set of 360 molecules and expected correlations between some of the values have been verified using the scatter plot matrix shown in Figure 5.6. In the plotted matrix, there are comparisons made between the computed values of solvation free energy, reduction potentials, cell potential, stability of the neutral molecule, and the HOMO-LUMO energy gap ($E_{HOMO-LUMO}$). These scatter plots may then be used to examine any correlation between the different properties in order to eliminate highly correlated observables from the scoring functions.

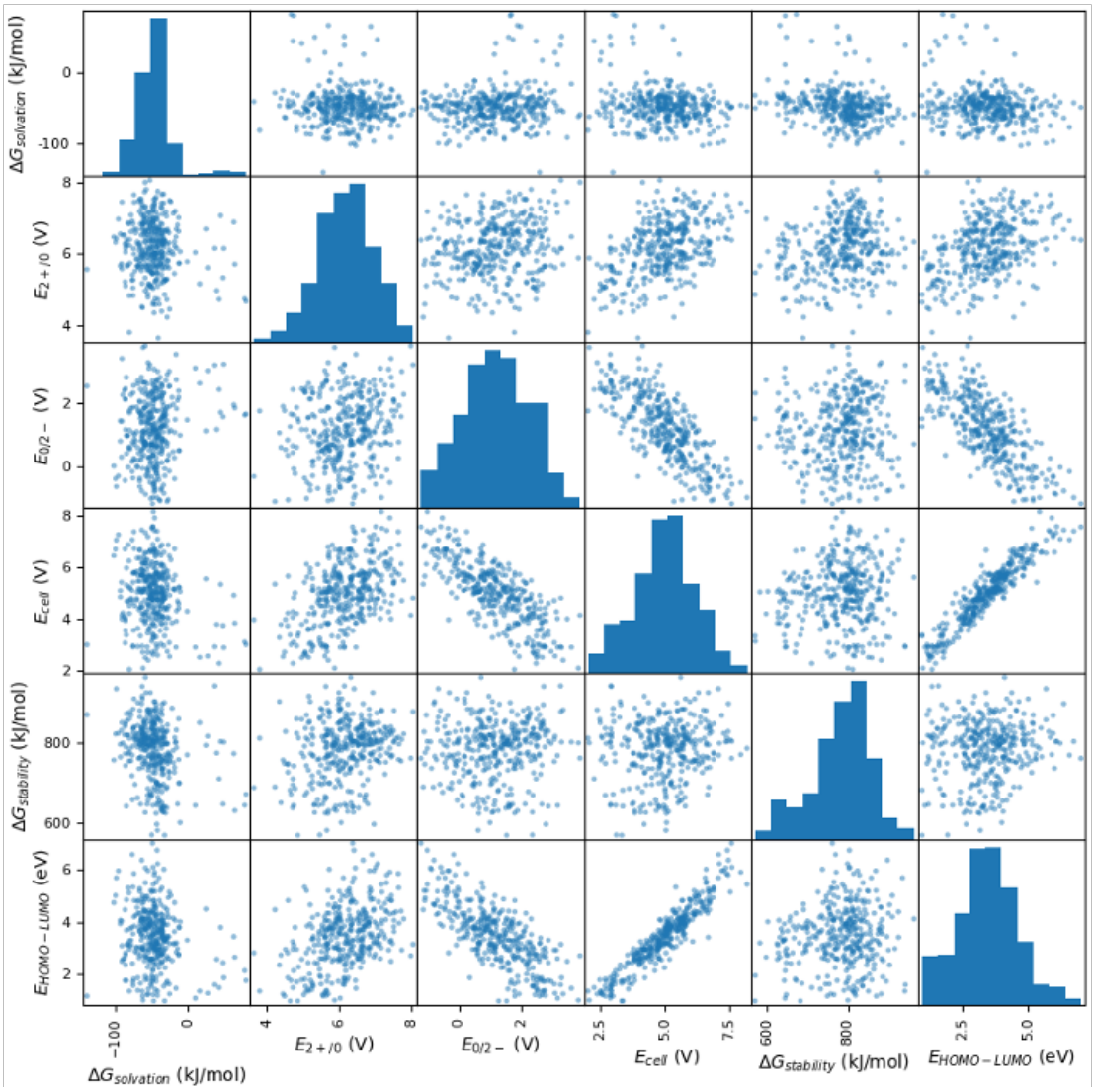


Figure 5.6: Scatter plot matrix of different properties from the original generated data set of molecules. Population size = 360 molecules.

The first correlation observed is the positive correlation between $E_{HOMO-LUMO}$ and E_{cell} which confirms the previous observations.[70] In line with that result, there also exists a weak positive correlation between $E_{2+/0}$ and $E_{HOMO-LUMO}$ and a stronger negative correlation between $E_{0/2-}$ and $E_{HOMO-LUMO}$. The last observed correlations are between the reduction and cell potential values. $E_{2+/0}$ shows a positive correlation with the E_{cell} while $E_{0/2-}$ shows a negative correlation with E_{cell}. There appears to be no correlation between $\Delta G_{solvation}$ and any of the other proper-

ties. The same observation is made with $\Delta G_{stability}$. This scatter plot matrix allowed for some insight as to which qualities to include in the scoring functions, or rather which not to include. $E_{HOMO-LUMO}$ was not included due to its strong positive correlation with E_{cell} therefore only one needs to be accounted for in the scoring functions. The other two properties, $E_{2+/0}$ and $E_{0/2-}$ were not included since these values are used to calculate E_{cell} which is part of the scoring functions.

5.4 Reduction and Cell Potentials

5.4.1 DMF Solvent Window

Computations were performed on each randomly generated molecule in the neutral, dication, and dianion states in DMF to determine their Gibb's free energy. These values were then used in the Nernst equation (Eq. 4.1) to determine their respective reduction potentials followed by the calculation of their overall cell potential from the reduction potentials. In order for the generated molecules to be possible redox-active candidates, they must fall inside the calculated window for the given solvent. For DMF, this window was calculated to be between -0.797 to 6.335 V for the reduction potentials and not above 7.132 V for the cell potential. The overall cell potential range of the calculated values for the 360 initial molecules was 2.038 to 8.160 V, however, since any value above 7.132 V falls outside of the window for DMF, this range changes to 2.038 to 6.152 V for the molecules within the given windows. Most of the molecules whose cell potentials are too high also have reduction potentials that fall outside of their given windows as well. Of the 360 total molecules, 207 molecules had values below the upper solvent window limit of 6.335 V for $E_{2+/0}$ and 304 molecules above the lower limit of 0.000 V for $E_{0/2-}$.

5.4.2 E_{cell} Ranking

Reported in Table 5.1 and Figure 5.7 are the molecules with the five highest values of cell potential from the randomly generated population that fell inside the windows for E_{cell}, $\Delta G_{stability}$, and $\Delta G_{solvation}$.

Table 5.1: Top five molecules with the highest cell potential values.

Rank	Label	E_{cell} (V)	$\Delta G_{stability}$ (kJ/mol)	$\Delta G_{solvation}$ (kJ/mol)
1	600-N-2-5-6-DB-2-FG–4,2-L3	6.152	734.601	-43.984
2	555-N-1-2-3-DB-4-FG–5,7-L5	5.850	810.827	-69.931
3	396-N-2-3-5-6-DB-3-FG–2,8-L2	5.795	910.897	-74.213
4	393-N-1-2-6-DB-FG–1,20-L2	5.785	641.922	-35.663
5	326-N-1-3-4-5-DB-1-FG–6,15-L2	5.718	835.454	-63.244

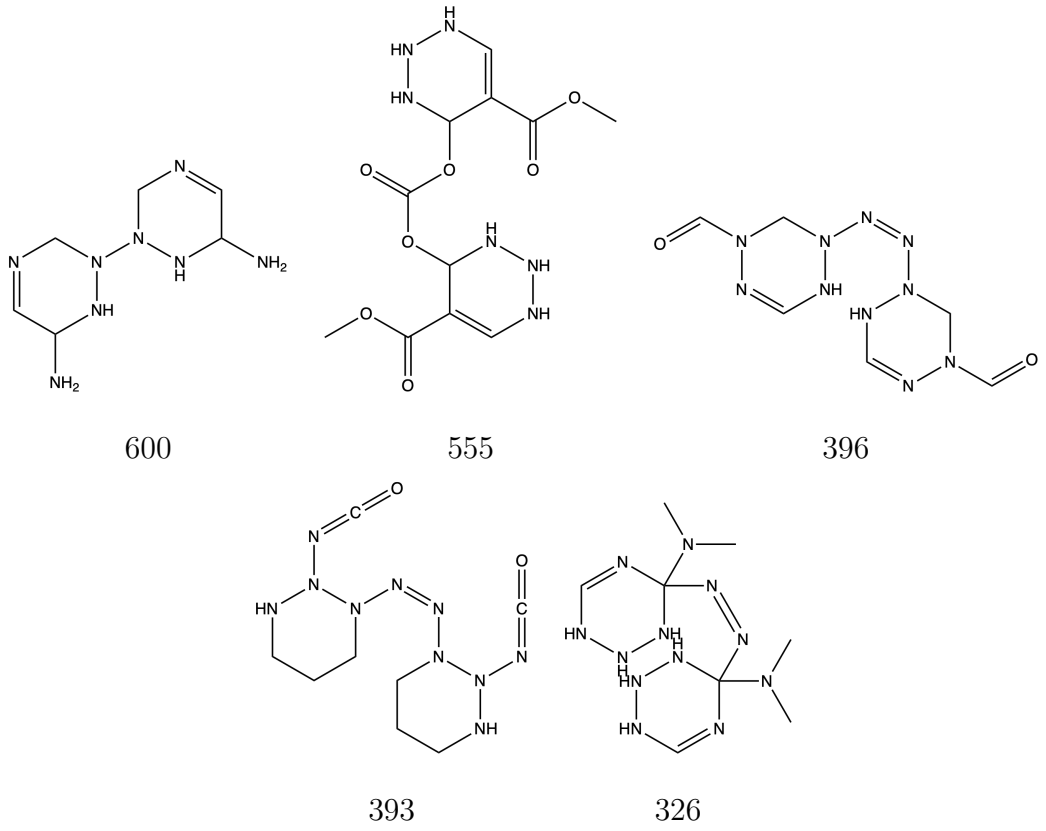


Figure 5.7: Structures of top five molecules with the highest cell potential values.

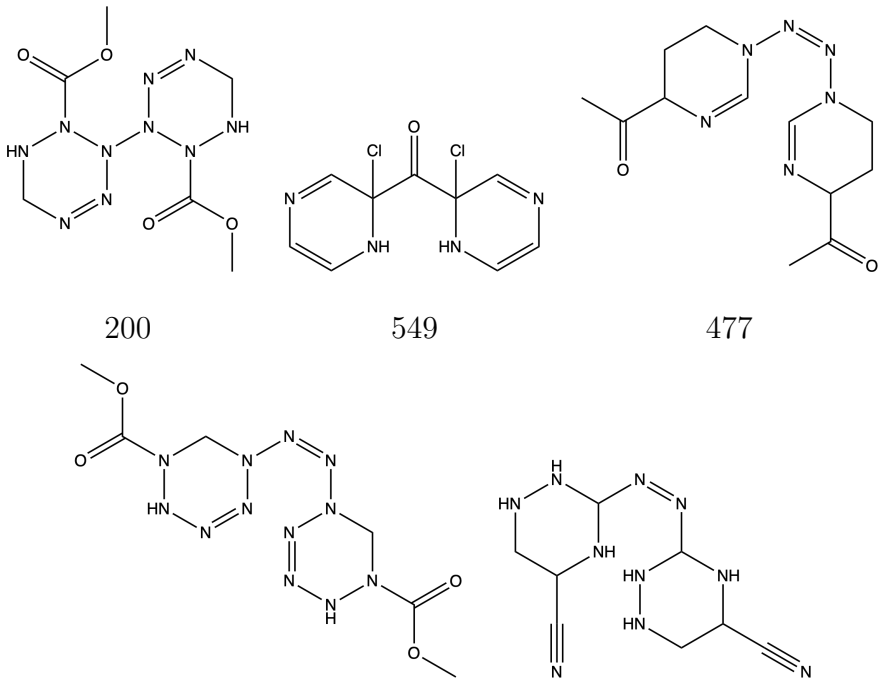
Most of these structures exhibit good values for E_{cell}, $\Delta G_{stability}$, and $\Delta G_{solvation}$, however, molecule 393 shows a lower than expected value for stability and less negative than expected value for solvation free energy. Because of this, molecule 393 actually fails in both scoring functions, despite having one of the highest values of cell potential. There are some structural similarities between the structures shown in Figure 5.7. These structures all have one double bond in each ring except for molecule 393 which actually has no double bonds present in its rings. These molecules also all have 3-4 nitrogen atoms substituted throughout each ring structure. Each of these top scoring molecules have relatively high values for $E_{2+/0}$ and relatively low values of $E_{0/2-}$, which is what allows for their respective cell potentials to remain high as well. The values of $E_{2+/0}$ for these top five molecules range from 5.855 to 6.294 V and the values of $E_{0/2-}$ range from 0.006 to 0.304 V.

5.4.3 Effect of Substituents on Reduction and Cell Potential

High cell potentials can be mostly achieved when a molecule has a large $E_{2+/0}$ value and small $E_{0/2-}$ while remaining within the solvent potential window. Some factors that affect the reduction potentials and in turn the cell potential are the substituents themselves. The functional groups used as substituents can be classified as either EWGs or EDGs, which act to either increase or decrease the reduction potentials as described in Chapter 4.7. Substituents that are EWGs are expected to increase reduction potential values due to the stabilizing effects of the groups while EDGs are expected to decrease reduction potential values by preventing the entry of electrons to the ring.[82]

Table 5.2: Top five and bottom five molecules from highest to lowest $E_{2+/0}$ values, respectively.

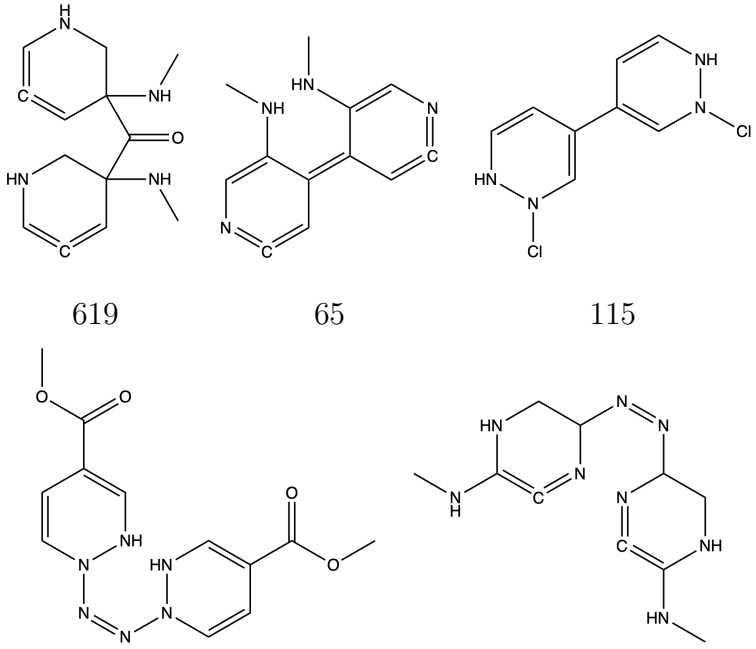
Rank	Label	Group	$E_{2+/0}$ (V)	$E_{0/2-}$ (V)	E_{cell} (V)
1	200-N-1-2-4-5-6-DB-4-FG–1,7-L3	EWG	6.334	0.829	5.505
2	549-N-2-5-DB-1-3-FG–6,1-L1	EWG	6.328	1.800	4.528
3	477-N-4-6-DB-4-FG–3,13-L2	EWG	6.324	0.713	5.611
4	118-N-2-3-4-5-6-DB-4-FG–2,7-L2	EWG	6.322	0.833	5.489
5	597-N-1-2-5-DB-FG–4,4-L2	EWG	6.299	0.686	5.612
...	...	...	...	...	...
152	619-N-2-DB-3-4-FG–6,9-L1	EDG	4.492	1.779	2.713
153	65-N-3-DB-1-3-4-FG–1,9-L100	EDG	4.451	1.913	2.538
154	115-N-3-4-DB-1-5-FG–4,1-L3	EWG	4.393	0.606	3.787
155	418-N-5-6-DB-1-3-FG–3,7-L2	EWG	4.350	1.071	3.279
156	264-N-2-5-DB-3-4-FG–3,9-L2	EDG	3.805	1.767	2.038



200 549 477

118 597

(a) Top five molecules with the highest $E_{2+/0}$ values.



619 65 115

418 264

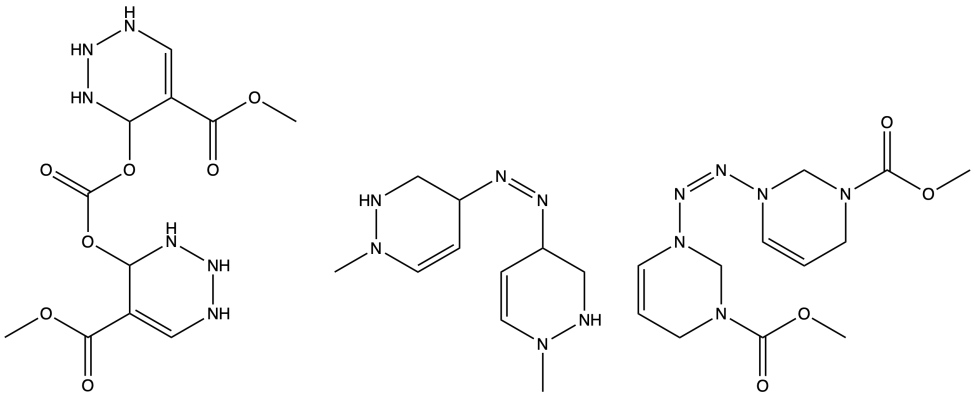
(b) Bottom five molecules with the lowest $E_{2+/0}$ values.

Figure 5.8: Structures of molecules with (a) highest and (b) lowest $E_{2+/0}$ values.

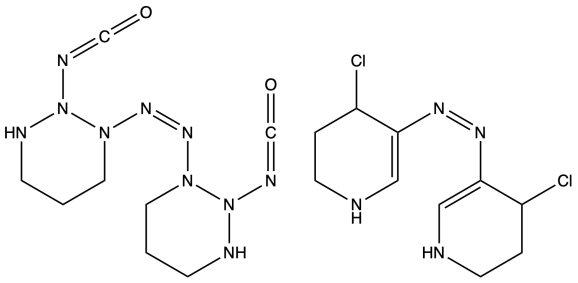
Table 5.2 and shows the top and bottom five molecules from highest to lowest $E_{2+/0}$ values to observe the effects of the substituents, and their respective structures are shown in Figure 5.8 where the population has been reduced to only the 156 molecules that fell inside the windows for E_{cell}, $\Delta G_{stability}$, and $\Delta G_{solvation}$. The substituents of the molecules with the five highest $E_{2+/0}$ values are all EWGs which is expected. As for the five lowest values, there are molecules that are substituted with EDGs that have some of the lowest values of $E_{2+/0}$ as expected since EDGs should decrease reduction potential. Molecules 115 and 418 are two of the molecules with the lowest $E_{2+/0}$ values, despite both being substituted with EWGs. Therefore there must be additional characteristics of these molecules that are affecting their reduction potential values. In general, these molecules have shown that EWGs will increase reduction potential values while EDGs will decrease reduction potential values.

Table 5.3: Top five and bottom five molecules from lowest to highest $E_{0/2-}$ values, respectively.

Rank	Label	Group	$E_{0/2-}$ (V)	$E_{2+/0}$ (V)	E_{cell} (V)
1	555-N-1-2-3-DB-4-FG–5,7-L5	EWG	0.006	5.855	5.850
2	275-N-3-4-DB-1-FG–3,10-L2	EDG	0.013	5.275	5.261
3	243-N-2-6-DB-4-FG–2,7-L2	EWG	0.092	5.805	5.714
4	393-N-1-2-6-DB-FG–1,20-L2	EWG	0.121	5.906	5.785
5	513-N-4-DB-5-FG–1,1-L2	EWG	0.139	5.613	5.474
...	...	...	...	...	...
152	496-N-1-2-3-5-DB-3-4-6-FG–2,11-L2	EDG	2.838	6.215	3.377
153	579-N-1-DB-3-4-6-FG–3,1-L3	EWG	2.927	5.665	2.738
154	499-N-1-2-3-4-DB-3-5-FG–2,13-L2	EWG	3.030	6.110	3.080
155	506-N-DB-1-3-6-FG–2,8-L1	EWG	3.307	5.700	2.393
156	546-N-1-DB-1-4-5-FG–2,5-L1	EWG	3.790	5.877	2.087

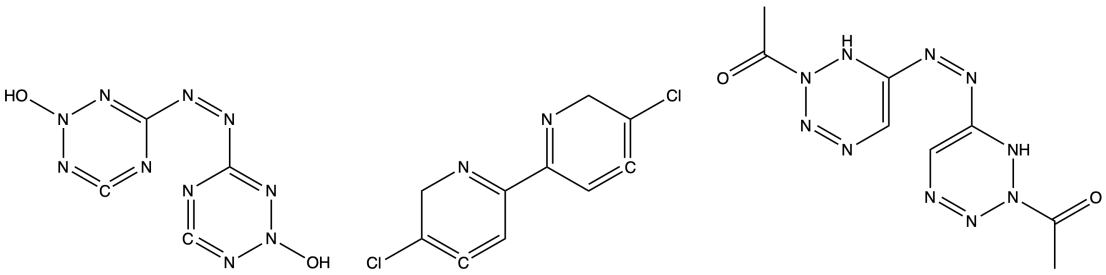


555 275 243

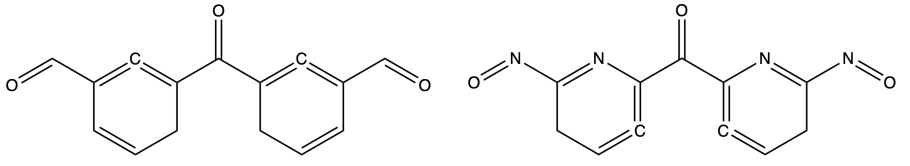


393 513

(a) Top five molecules with the lowest $E_{0/2-}$ values.



496 579 499



506 546

(b) Bottom five molecules with the highest $E_{0/2-}$ values.

Figure 5.9: Structures of molecules with (a) lowest and (b) highest $E_{0/2-}$ values.

Table 5.3 shows the top and bottom five molecules from lowest to highest $E_{0/2-}$ values to inspect the effects of the substituents, and their respective structures are shown in Figure 5.9 where the population has been reduced to only the 156 molecules that fell inside the windows for E_{cell}, $\Delta G_{stability}$, and $\Delta G_{solvation}$. In this table, it would be expected that molecules with the lowest $E_{0/2-}$ values would be substituted with EDGs. This is not the case, however, where four of the five molecules with the lowest $E_{0/2-}$ values are substituted with EWGs. Looking at the $E_{2+/0}$ values of these molecules, they are relatively high, showing that these EWGs have a stronger effect on $E_{2+/0}$ than $E_{0/2-}$. As for the molecules with the highest values of $E_{0/2-}$, most are substituted with EWGs and both have relatively high $E_{0/2-}$ and $E_{2+/0}$ values. Overall, these results appear to be fairly consistent with the strong effect that EWGs have over EDGs on reduction potentials. These are general comparisons, however, since the generated molecules do not have the same backbone structure to fully compare the effects of the substituents.

5.5 Thermodynamic Stability

As described in Chapter 4.5, determining the thermodynamic stability for these molecules was achieved by calculating the energy for the loss of a substituent from one ring of the molecule as a proxy. Thermodynamic stability is related to the stability of the activated complex, or intermediate state, of a chemical reaction,[83] which as seen in Figure 4.4, is $\mathrm{FGRR^- + FG^+ + H_2}$ after the loss of a substituent. Qualitative estimates of stability were made under the assumption that the magnitude of energy for the deprotonation of FGRRH + HFG after the loss of a substituent is roughly equal to that of the deprotonation of an unsubstituted BPY molecule. This allowed for an estimate of thermodynamic stability for each molecule to be compared to that of BPY. Most molecules were within the window of having a positive $\Delta G_{stability}$ value

where the range of values is from 568.970 to 964.513 kJ/mol. Only five molecules had negative values of $\Delta G_{stability}$ and were therefore deemed to be unstable molecules since the loss of a substituent would be favourable in these cases. Of the 360 total molecules, 355 molecules fell inside the window for stability.

5.5.1 $\Delta G_{stability}$ Ranking

Table 5.4 shows the top five molecules from the randomly generated population that fell inside the windows for E_{cell}, $\Delta G_{stability}$, and $\Delta G_{solvation}$ with the highest values for thermodynamic stability and their structures are shown in Figure 5.10.

Table 5.4: Top five molecules with the highest free energy of stability values.

Rank	Label	$\Delta G_{stability}$ (kJ/mol)	E_{cell} (V)	$\Delta G_{solvation}$ (kJ/mol)
1	477-N-4-6-DB-4-FG–3,13-L2	964.513	5.611	-58.414
2	603-N-DB-4-5-FG–1,7-L1	941.885	2.940	-62.936
3	263-N-3-4-6-DB-1-FG-2,13-L2	917.145	4.209	-62.936
4	396-N-2-3-5-6-DB-3-FG–2,8-L2	910.897	5.795	-74.213
5	418-N-5-6-DB-1-3-FG–3,7-L2	903.913	3.279	-46.429

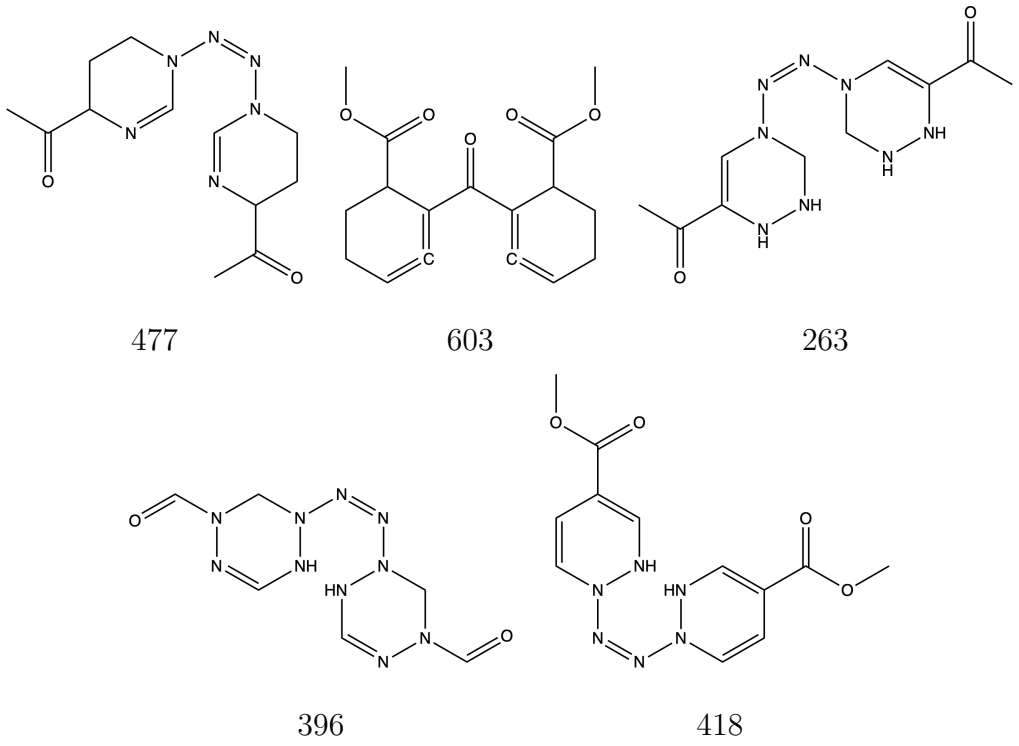


Figure 5.10: Structures of top five molecules with the highest free energy of stability values.

The range of values for these molecules is from 903.913 to 964.513 kJ/mol, which are higher values than that for the reference BPY molecule and the average stability of the randomly generated data set. Molecules 603 and 418 have the lowest cell potential values of these top-ranking molecules at 2.940 V and 3.279 V, respectively, where higher cell potential values would be more favourable. The structure of molecule 603 has no nitrogen atoms throughout its rings, however, it does have an *sp*-hybridized carbon atom that is not favourable with respect to ring stability. All of these top five molecules for stability have either an acetyl (molecules 477 and 263), ester (molecules 603 and 418), or aldehyde (molecule 396) group as their substituted functional group, leading to more stable molecules due to the stabilizing effects of these moderately deactivating EWGs.

5.6 Solvation Free Energy in the Neutral State

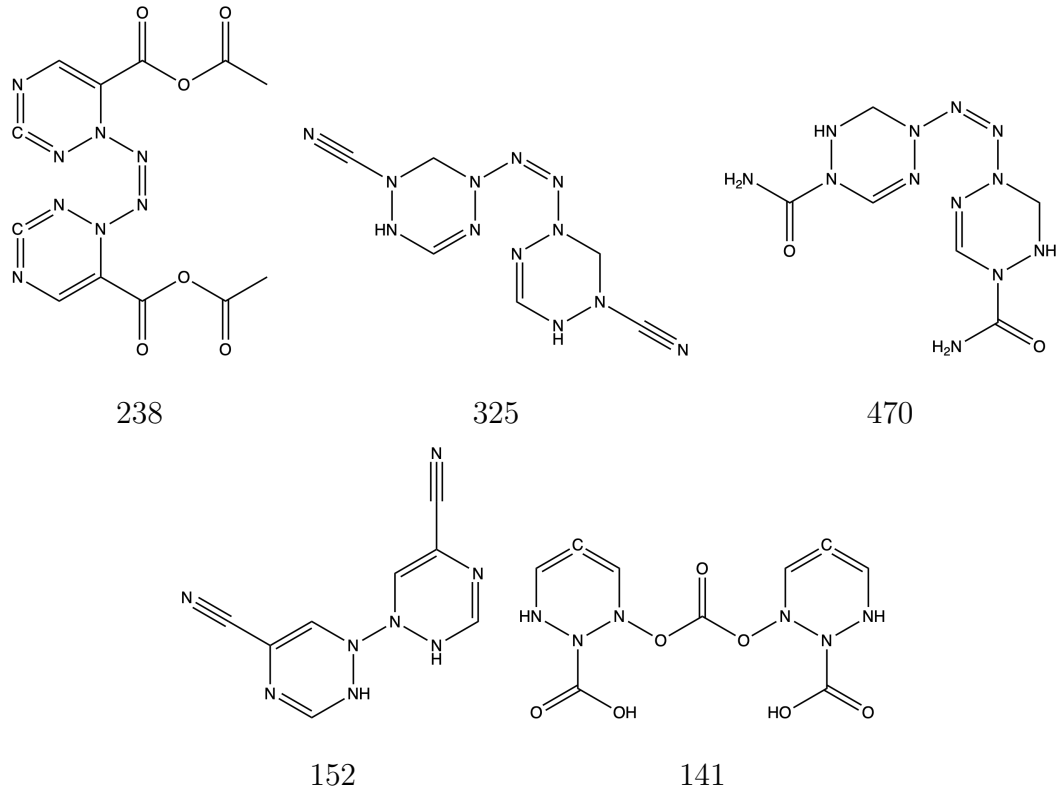
Computations were performed on the neutral state of each generated molecule in both the gas phase and DMF as the solvent. This allowed for the calculation of solvation free energy as described in Chapter 4.6 to gain an understanding of the solubility of these molecules. It has been shown that solvation free energy has a general positive correlation with the solubility of organic molecules[70] where a more negative solvation free energy indicates a more soluble molecule. For this study, any molecule that had a positive value was determined to be insoluble and therefore not a good candidate as a redox-active material for non-aqueous RFBs. In total, 346 of the total calculated molecules had a negative solvation free energy and were therefore deemed to be soluble in the neutral state, leaving only 14 molecules that fell outside the desired window for solvation free energy. The values for those within the window range from -139.359 to -0.462 kJ/mol where the more negative the solvation free energy, the more soluble the molecule should be. More calculations would be needed in order to determine if these molecules are also soluble in their respective dication and dianion states as well since only the solvation free energy in the neutral state was calculated for this project.

5.6.1 $\Delta G_{solvation}$ Ranking

The molecules with the five most negative solvation free energy values from the randomly generated population that fell inside the windows for E_{cell}, $\Delta G_{stability}$, and $\Delta G_{solvation}$ have been reported in Table 5.5 and their respective structures are shown in Figure 5.11.

Table 5.5: Top five molecules with the most negative solvation free energy values.

Rank	Label	$\Delta G_{solvation}$ (kJ/mol)	E_{cell} (V)	$\Delta G_{stability}$ (kJ/mol)
1	238-N-3-5-6-DB-1-3-4-FG–1,21-L2	-139.359	2.997	870.944
2	325-N-1-3-4-6-DB-1-FG–4,4-L2	-93.093	4.851	764.945
3	470-N-2-3-5-6-DB-4-FG–3,3-L2	-91.553	5.423	831.900
4	152-N-3-5-6-DB-1-3-FG–2,4-L3	-90.962	4.135	851.273
5	141-N-4-5-6-DB-1-2-FG–5,6-L5	-86.436	3.070	773.393



238 325 470

152 141

Figure 5.11: Structures of top five molecules with the most negative solvation free energy values.

All of these molecules have significantly more negative values of $\Delta G_{solvation}$ than both the BPY reference molecule and the average value of the data set, meaning they all should be more soluble given the positive correlation between solvation free

energy and solubility.[70] Looking at the structures of molecules 238 and 141, both have an *sp*-hybridized carbon atom present in their rings. Both of these molecules have relatively good values for their solvation free energy and stability, however, their cell potential values are much lower than desired. Not only would these molecules most likely be poor active material candidates due to their low cell potential despite their favourable solubility, they would be unstable molecules because of their *sp*-hybridized carbon atom. As for molecules 325, 470, and 152, their data values are all in the desired ranges for potential active material candidates. Molecules 325 and 152 are both substituted with a nitrile group while molecule 470 is substituted with a primary amide group. These groups are EWGs and therefore would be expected to increase their respective reduction potentials, which can be seen with the $E_{2+/0}$ values of 5.657 V, 5.720 V, and 5.577 V for molecules 325, 470, and 152, respectively. Of these top five molecules with the most negative solvation free energy, molecules 238 and 141 are outliers from the others, given their structural differences related to the *sp*-hybridized carbon atoms. Molecule 238 is also an outlier with its $\Delta G_{solvation}$ value that is much more negative than the other randomly generated molecules.

5.7 Ranking of Initial Generated Molecules

Combining the windows of $E_{2+/0}$, $E_{0/2-}$, E_{cell}, $\Delta G_{stability}$, and $\Delta G_{solvation}$, the population was able to be narrowed down to only those whose values fell inside all of the desired windows, reducing the population size from 360 to 156 molecules. Each molecule in the reduced population was then ranked based on its relative scoring function as described in Chapter 4.10 to further reduce the population into two smaller populations from each of the given scoring functions. This helped to determine which molecules may be ideal candidates as redox-active materials for use in non-aqueous RFBs based on their cell potential, thermodynamic stability, and solva-

tion free energy altogether. Each of these values were weighted equally in the scoring function and either compared to the unsubstituted BPY molecule ($S_{BPYref.}$) as seen in Figure 2.3 or the average values of the randomly generated data set ($S_{avg.ref.}$) depending on the scoring function used. For the scoring function that references the BPY molecule, any molecule with a score greater than 1.000 is said to be a better candidate than the reference itself. Those molecules that are deemed better candidates based on their score may not necessarily be better than BPY in all three parameters of the scoring function, but overall they show to be better based on their total score. As for the second scoring function that references the average and standard deviation values of the randomly generated set of molecules that fall within the calculated windows for reduction and cell potential, stability, and solvation free energy, any molecule with a score greater than 0.000 are considered possible candidates because their calculated values for these parameters are greater than the average values. Whereas the score of BPY is 1.000 by definition in the first scoring function, it is 0.750 in the second scoring function, showing that BPY itself is better than the average of the randomly generated set of molecules.

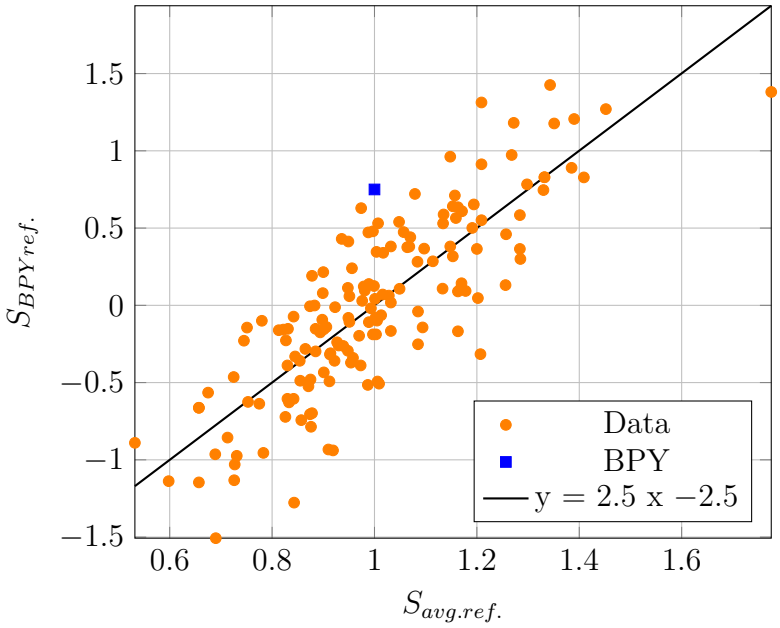


Figure 5.12: Plot of $S_{BPYref.}$ vs. $S_{avg.ref.}$.

Figure 5.12 shows a plot of $S_{BPYref.}$ vs. $S_{avg.ref.}$ where a positive correlation can be seen between these two values. This correlation shows that if a molecule scores high with one scoring function, it typically also scores high with the other due to their linear relation. The same is true if a molecule has a low score with one scoring function, it will also have a low score with the other. Overall, it should not overly matter which scoring function is used to rank the molecules, however, the relative order of the ranking may vary between the two scoring functions.

Table 5.6: Comparison of score values using two different scoring functions.

Label	$S_{BPYref.}$	$S_{avg.ref.}$
5-N-1-2-3-4-DB-1-FG–5,9-L100	0.906	-0.140
7-N-1-3-4-DB-2-FG–4,8-L100	1.272	1.181
11-N-1-2-3-4-5-DB-6-FG–5,9-L2	1.202	0.047
19-N-4-DB-1-2-FG–5,1-L100	0.874	-0.705
29-N-1-3-5-DB-2-FG–5,8-L100	1.351	1.177
47-N-1-2-3-4-DB-1-5-FG–3,5-L5	0.874	-0.006
51-N-2-3-5-DB-2-FG–4,4-L100	0.989	0.138
52-N-1-DB-2-4-FG–6,4-L2	0.898	-0.093
53-N-1-2-3-4-DB-2-FG–5,7-L100	0.974	0.629
54-N-1-3-4-6-DB-1-4-FG–3,1-L2	1.009	-0.507
55-N-3-4-DB-1-FG–2,5-L100	0.923	-0.012
58-N-1-3-6-DB-1-4-FG–2,4-L3	1.005	-0.099
59-N-6-DB-1-3-FG–1,6-L3	1.135	0.588
61-N-2-3-5-DB-3-FG–1,5-L3	0.999	0.125
65-N-3-DB-1-3-4-FG–1,9-L100	0.657	-1.146
BPY	1.000	0.750

To compare particular values from the two scoring functions, Table 5.6 shows

score values from the two different scoring functions for the first 15 randomly generated molecules. For the molecules shown in this table, the correlation between these two scoring functions holds true, however, there are some molecules such as molecules 51, 53, 54, 58, and 61 that have a score that fits the desired outcome of one scoring function but not the other. This is expected since the correlation between the two scoring functions is not a perfect correlation, therefore it must be kept in mind that some molecules may be better than the average of the randomly generated data set but not the reference BPY molecule, and vice versa. The top scoring molecules from each of the two scoring functions will now be reported.

Table 5.7: Top scoring molecules from the random generation based on $S_{BPYref.}$.

Rank	Label	E_{cell} (V)	$\Delta G_{stab.}$ (kJ/mol)	$\Delta G_{solv.}$ (kJ/mol)	S
1	238-N-3-5-6-DB-1-3-4-FG-1,21-L2	2.997	870.944	-139.359	1.775
2	470-N-2-3-5-6-DB-4-FG-3,3-L2	5.423	831.900	-91.552	1.452
3	325-N-1-3-4-6-DB-1-FG-4,4-L2	4.851	764.945	-93.093	1.409
4	276-N-5-DB-3-FG-3,21-L2	5.306	855.076	-84.567	1.390
5	152-N-3-5-6-DB-1-3-FG-2,4-L3	4.135	851.273	-90.962	1.385
6	29-N-1-3-5-DB-2-FG-5,8-L100	5.185	877.462	-79.985	1.351
7	396-N-2-3-5-6-DB-3-FG-2,8-L2	5.795	910.897	-74.213	1.343
8	386-N-1-3-4-6-DB-2-4-FG-2,11-L2	4.686	820.231	-83.324	1.332
9	453-N-1-3-4-6-DB-2-4-FG-2,11-L2	4.686	820.231	-83.324	1.332
10	265-N-1-3-4-DB-1-2-FG-4,21-L100	4.013	847.372	-85.827	1.330
11	328-N-2-5-DB-3-FG-4,21-L100	4.214	860.717	-80.690	1.298
12	454-N-3-5-DB-1-3-FG-5,3-L100	3.926	743.443	-85.852	1.285
13	271-N-1-2-3-4-DB-1-5-FG-3,4-L2	3.560	791.181	-85.910	1.284
14	444-N-1-3-4-6-DB-1-4-FG-3,4-L2	4.510	786.059	-80.490	1.284
15	7-N-1-3-4-DB-2-FG-4,8-L100	5.465	901.109	-68.761	1.272
	Reference BPY	6.205	864.277	-36.343	1.000

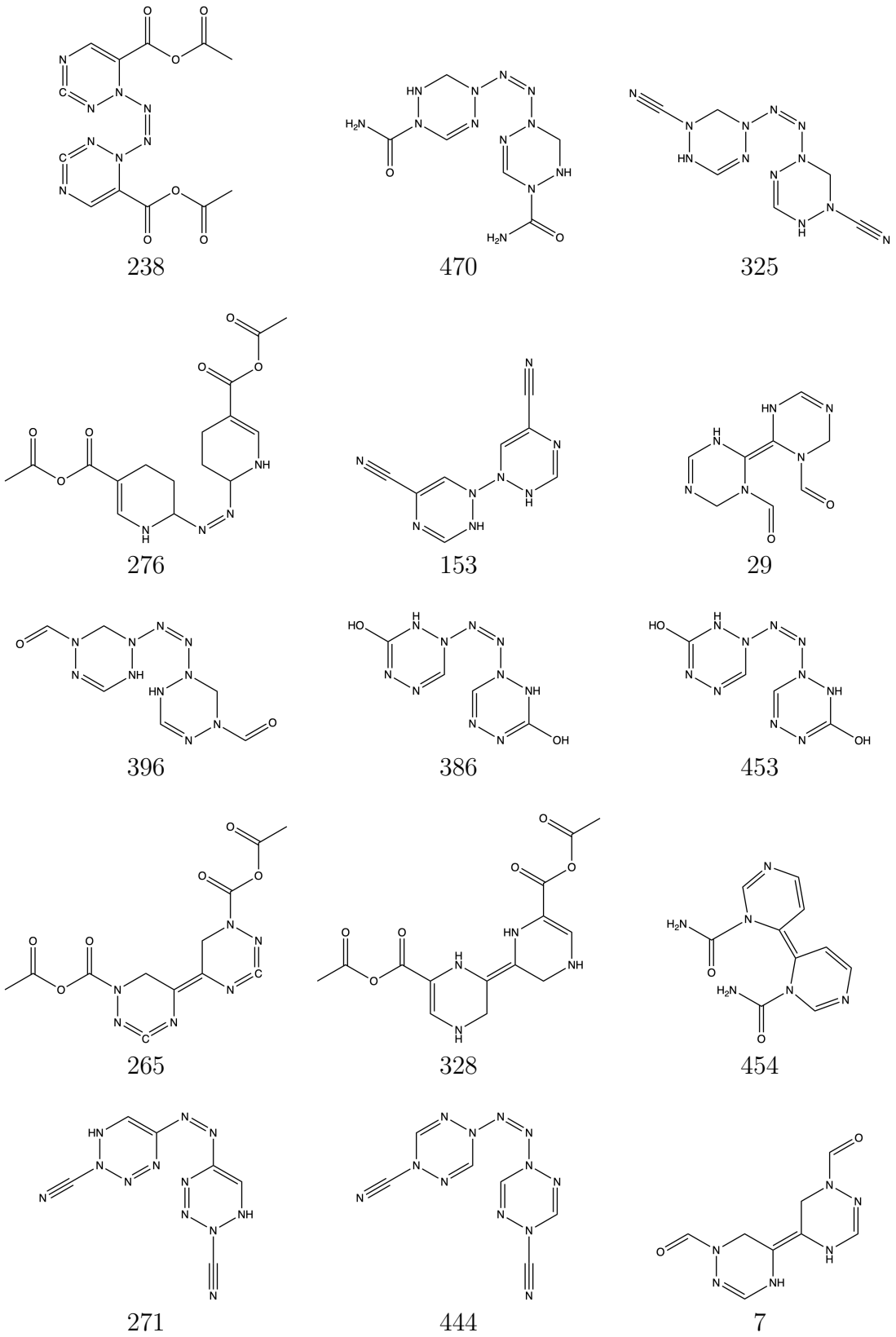


Figure 5.13: Structures of the top scoring molecules from the random generation based on $S_{BPYref.}$.

Table 5.7 shows the ranking of the top 15 randomly generated molecules based on their calculated value of $S_{BPYref.}$. The molecules in this table can be compared to the values of the reference BPY molecule used in the scoring function, which are 6.205 V, 864.277 kJ/mol, and -36.344 kJ/mol for its E_{cell}, $\Delta G_{stability}$, and $\Delta G_{solvation}$, respectively. The top-ranking molecule, molecule 283, has good stability and solubility compared to the values of the reference BPY molecule, however, the cell potential is lower than would would typically be desired compared to the cell potential values of other high ranking molecules. The solvation free energy of this molecule is also much more negative than the next highest ranking molecules, making it an outlier from the rest of the data and therefore not actually a good candidate. All of the top-ranking molecules have a more negative and therefore better solvation free energy value than the reference BPY.

Looking at the structural features of the molecules in Figure 5.13 which correspond to those seen in Table 5.7, all of the molecules except molecule 276 have 2-4 nitrogen atoms per ring and all have 1-2 double bonds per ring except for molecule 238. While duplicate molecules were less unlikely to occur given the amount of freedom in how these structures were constructed, they were still possible and this can be seen with the 8th and 9th ranked molecules 386 and 453 which are structurally identical. While duplicate molecules would not typically be desired, these may actually be beneficial in the GA process since these highly ranked duplicate molecules would have more chances of passing on their desirable traits throughout the GA generations. Some of these molecules are structurally very similar but not identical, for example ranked molecules 271 and 444, which are ranked 13^{th} and 14^{th}, respectively. Both of these molecules have a nitrile group at position 3 and their two rings are connected by the same dinitrogen linker. They also both have four nitrogen atoms and two double bonds in each ring, which only differ by their positions throughout the rings. Overall molecules 444 and 271 have the same score, however, while molecule

444 has a higher cell potential by 0.950 V, molecule 271 is ranked higher and has a slightly better stability with a difference of 5.122 kJ/mol and solvation free energy with a difference of 5.420 kJ/mol.

Table 5.8: Top scoring molecules from the random generation based on $S_{avg.ref.}$.

Rank	Label	E_{cell} (V)	$\Delta G_{stab.}$ (kJ/mol)	$\Delta G_{solv.}$ (kJ/mol)	S
1	396-N-2-3-5-6-DB-3-FG-2,8-L2	5.795	910.897	-74.213	1.426
2	238-N-3-5-6-DB-1-3-4-FG-1,21-L2	2.997	870.944	-139.359	1.381
3	477-N-4-6-DB-4-FG-3,13-L2	5.611	964.513	-58.414	1.313
4	470-N-2-3-5-6-DB-4-FG-3,3-L2	5.423	831.900	-91.553	1.270
5	276-N-5-DB-3-FG-3,21-L2	5.306	855.076	-84.567	1.206
6	7-N-1-3-4-DB-2-FG-4,8-L100	5.465	901.109	-68.761	1.181
7	29-N-1-3-5-DB-2-FG-5,8-L100	5.185	877.462	-79.985	1.177
8	555-N-1-2-3-DB-4-FG-5,7-L5	5.850	810.827	-69.931	0.973
9	646-N-1-2-4-6-DB-FG-5,4-L2	5.697	885.087	-54.621	0.962
10	326-N-1-3-4-5-DB-1-FG-6,15-L2	5.718	835.087	-63.244	0.913
11	152-N-3-5-6-DB-1-3-FG-2,4-L3	4.135	851.273	-90.962	0.890
12	386-N-1-3-4-6-DB-2-4-FG-2,11-L2	4.686	820.231	-83.324	0.829
13	453-N-1-3-4-6-DB-2-4-FG-2,11-L2	4.686	820.231	-83.324	0.829
14	325-N-1-3-4-6-DB-1-FG-4,4-L2	4.851	764.945	-93.093	0.828
15	328-N-2-5-DB-3-FG-4,21-L100	4.214	860.717	-80.690	0.783
	Average Reference Values	4.384	775.285	-50.742	0.000

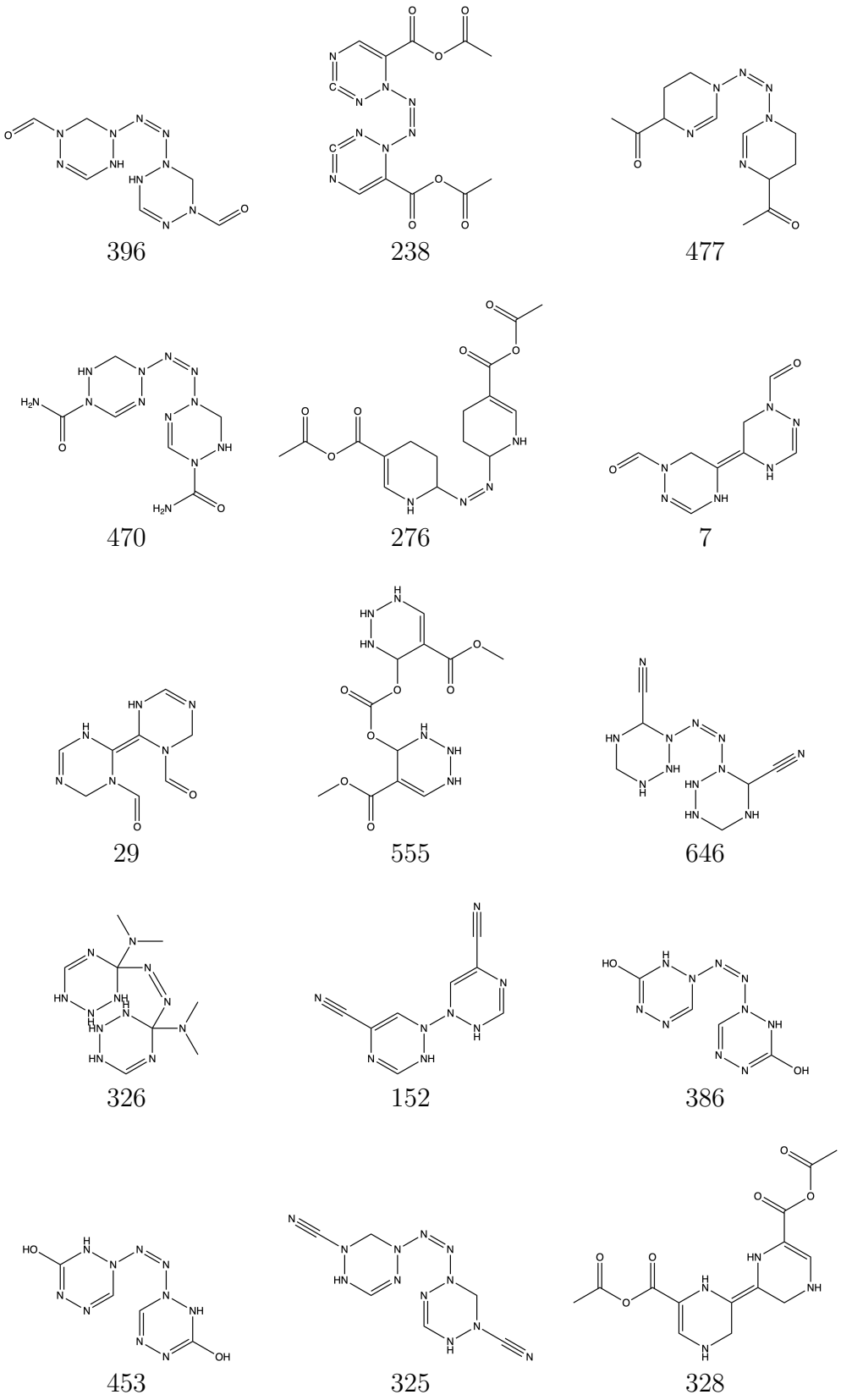


Figure 5.14: Structures of the top scoring molecules from the random generation based on $S_{avg.ref.}$.

Table 5.8 shows the ranking of the top 15 randomly generated molecules based on their calculated values of $S_{avg.ref.}$. Comparing these molecules to the average values of the reduced randomly generated data set, which are 4.384 V, 775.295 kJ/mol, and -50.742 kJ/mol for E_{cell}, $\Delta G_{stability}$, and $\Delta G_{solvation}$, respectively, all of the values of $\Delta G_{stability}$ from the top 15 molecules except for molecule 325 are higher than the average value of the data set and all of the values of $\Delta G_{solvation}$ are more negative, as desired. This is not the case for E_{cell}, however, where some molecules have a lower cell potential value than the average of the data set. However, overall based on their general scores, they are still good candidates as redox-active materials for non-aqueous RFBs.

Similarly to the structures seen in Figure 5.13, most of the structures of the top 15 molecules presented in Figure 5.14 also have 2-4 nitrogen per ring and 1-2 double bonds per ring. The exceptions to these are molecule 238 with three double bonds per ring and molecule 276 with one nitrogen atom per ring. Molecule 646 also has no double bonds present throughout its rings. The duplicate molecules seen in Table 5.7 also appear in Table 5.8 but they are ranked lower at 12th and 13th. Of the top 15 molecules, 10 of these molecules are present in both tables but they are ranked at different positions among the other top ranked molecules.

5.8 Genetic Algorithm

A GA was designed for this project with the goal of improving the randomly generated set of molecules by combining features of two high scoring molecules to generate entirely new molecules. For this project, a GA was used in two different runs; the first GA ranked the molecules based on their $S_{BPYref.}$ values and the second GA ranked them based on their $S_{avg.ref.}$ values. After each generation, DFT calculations were performed on each newly generated molecule to determine their reduction

and cell potentials, stability, and solvation free energy as outlined in Chapter 4.3. Next, these molecule were evaluated to determine whether they fell within the given window for reduction and cell potentials, stability, and solvation free energy, and then ranked with their respective scoring functions for the given GA. Any molecules that fell within the desired windows and had a score higher than 1.000 for $S_{BPYref.}$ or 0.000 for $S_{avg.ref.}$ were added to the population for the next generation and the equivalent number of molecules that ranked lower than these were removed, keeping the total number of molecules in the population the same throughout each generation. The goal of this process was to maximize the score of the molecules in each subsequent generation.

5.8.1 Characteristics of GA Generated Molecules

In total from the two GAs that were run, 300 new molecules were generated; 200 from the first GA and 100 from the second GA. The characteristics of these molecules will now be compared with those from the initial randomly generated population to observe how the prevalence of certain structural characteristics may have changed throughout the application of the GAs. These comparisons give insight into the characteristics of generated populations as a whole, but not necessarily which are characteristics of the "best" molecules.

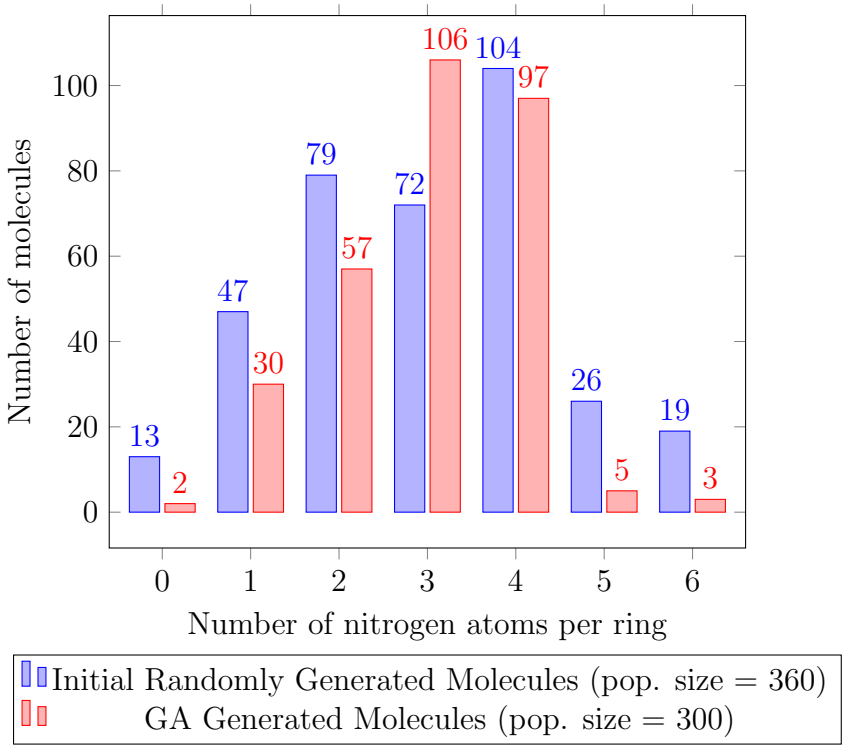


Figure 5.15: Distribution of generated structures by the number of nitrogen atoms present in each ring.

Looking at Figure 5.15, the GA generated molecules follow a similar distribution of the number of nitrogen atom present in each ring as for the initial randomly generated molecules. While for the initial generated molecules the most prevalent number of nitrogen atoms was four, for the GA generated molecules this changed where three nitrogen atoms per ring are present most often in these molecules, however, four nitrogen atoms per ring are still seen in a large portion of the GA generated molecules. There are very few GA generated molecules that had zero, five, or six nitrogen atoms in each of their rings, less than 2% of molecules each, showing that potential candidates prefer to have 1-4 nitrogen atoms present in each ring.

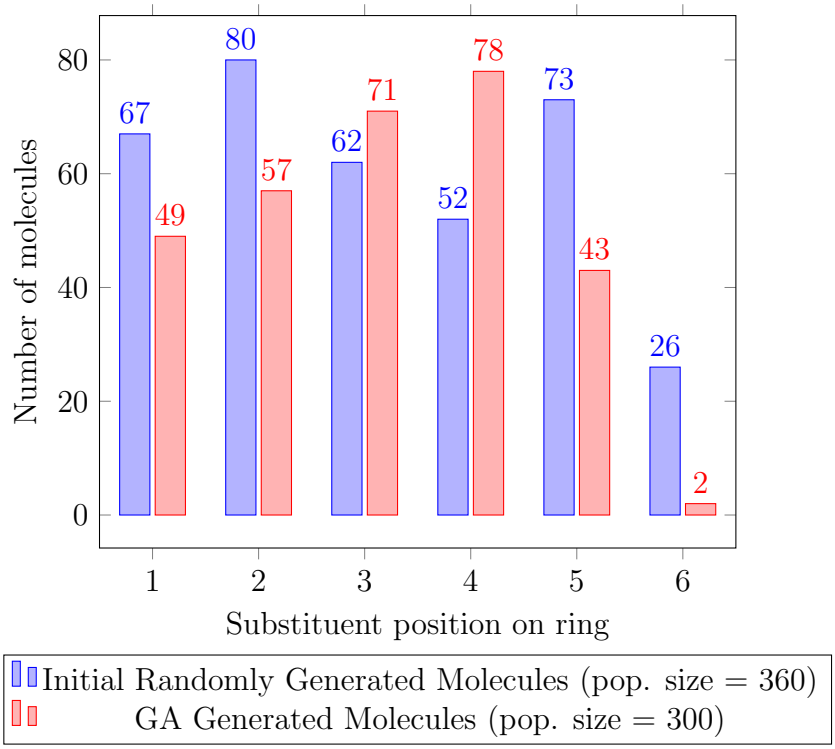


Figure 5.16: Distribution of generated molecules by the position of the substituent on the rings.

There are some changes in the distribution of the positions of the substituents on the ring as shown in Figure 5.16. While for the initial generated molecules most groups were subtituted at positions 2 and 5, this changed for the GA generated molecules where most groups were substituted at positions 3 and 4. It is also interesting to note that while there were 26 initial randomly generated molecules with substituents on position 6, this decreased to only two molecules for the GA generated molecules therefore showing that position 6 is most likely not an ideal position for substituents given that this is also the position where the linker attaches the two rings, leading to large amounts of steric hinderance within the molecules.

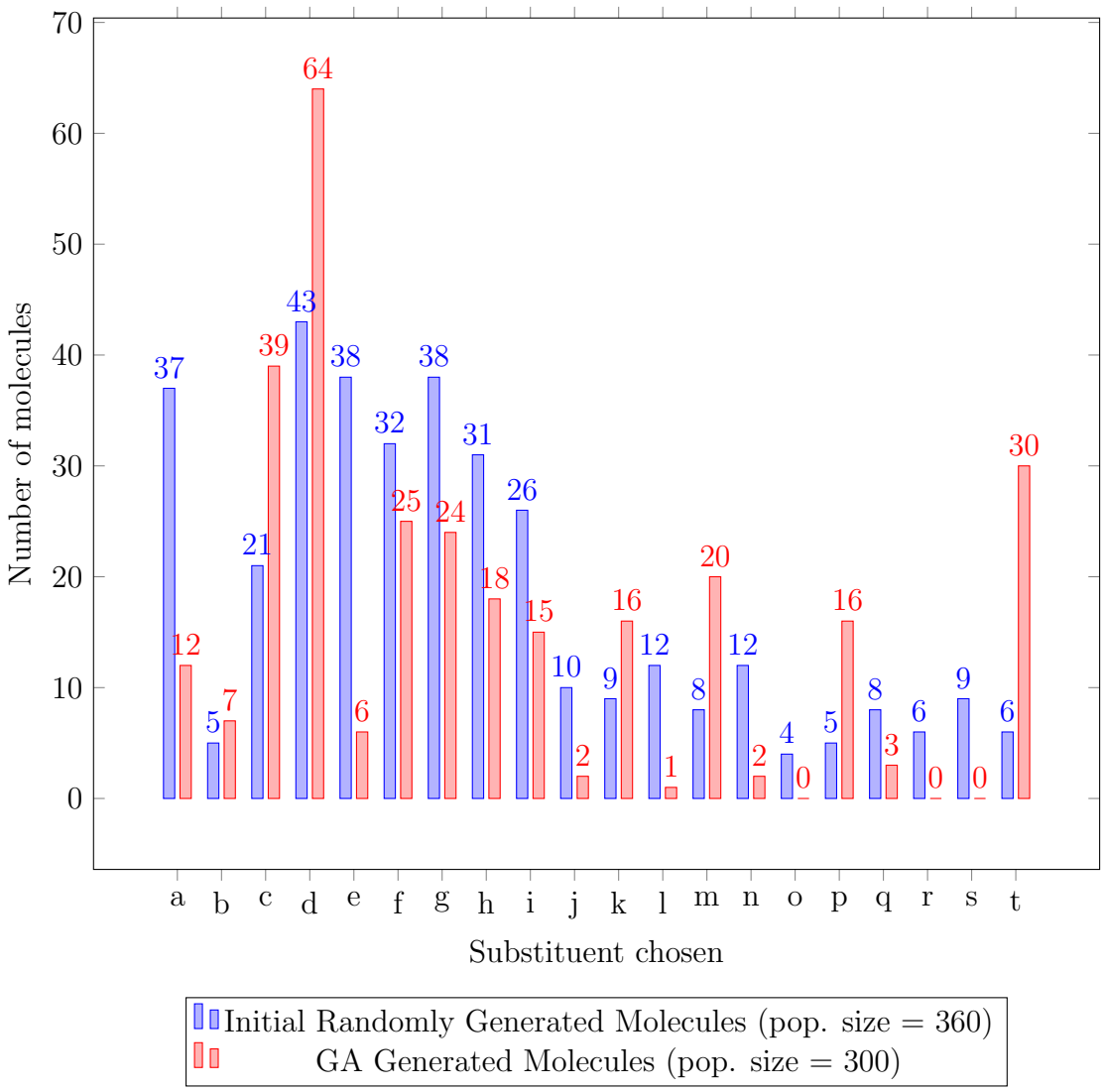


Figure 5.17: Distribution of generated molecules by the number of times each substituent was chosen.

Considerable changes were seen in the distribution of functional groups chosen to be substituted onto the GA generated molecules as seen in Figure 5.17 where the substituents correspond to those seen in Figure 4.6. While substituents (a) and (e) were some of the most prevalent substituents in the initial randomly generated set of molecules, they saw a significant decrease in the number of molecules with these substituents in the GA generated population. Substituents (o), (r), and (s) were not present on many molecules in the initial randomly generated population and these substituents were not passed on to any molecule throughout the GA.

They are therefore likely to not be good substituents for potential active material candidates. There were some increases in the prevalence of some substituents as well as can be seen with substituents (b), (c), (d), (k), (m), (p), and (t), showing that the GAs appeared to favour passing on these substituents over others throughout the generations.

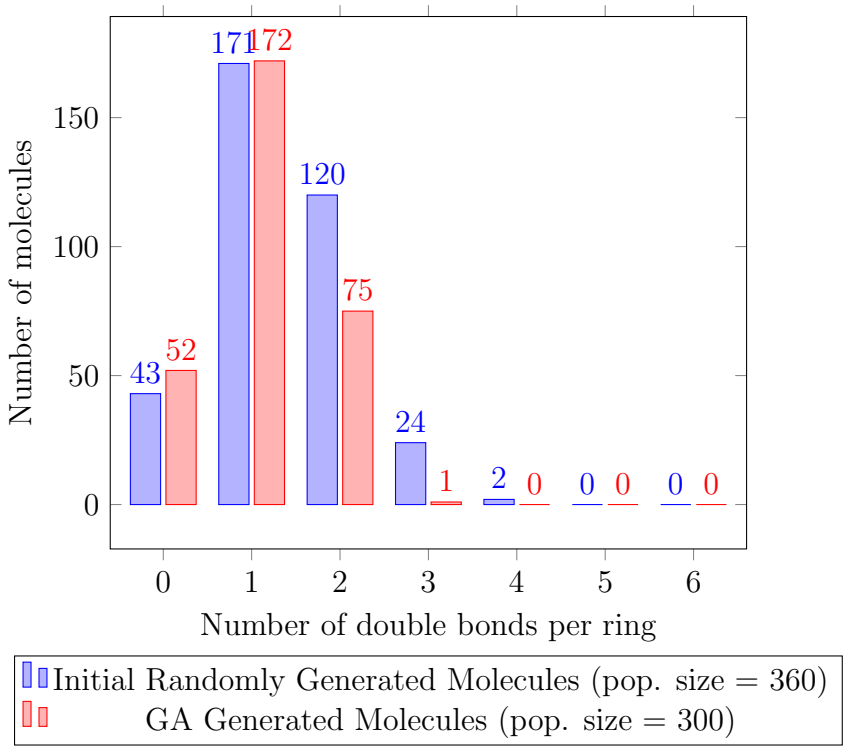


Figure 5.18: Distribution of generated molecules by the number of double bonds present in each ring.

Figure 5.18 shows the distribution of the number of double bonds present in each ring of the initial randomly generated molecules and the GA generated molecules. There is virtually no change in the number of molecules with only one double bond per ring from both sets of molecules, however, there is a slight increase in the number of molecules with no double bonds present in their rings. As for 2-4 double bonds, there was a slight decrease in the number of molecules with two double bonds per ring. While there were 24 molecules in the initial randomly generated population with three double bonds per ring, only one GA generated molecule had three double

bonds in its ring and no molecules were present after the GAs with four double bonds per ring. These decreases are most likely due to the mutations applied during the crossover step of the GA.

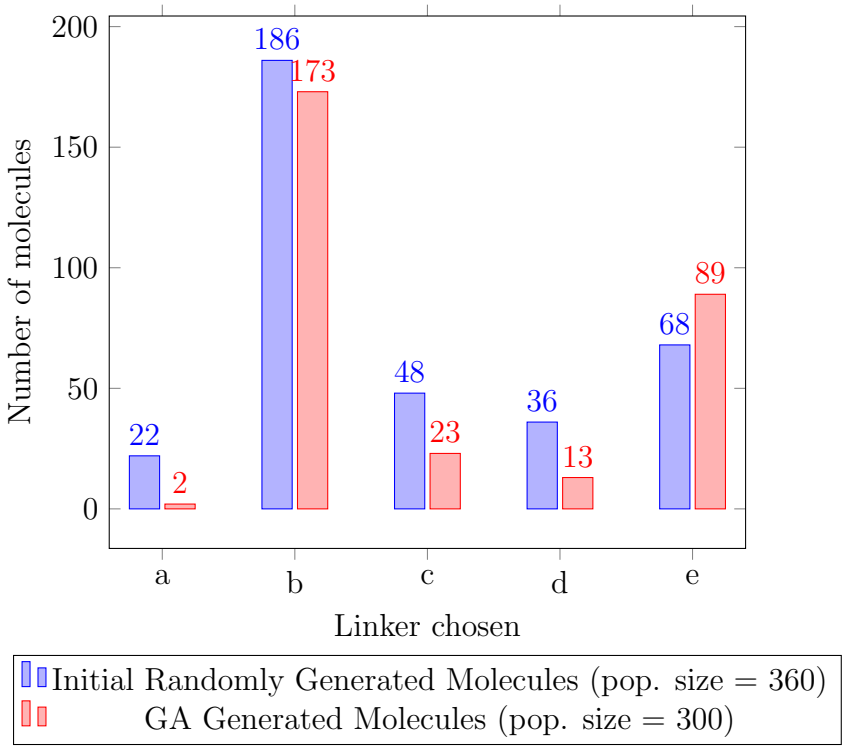


Figure 5.19: Distribution of generated molecules by the linker chosen to connect the two rings.

Lastly, the change in the distributions of the linkers chosen are shown in Figure 5.19 where the linkers correspond to those shown in Figure 4.7. Linker (b) remains the most prevalent linker throughout both populations, however, for the GA generated molecules, linker (e) saw a slight increase.

Overall, changes were seen in the distributions of some characteristics of the generated molecules between the initial randomly generated set of molecules and the GA generated molecules. These changes show that some characteristics may be favoured over others in the GA and that those characteristics may be ones that provide good candidates as active materials for use in non-aqueous RFBs.

5.8.2 First GA

From the set of 156 molecules that fell within the desired windows, 70 of these molecules had a score greater than 1.000 for $S_{BPYref.}$ and therefore made up the initial population for this GA. For this GA, 28 generations were run until the algorithm was considered converged, as no changes were noticed in the high ranking population. The first three generations of this GA each generated 25 molecules, however, from the fourth generation onward, this number was reduced to five molecules per generation with the goal of speeding up the GA, leading to a total of 200 new molecules being created by this GA. From these 200 molecules, only 99 molecules had a score high enough to be added to the GA population. Of these 99 GA generated molecules, 48 remained in the population of 70 molecules after all 28 generations were run, making up almost 69% of the final GA population.

Table 5.9 shows the 15 top ranked molecules from the first GA that was run based on their respective score values using $S_{BPYref.}$. Of these top 15 molecules, three molecules are from the initial GA population, marked by Gen. #0, and the remainder were generated throughout various GA generations. In this first GA, 28 generations were run, however, the last generation to have a new molecule in the top 15 molecules was generation 26 which is ranked 9th. For this reason, the GA was considered converged after 28 generations.

Table 5.9: Top 15 molecules from the first GA ranked based on $S_{BPY_{ref.}}$ values. Generation #0 is the initial GA population before the GA was run.

Rank	Gen. #	Label	E_{cell} (V)	$\Delta G_{stab.}$ (kJ/mol)	$\Delta G_{solv.}$ (kJ/mol)
1	0	238-N-3-5-6-DB-1-3-4-FG–1,21-L2	2.997	870.944	-139.359
2	3	716-N-1-3-4-DB-2-FG–4,8-L100	5.134	875.338	-112.444
3	17	791-N-1-3-4-DB-2-FG–4,8-L100	5.134	875.338	-112.444
4	3	725-N-1-3-4-5-DB-2-FG–4,4-L100	5.151	790.379	-115.719
5	9	754-N-1-3-4-DB-2-FG–4,4-L100	5.884	793.906	-108.237
6	20	807-N-1-3-4-DB-2-FG–4,4-L100	5.884	793.906	-108.237
7	21	813-N-1-3-4-DB-2-FG–4,4-L100	5.884	793.906	-108.237
8	3	719-N-1-2-3-5-DB-4-FG–1,13-L2	5.903	786.649	-100.952
9	26	837-N-1-3-4-DB-2-FG–4,21-L100	5.064	850.661	-97.893
10	9	755-N-1-3-4-DB-FG–3,4-L2	5.635	829.306	-94.072
11	8	749-N-1-3-5-DB-2-FG–5,8-L100	5.321	874.609	-92.312
12	0	470-N-2-3-5-6-DB-4-FG–3,3-L2	5.423	831.900	-91.553
13	5	733-N-1-2-3-5-DB-4-6-FG–3,3-L2	4.588	783.646	-96.182
14	2	684-N-1-2-3-5-DB-4-FG–1,7-L100	5.354	835.535	-88.126
15	0	325-N-1-3-4-6-DB-1-FG–4,4-L2	4.851	764.945	-93.093

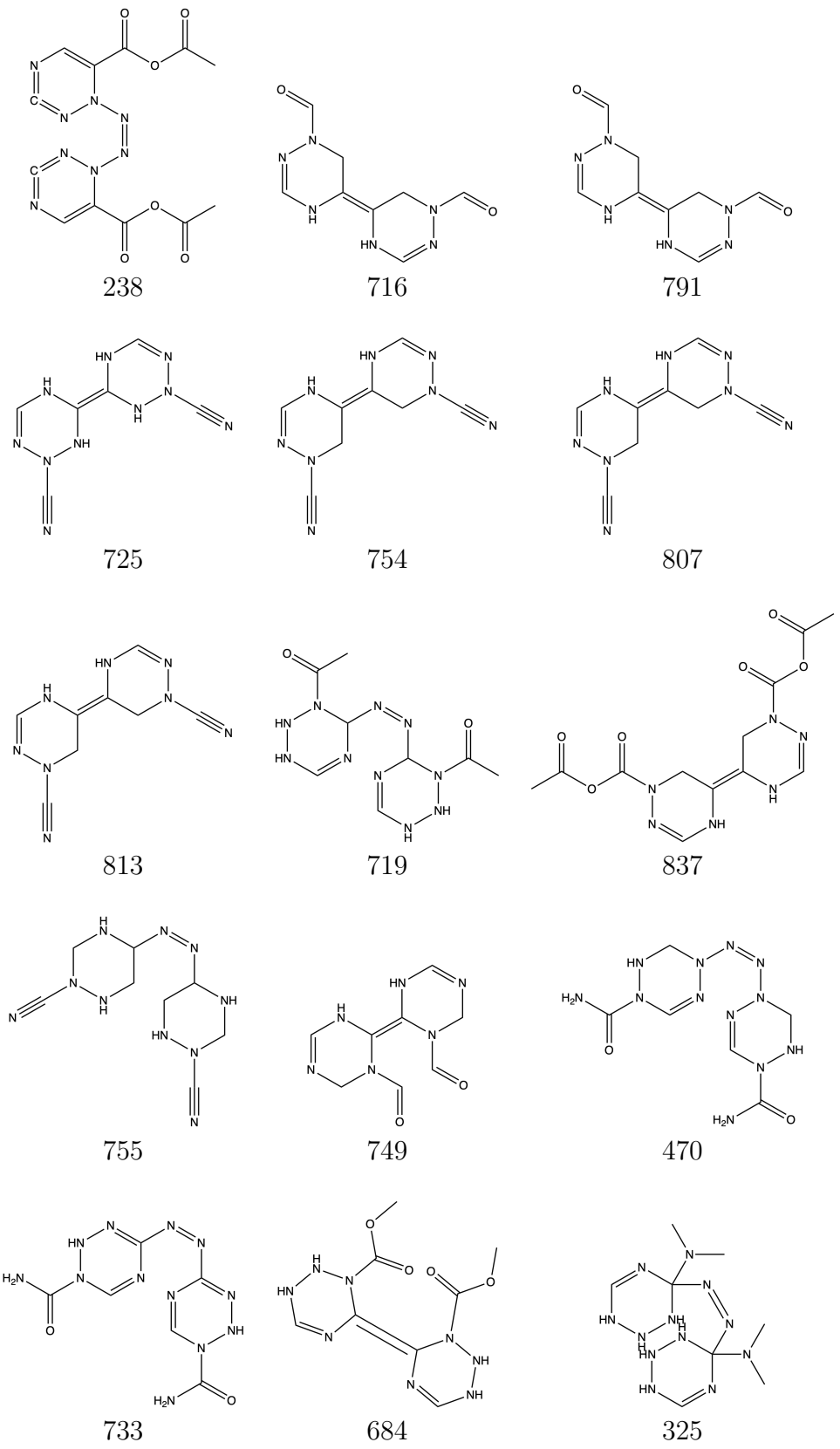


Figure 5.20: Structures of the top 15 molecules generated from the first GA.

Figure 5.20 shows the structures of the top ranked molecules from Table 5.9. It appears that there are a couple sets of duplicate molecules in these top ranked molecules. The first set are molecules 716 and 791, which were generated in generations 3 and 17, respectively. Molecule 716 was the result of the random combination of two molecules, however, molecule 791 became a duplicate molecule because of the crossing over of molecule 716 with another molecule that shared the identical backbone and only differed by the substituent and its position. Therefore as a result of crossover of these two molecules, molecule 791 became a duplicate of molecule 716. The second set of identical molecules are molecules 754, 807, and 813 from generations 9, 20, and 21, respectively. Molecule 807 became a duplicate of molecule 754 for the same reason as the previously mentioned set, however, molecule 813 was the result of two random molecules that were crossed over to give a combination of features that made molecule 813 a duplicate. Some duplicate molecules that exist at lower ranking in the GA population are the result of the same molecule being crossed over with itself. A parameter may be implemented in a future GA to avoid duplicate molecules as the result of the same molecule being paired with itself in the crossover step, however, in a GA duplicate molecules may not necessarily be a negative addition to the population. There are some molecules among this population with structural similarities such as the position of nitrogen atoms or double bonds that despite the differences in their values of cell potential, stability, and solvation free energy, rank closely to each other in some cases.

5.8.3 Second GA

For the second GA, the population was determined and ranked based on the $S_{avg.ref.}$ scoring function that references the average values of the initial randomly generated population. From the set of 156 molecules that are within the windows of reduction and cell potentials, stability, and solvation free energy, 74 molecules had a

score greater than 0.000 for $S_{avg.ref.}$, allowing these molecules to make up the initial GA population. For this GA, 20 generations were run with 100 molecules generated in total as each generation of this GA generated five new molecules per generation. Of the 100 GA generated molecules, 50 molecules fell within the desired windows and had a score high enough to be added to the GA. Although 50 molecules had the potential to be added to the GA, only 36 GA generated molecules remained in the GA population of 74 molecules after all 20 generations, making up almost 49% of the final GA population.

Table 5.10 shows the top 15 ranked molecules from the second GA where the $S_{avg.ref.}$ scoring function was used to rank these molecules. Seven of these top-ranking molecules are from the initial GA population, leaving the remaining 8 molecules as the result of one of the 20 GA generations.

Table 5.10: Top 15 molecules from the second GA ranked based on $S_{avg.ref.}$ values. Generation #0 is the initial GA population before the GA was run.

Rank	Gen. #	Label	E_{cell} (V)	$\Delta G_{stab.}$ (kJ/mol)	$\Delta G_{solv.}$ (kJ/mol)
1	0	396-N-2-3-5-6-DB-3-FG-2,8-L2	5.795	910.897	-74.213
2	0	238-N-3-5-6-DB-1-3-4-FG-1,21-L2	2.997	870.944	-139.359
3	13	939-N-1-3-4-DB-FG-3,4-L2	5.635	829.306	-94.072
4	13	936-N-1-3-4-6-DB-4-FG-3,6-L2	5.530	881.491	-81.253
5	0	477-N-4-6-DB-4-FG-3,13-L2	5.611	964.513	-58.414
6	0	470-N-2-3-5-6-DB-4-FG-3,3-L2	5.423	831.900	-91.553
7	18	963-N-1-2-4-DB-3-FG-5,4-L100	5.564	815.207	-91.002
8	5	900-N-2-3-5-DB-FG-2,8-L2	5.580	931.029	-61.353
9	17	957-N-2-3-5-DB-FG-2,8-L2	5.580	931.029	-61.353
10	20	971-N-2-3-5-DB-FG-2,8-L2	5.580	931.029	-61.353
11	0	276-N-5-DB-3-FG-3,21-L2	5.306	855.076	-84.567
12	0	7-N-1-3-4-DB-2-FG-4,8-L100	5.465	901.109	-68.761
13	0	29-N-1-3-5-DB-2-FG-5,8-L100	5.185	877.462	-79.985
14	12	931-N-3-5-DB-1-4-FG-3,3-L2	5.227	815.416	-92.938
15	1	878-N-1-2-3-5-DB-5-FG-1,13-L2	5.336	835.576	-85.373

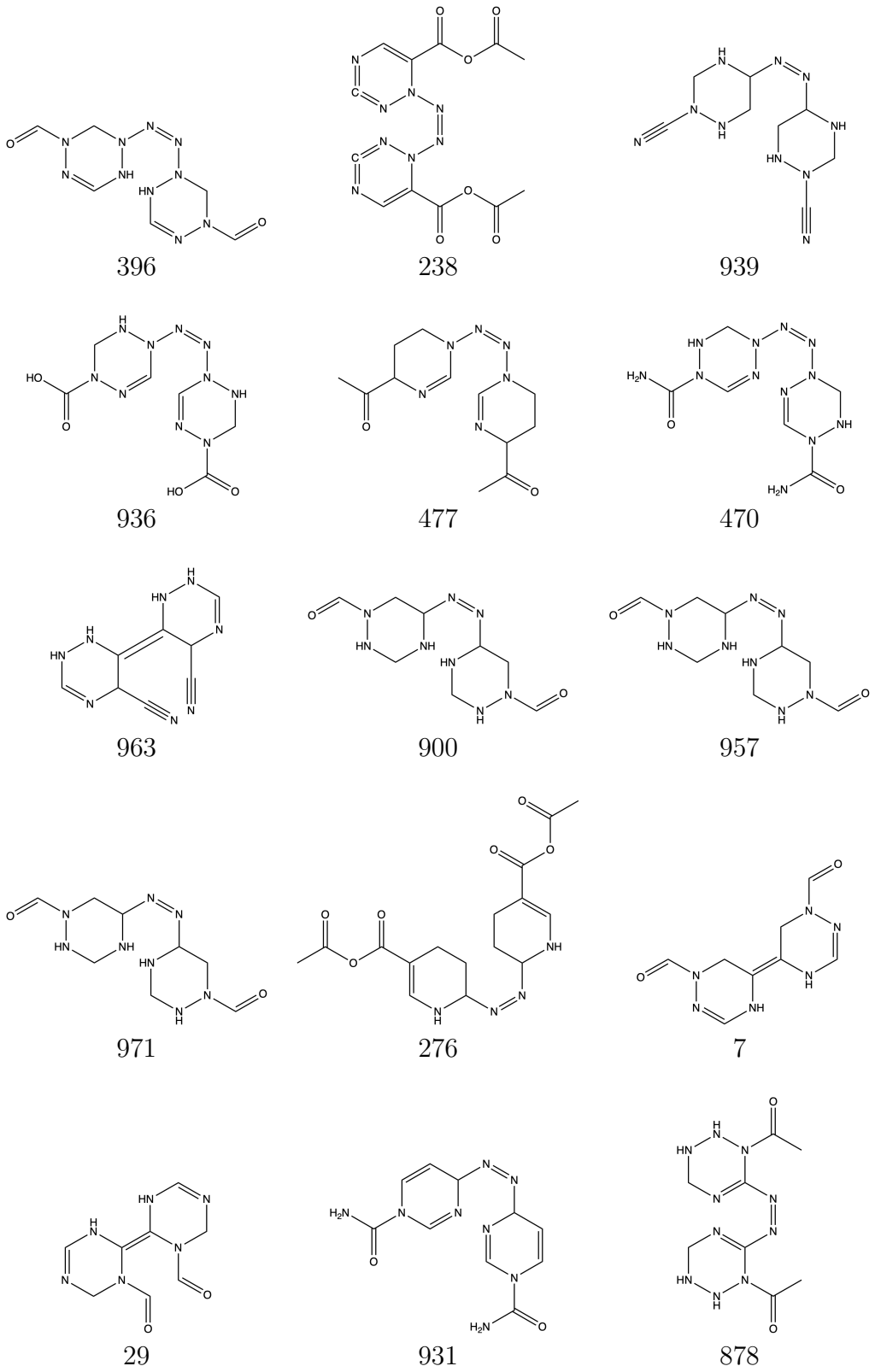
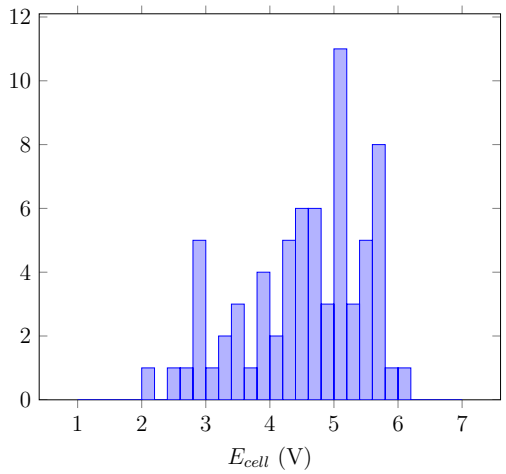


Figure 5.21: Structures of the top 15 molecules generated from the second GA.

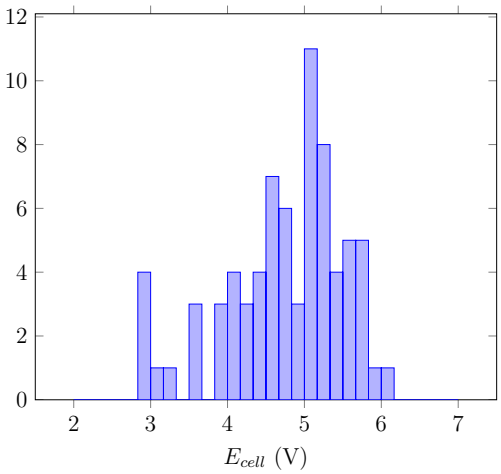
The structures of the top-ranking molecules from Table 5.10 are shown in Figure 5.21. Among these top 15 ranked molecules, there is only one set of identical molecules which are molecules 900, 957, and 971 from generations 5, 17, and 20, respectively. Molecule 900 is the result of crossover from two randomly chosen molecules from the initial GA population. Molecule 957 on the other hand is the result of molecule 900 crossing over with another molecule where the features of molecule 900 happen to be chosen over those of the other molecule in the random crossover event. The same reasoning can be applied to molecule 971 except molecule 957 passed on its traits instead. Although only one set of identical molecules appear in the top 15 ranked molecules of this second GA, others are to be expected in the rest of the population outside of the top 15 molecules as well.

5.8.4 Comparison of First and Second GAs

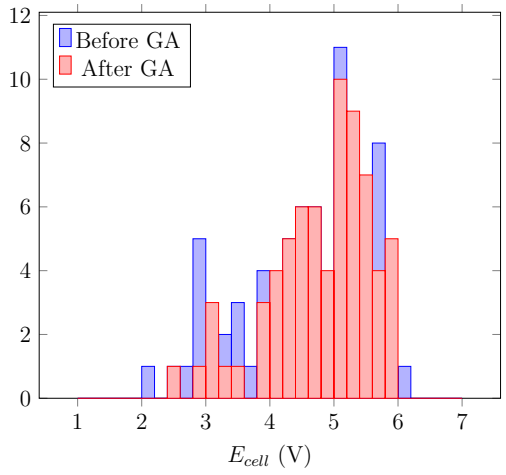
When comparing the outcomes of the two GAs, one may look at how the distributions of the two different populations changed throughout the generations of each GA. Shown in this section are the changes in distribution of E_{cell}, $\Delta G_{stability}$, $\Delta G_{solvation}$, and S values for the two GA populations before and after the operation of the GA.



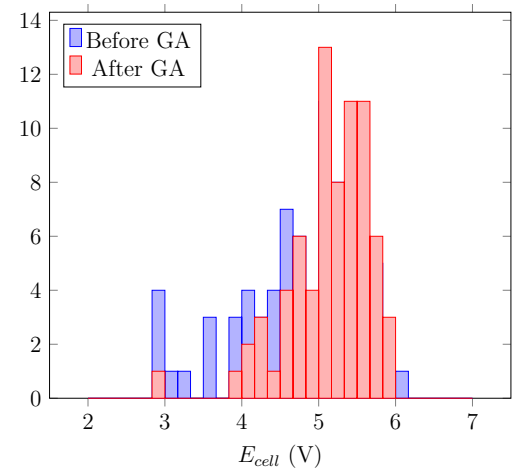
(a) Before 1st GA



(b) Before 2nd GA



(c) Last Generation of 1st GA



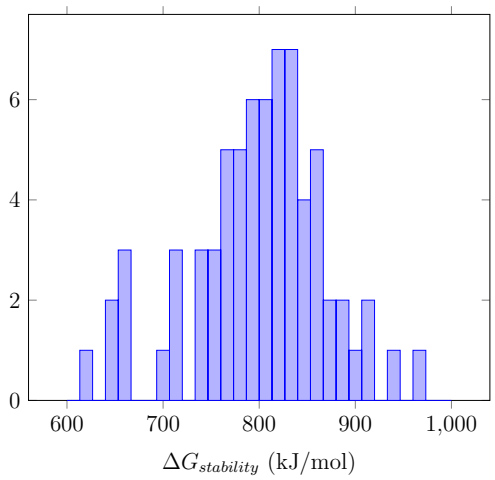
(d) Last Generation of 2nd GA

Figure 5.22: Distribution of E_{cell}. (a) Before the first GA (b) Before the second GA (c) After the last (28^{th}) generation of the first GA (d) After the last (20^{th}) generation of the second GA.

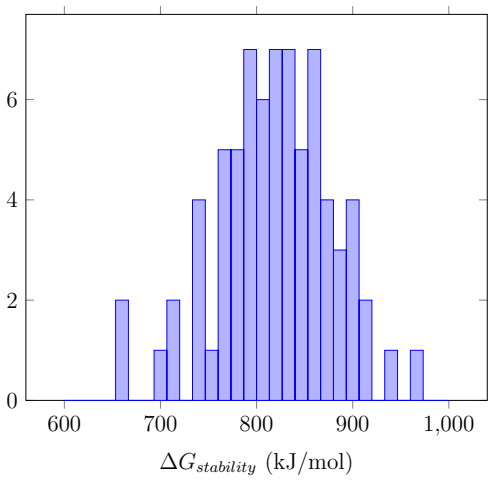
Figure 5.22 shows the change in distribution of E_{cell} values for the two GA populations before any generations are run and after the last generation of each GA where the population size for the first GA is 70 molecules and 74 molecules for the second GA. Looking at Figure 5.22a before the first GA was run, the overall distribution lies from around 2.0 V to just above 6.0 V with most of the distribution of E_{cell} laying between 4.0 to 6.0 V. After the last generation of this GA (Figure

5.22c), there is a shift towards higher values of E_{cell} with most of the population lying between 4.0 to 6.0 V and only a few values around 2.5 to 3.5 V. This shift shows that through the use of the GA, the average cell potential values of the population are able to shift towards higher and therefore more favourable values.

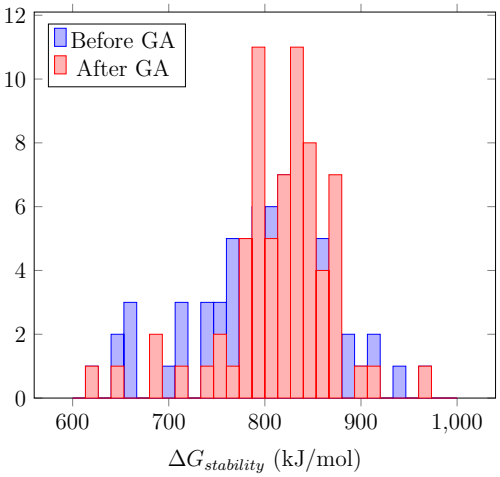
As for the second GA before the first generation was run (Figure 5.22b), the values of E_{cell} range from just below 3.0 V to just above 6.0 V. After the last generation, the range of values are from around 4.0 to 6.0 V with one outlier around 3.0 V as seen in Figure 5.22d. Most of the distribution now lies between 5.0 to 6.0 V which is more of a shift in E_{cell} values than seen in the first GA. Overall, from the changes in the distributions of E_{cell} for both GAs, the cell potential values of the two populations are shifting to higher values over the course of the generations.



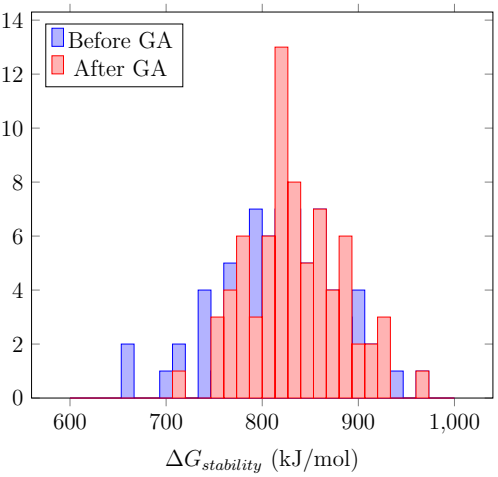
(a) Before 1st GA



(b) Before 2nd GA



(c) Last Generation of 1st GA

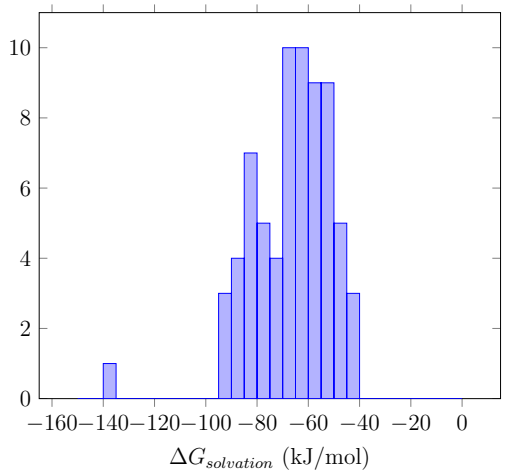


(d) Last Generation of 2nd GA

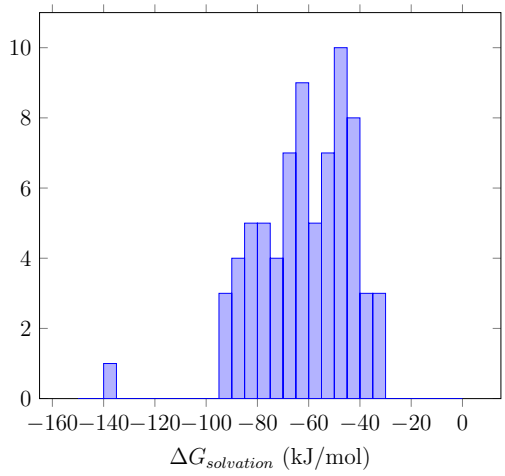
Figure 5.23: Distribution of $\Delta G_{stability}$. (a) Before the first GA (b) Before the second GA (c) After the last (28^{th}) generation of the first GA (d) After the last (20^{th}) generation of the second GA.

Figure 5.23 shows the change in distribution of $\Delta G_{stability}$ from before each GA and after the last generation of the respective GA. Looking at the two distributions before the first (Figure 5.23a) and second (Figure 5.23b) GAs, both have an approximately normal distribution centred around 800 to 850 kJ/mol with their overall distribution ranging in the window of 600 to 1000 kJ/mol. After the last generation of the first GA, most of the distribution shifted to a more narrow window of just

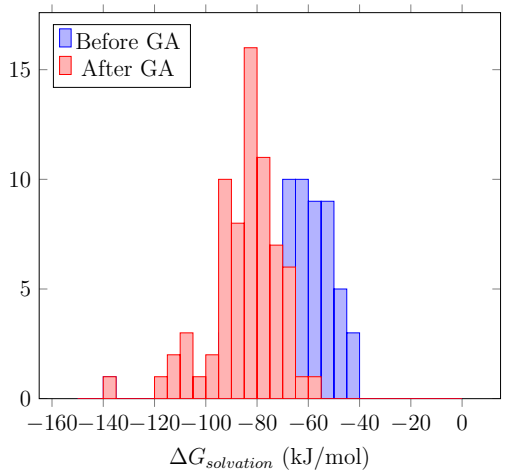
below 800 to 900 kJ/mol as seen in Figure 5.23c. For the second GA, the range of values after the last generation (Figure 5.23d) is wider with most of the distribution lying between 750 to 950 kJ/mol.



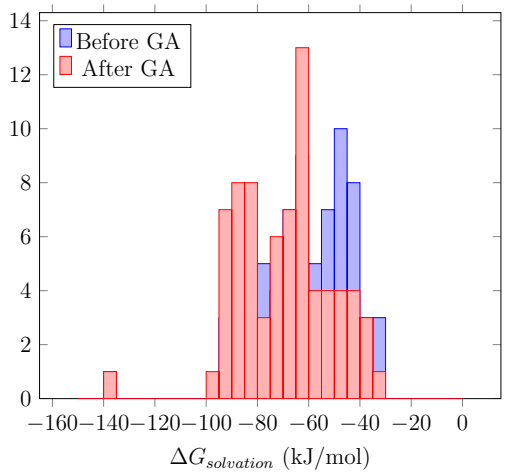
(a) Before 1st GA



(b) Before 2nd GA



(c) Last Generation of 1st GA

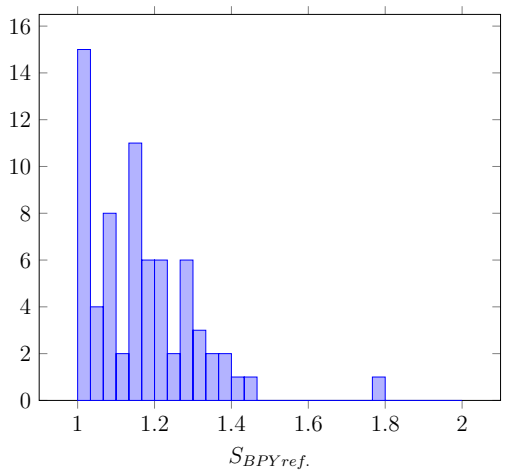


(d) Last Generation of 2nd GA

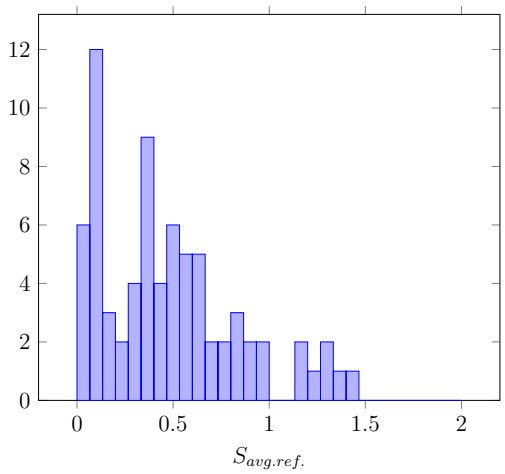
Figure 5.24: Distribution of $\Delta G_{solvation}$. (a) Before the first GA (b) Before the second GA (c) After the last (28^{th}) generation of the first GA (d) After the last (20^{th}) generation of the second GA.

As for the distributions of $\Delta G_{solvation}$ shown in Figure 5.24, before the first GA was run, Figure 5.24a shows that the distribution of this population was mostly between -100 to -40 kJ/mol with one outlier at around -140 kJ/mol that appears

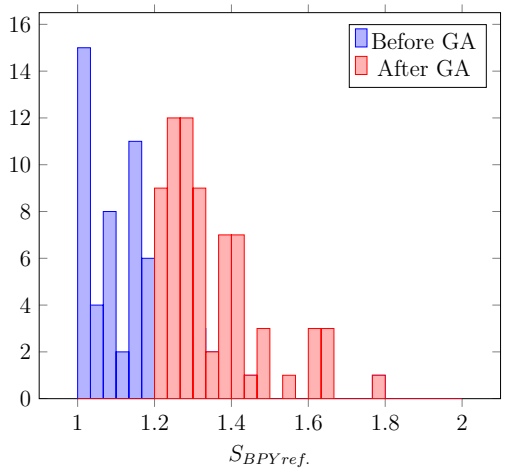
in all four distribution plots. After the last generation of the first GA (Figure 5.24c), this distribution has shifted towards more negative values with most of the distribution still lying between -100 to -60 kJ/mol, however, a few new molecules with $\Delta G_{solvation}$ values around -100 to -120 kJ/mol were generated throughout the GA. Before the second GA, the population had a wider distribution with most of the distribution lying between -100 to -30 kJ/mol as shown in Figure 5.24b. After the last generation (Figure 5.24d), this range did not change overall, but the distribution within this range itself did change, showing that changes were made to the population throughout the GA generations.



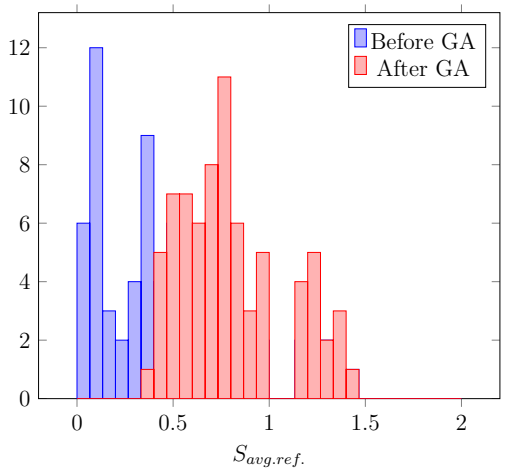
(a) Before 1st GA



(b) Before 2nd GA



(c) Last Generation of 1st GA



(d) Last Generation of 2nd GA

Figure 5.25: Distribution of S. (a) Before the first GA (b) Before the second GA (c) After the last (28^{th}) generation of the first GA (d) After the last (20^{th}) generation of the second GA.

The last distributions are for the score values as shown in Figure 5.25. For the first GA, the minimum value for the score is set at 1.000 because the scoring function used references an unsubstituted BPY molecule therefore a score above 1.000 indicates that the given molecule has better values in one or more of E_{cell}, $\Delta G_{stability}$, or $\Delta G_{solvation}$ than BPY itself. As for the second GA, the minimum score is 0.000 because the scoring function references the average values of a randomly generated molecule, therefore a molecule with a score above 0.000 has better values

in one or more of E_{cell}, $\Delta G_{stability}$, or $\Delta G_{solvation}$ than the average values of the randomly generated population.

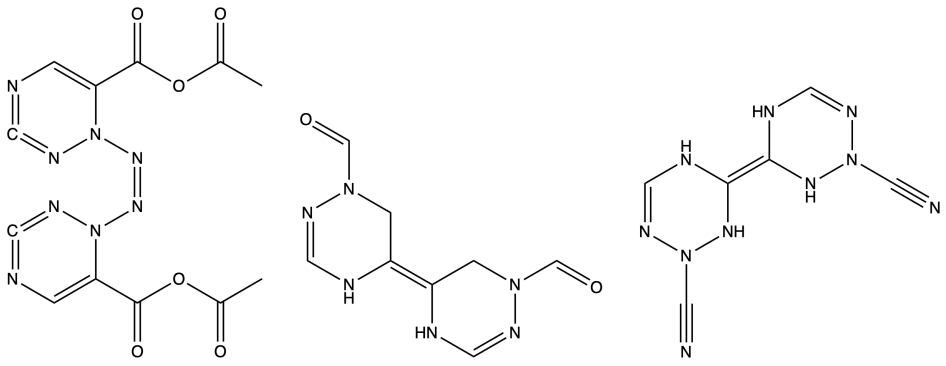
Before the first GA (Figure 5.25a), the distribution for the score values lies between 1.0 to around 1.4 with an outlier close to 1.8, however, most of the distribution is between 1.0 to 1.2. After all of the GA generations were run, there is a shift in the minimum score value from 1.0 to 1.2 with most of the distribution now lying between 1.2 to 1.4 and a few newly generated molecules with a score around 1.6 to 1.7 as seen in Figure 5.25c. As for the second GA, Figure 5.25b shows that most of the distribution is lying between 0.0 and 0.7 upwards of 1.0. After the last generation of this GA (Figure 5.25d), this distribution has shifted the minimum score value from 0.0 to around 0.4 with the most of the distribution in the range of 0.4 to 1.0, however, there are also newly generated molecules with score values around 1.2 to 1.5. With both of these GAs, the minimum score values shift to higher values throughout the GA generations. These shifts in turn show that through the use of the respective GAs, the generated molecules are improving overall throughout the GA generations.

5.9 Proposed Redox-Active Material Candidates

From the top scoring molecules of the two GAs, two sets of molecules can be proposed as potential candidates for use as redox-active materials in non-aqueous RFBs.

Table 5.11: Top scoring molecules from the first GA based on $S_{BPYref.}$.

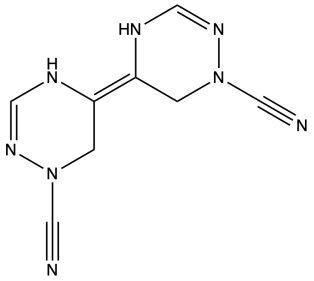
Rank	Label	E_{cell} (V)	$\Delta G_{stability}$ (kJ/mol)	$\Delta G_{solvation}$ (kJ/mol)
1	238-N-3-5-6-DB-1-3-4-FG–1,21-L2	2.997	870.944	-139.359
2	716-N-1-3-4-DB-2-FG–4,8-L100 791-N-1-3-4-DB-2-FG–4,8-L100	5.134	875.338	-112.444
3	725-N-1-3-4-5-DB-2-FG–4,4-L100	5.151	790.379	-115.719
4	754-N-1-3-4-DB-2-FG–4,4-L100 807-N-1-3-4-DB-2-FG–4,4-L100 813-N-1-3-4-DB-2-FG–4,4-L100	5.884	793.906	-108.237
5	719-N-1-2-3-5-DB-4-FG–1,13-L2	5.903	786.649	-100.952
	Referenc BPY	6.205	864.277	-36.344



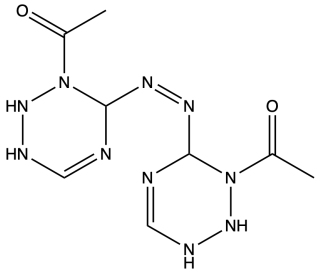
238

716, 791

725



754, 807, 813



719

Figure 5.26: Structures of top scoring molecules from the first GA.

In total, the first GA generated 200 molecules with 99 of these molecules having the potential to be added to the GA population due to their score values. The top five molecules from the first GA have been summarized in Table 5.11 where the duplicate molecules have been grouped together as the same ranking and their structures can be seen in Figure 5.26. Of the top scoring molecules from the first GA, all of the molecules were generated throughout the GA except for molecule 238 that is retained from the first generation and is most likely an outlying structure for which the DFT computation converged successfully despite the *sp*-hybridized carbon atom.

The values of the top scoring molecules in Table 5.11 can be compared to the values from the reference BPY molecule, which are 6.205 V, 864.277 kJ/mol, and -36.344 kJ/mol for E_{cell}, $\Delta G_{stability}$, and $\Delta G_{solvation}$, respectively. The cell potential values of all top five molecules are lower than that of BPY itself, however, the top two molecules have a slightly higher stability value than BPY. As for solvation free energy, all values in this table are significantly more negative than for BPY. As mentioned, the large difference in solvation free energy between the generated molecules and BPY may have caused solvation free energy to have more of an influence on the score values even though E_{cell}, $\Delta G_{stability}$, and $\Delta G_{solvation}$ are weighted equally in the scoring function. Based on the overall score value of these molecules, they would be better candidates for non-aqueous RFB active materials than BPY itself.

Table 5.12: Top scoring molecules from the second GA based on $S_{avg.ref.}$.

Rank	Label	E_{cell} (V)	$\Delta G_{stability}$ (kJ/mol)	$\Delta G_{solvation}$ (kJ/mol)
1	396-N-2-3-5-6-DB-3-FG-2,8-L2	5.795	910.897	-74.213
2	238-N-3-5-6-DB-1-3-4-FG-1,21-L2	2.997	870.944	-139.359
3	939-N-1-3-4-DB-FG-3,4-L2	5.635	829.306	-94.072
4	936-N-1-3-4-6-DB-4-FG-3,6-L2	5.530	881.491	-81.253
5	477-N-4-6-DB-4-FG-3,13-L2	5.611	964.513	-58.414
	Average Reference Values	4.384	775.295	-50.742

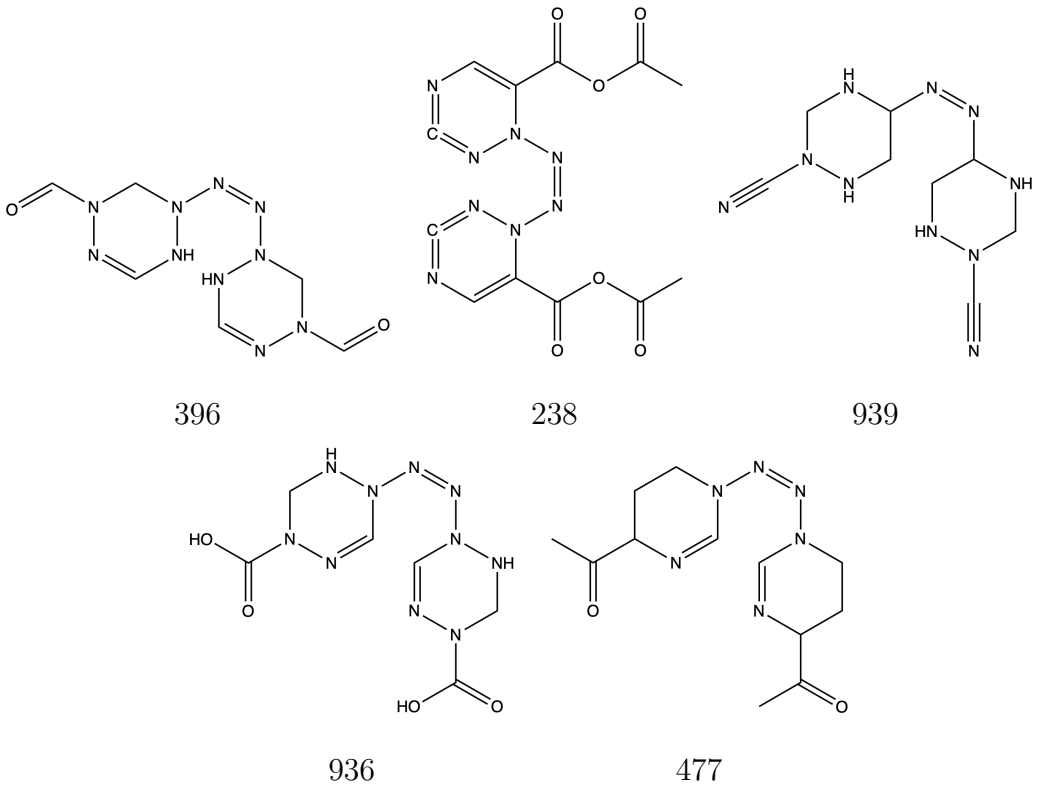


Figure 5.27: Structures of top scoring molecules from the second GA.

As for the second GA, 100 molecules were generated in total with 50 of these molecules with scores values high enough for them to be added to the GA population. Table 5.12 summarizes the top scoring molecules from the second GA and their

structures are shown in Figure 5.27. Unlike the first GA, only two of the top five molecules were generated by the GA and no duplicate molecules fell within these top five scoring molecules. Molecule 238 also lies within the top scoring molecules from the second GA but because of its *sp*-hybridized carbon atom, it would likely not be a good active material candidate despite its high score value.

Comparing the values in Table 5.12 to those of the average reference values, which are 4.384 V, 775.295 kJ/mol, and -50.742 kJ/mol for E_{cell}, $\Delta G_{stability}$, and $\Delta G_{solvation}$, respectively, all of the top scoring molecules have a higher cell potential than the reference values with the exception of molecule 238. All of these molecules do, however, have higher values for stability than the reference as well as more negative solvation free energies. Based on these observations and the overall scores of these molecules, they show better results as potential active materials than the average values of the initial randomly generated set of molecules.

As many of these top scoring molecules have high nitrogen content in their rings, extra ring-stability assessments would help to eliminate any that may not be stable in other areas of the molecule, not just in terms of the loss of substituents. Overall, these top scoring molecules from the two GAs of this project show potential as redox-active materials for use in non-aqueous RFBs due to their computed values of cell potential, stability, and solvation free energy.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

RFBs have been proposed as a solution for the ongoing energy storage problem that has developed from the increasing demand for energy and the resources needed to provide this energy.[1,2] By pairing a large scale energy storage system like RFBs with renewable energy sources, the energy generated can be stored in excess quantities to be used on demand.[7,8] RFBs are highly favourable as large scale energy storage systems because of their flexible design, allowing their size to be scaled to that needed for specific applications. They also show advantages over other energy storage systems for reasons such as their ability to effectively combine energy efficiency, capital costs, and life cycle costs compared to conventional energy storage systems.[20]

RFBs can be categorized as either inorganic or organic where the most commonly researched RFB is the vanadium inorganic RFB.[21] Organic RFBs can be further categorized as either aqueous or non-aqueous depending on the type of solvent being used in the battery system.[11] In this project, organic molecules for use in non-aqueous RFBs were studied through the automated random generation of molecules,

followed by computational calculations to determine various redox properties of these molecules. ML methods were used to automate various processes of this project as well as through the application of a GA to further improve the generated molecules. GAs work through the use of selection, crossover, and mutation methods where in this project, selection was employed through the use of rank selection to determine the population for the GA by only choosing high scoring individuals for the population and also through stochastic universal sampling to randomly choose two parent molecules from the population to be paired together for the crossover event. Uniform crossover was applied to the two parent molecules to randomly swap traits of these two molecules and generate a new offspring molecule with features from both parents. Lastly, mutations were applied in a way that would benefit the molecule if valency issues arose during the crossover step.

A set of 360 molecules were randomly generated initially and ranked based on their calculated cell potential, stability, and solvation free energy values to generate two GA populations. The first GA population compared the calculated values to those of a reference BPY molecule while the second GA compared values to the average of the randomly generated population to generate two different scoring functions for ranking the molecules. After multiple generations of each GA, improvements were seen in both populations as shown through the changes in distributions presented in Chapter 5.8.4. Over the course of the GAs, the populations shifted to higher score values, showing that the molecules generated through the GAs were improving in their E_{cell}, $\Delta G_{stability}$, and $\Delta G_{solvation}$ values and therefore in their abilities as redox-active materials for use in non-aqueous RFBs. Two sets of potential active material candidates were provided in Chapter 5.9 with varying structural characteristics, but all appear to be good candidates based on their calculated redox properties.

6.2 Future Work

While some active material candidates were identified, more work should be done to verify their potential as active materials. Many of the computations performed on the generated molecules were done in a way to minimize the computational time required for each molecule,therefore, more exhaustive and higher level calculations are needed to verify the approximations made in this project. This would include performing DFT geometry optimization calculations for top scoring molecules as well as more quantitative measures of stability. As mentioned, there are multiple ways that the stability of these molecules can be determined. In this project, that was done by computing the ΔG for the loss of a substituent as this was able to be easily implemented into the automation of these molecules without adding much additional computational time. While this method gave a qualitative measure of stability, it did have the setback of having to use an estimation of the fully saturated molecule ($\mathrm{FGRRH} + \mathrm{HFG}$) optimized by RDKit[74] therefore ΔG for the direct loss of a substituent ($\mathrm{FGRR}^{-} + \mathrm{FG}^{+}$) should also be calculated from DFT geometry optimization for top scoring molecules as a more quantitative measure of stability for the loss of a substituent. Other methods of investigating stability may take more time and computational resources to compute the stability of the generated molecules however looking into methods of automating the stability such as through ring breaking and reactive points could be more insightful. These methods combined with investigating the stability for the loss of a substituent would help to measure the overall stability of the generated molecules. For the calculations of stability and solvation free energy, only the values for the neutral state of each molecule were determined. Therefore, for the top scoring molecules, these values should also be calculated for the dication and dianion states as well to ensure that the top scoring molecules are also stable and soluble in these states, not just in the neutral state. It is important to computationally verify the top scoring molecules with more comprehensive calcula-

tions so that they may then be verified experimentally. To accompany experimental verification, methods of synthetic accessibility could be explored computationally to assess how easily the generated molecules may be synthesized.[84] While a set of active material candidates were identified in this project, it is clear that more methods of verifying the redox capabilities and stability of these molecules is necessary before experimental verification.

References

Advanced computing resources are provided by the Digital Research Alliance of Canada (Alliance), the organization responsible for digital research infrastructure in Canada, and ACENET, the regional partner in Atlantic Canada. ACENET is funded by the Alliance, the provinces of New Brunswick, Newfoundland & Labrador, Nova Scotia and Prince Edward Island, as well as the Atlantic Canada Opportunities Agency.

[1] M. Hoeven, World Energy Outlook 2011, International Energy Agency, **2015**.

[2] E. I. A. (US), *Annual Energy Outlook 2012, with Projections To 2035*, Government Printing Office, **2012**.

[3] P. Alotto, M. Guarnieri, F. Moro, "Redox flow batteries for the storage of renewable energy: A review", *Renewable and sustainable energy reviews* **2014**, *29*, 325–335.

[4] J. Kondoh, I. Ishii, H. Yamaguchi, A. Murata, K. Otani, K. Sakuta, N. Higuchi, S. Sekine, M. Kamimoto, "Electrical energy storage systems for energy networks", *Energy conversion and management* **2000**, *41*, 1863–1874.

[5] B. P. Roberts, C. Sandberg, "The role of energy storage in development of smart grids", *Proceedings of the IEEE* **2011**, *99*, 1139–1144.

[6] T. Hennessy in Annual Peer Review, **2006**.

[7] T. Shigematsu et al., "Redox flow battery for energy storage", *SEI technical review* **2011**, *73*, 13.

[8] C. P. De Leon, A. Frías-Ferrer, J. González-García, D. Szánto, F. C. Walsh, "Redox flow cells for energy conversion", *Journal of power sources* **2006**, *160*, 716–732.

[9] J. Liu, J.-G. Zhang, Z. Yang, J. P. Lemmon, C. Imhoff, G. L. Graff, L. Li, J. Hu, C. Wang, J. Xiao, et al., "Materials science and materials chemistry for large scale electrochemical energy storage: from transportation to electrical grid", *Advanced Functional Materials* **2013**, *23*, 929–946.

[10] F. Rahman, M. Skyllas-Kazacos, "Vanadium redox battery: Positive half-cell electrolyte studies", *Journal of Power Sources* **2009**, *189*, 1212–1219.

[11] W. Wang, Q. Luo, B. Li, X. Wei, L. Li, Z. Yang, "Recent progress in redox flow battery research and development", *Advanced Functional Materials* **2013**, *23*, 970–986.

[12] W. Wang, L. Li, Z. Nie, B. Chen, Q. Luo, Y. Shao, X. Wei, F. Chen, G.-G. Xia, Z. Yang, "A new hybrid redox flow battery with multiple redox couples", *Journal of Power Sources* **2012**, *216*, 99–103.

[13] M. Al Raihan, C. A. Dyker, "Ester-Substituted Bispyridinylidenes: Double Concerted Two-Electron Bipolar Molecules for Symmetric Organic Redox Flow Batteries", *ACS Energy Letters* **2023**, *8*, 3314–3322.

[14] M. Mohammed, M. B. Khan, E. B. M. Bashier, *Machine learning: algorithms and applications*, Crc Press, **2016**.

[15] L. P. Kaelbling, M. L. Littman, A. W. Moore, "Reinforcement learning: A survey", *Journal of artificial intelligence research* **1996**, *4*, 237–285.

[16] I. Gridin, *Learning Genetic Algorithms with Python: Empower the Performance of Machine Learning and AI Models with the Capabilities of a Powerful Search Algorithm*, BPB Publications, **2021**.

[17] E. J. Coster, J. M. Myrzik, B. Kruimer, W. L. Kling, "Integration issues of distributed generation in distribution grids", *Proceedings of the IEEE* **2010**, *99*, 28–39.

[18] M. Świerczyński, R. Teodorescu, C. N. Rasmussen, P. Rodriguez, H. Vikelgaard in 2010 IEEE international symposium on industrial electronics, IEEE, **2010**, pp. 3749–3756.

[19] S. Vazquez, S. Lukic, E. Galvan, L. G. Franquelo, J. M. Carrasco, J. I. Leon in IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society, **2011**, pp. 4636–4640.

[20] A. Z. Weber, M. M. Mench, J. P. Meyers, P. N. Ross, J. T. Gostick, Q. Liu, "Redox flow batteries: a review", *Journal of applied electrochemistry* **2011**, *41*, 1137–1164.

[21] Y. V. Tolmachev, "Flow batteries from 1879 to 2022 and beyond", *Journal of The Electrochemical Society* **2023**, *170*, 030505.

[22] A. Parasuraman, T. M. Lim, C. Menictas, M. Skyllas-Kazacos, "Review of material research and development for vanadium redox flow battery applications", *Electrochimica Acta* **2013**, *101*, 27–40.

[23] X.-P. Gao, H.-X. Yang, "Multi-electron reaction materials for high energy density batteries", *Energy & Environmental Science* **2010**, *3*, 174–189.

[24] P. Leung, X. Li, C. P. De León, L. Berlouis, C. J. Low, F. C. Walsh, "Progress in redox flow batteries, remaining challenges and their applications in energy storage", *Rsc Advances* **2012**, *2*, 10125–10156.

[25] J. Noack, N. Roznyatovskaya, T. Herr, P. Fischer, "The chemistry of redox-flow batteries", *Angewandte Chemie International Edition* **2015**, *54*, 9776–9809.

[26] Y. Zhao, Y. Ding, Y. Li, L. Peng, H. R. Byon, J. B. Goodenough, G. Yu, "A chemistry and material perspective on lithium redox flow batteries towards high-density electrical energy storage", *Chemical Society Reviews* **2015**, *44*, 7968–7996.

[27] M. Skyllas-Kazacos, M. Rychick, R. Robins, All-vanadium redox battery, US Patent 4,786,567, **1988**.

[28] M. Rychcik, M. Skyllas-Kazacos, "Evaluation of electrode materials for vanadium redox cell", *Journal of Power Sources* **1987**, *19*, 45–54.

[29] H. Zhang, W. Lu, X. Li, "Progress and perspectives of flow battery technologies", *Electrochemical Energy Reviews* **2019**, *2*, 492–506.

[30] H. Chen, G. Cong, Y.-C. Lu, "Recent progress in organic redox flow batteries: Active materials, electrolytes and membranes", *Journal of energy chemistry* **2018**, *27*, 1304–1325.

[31] J. Winsberg, T. Hagemann, T. Janoschka, M. D. Hager, U. S. Schubert, "Redox-flow batteries: from metals to organic redox-active materials", *Angewandte Chemie International Edition* **2017**, *56*, 686–711.

[32] P. Symons, "Quinones for redox flow batteries", *Current Opinion in Electrochemistry* **2021**, *29*, 100759.

[33] M. Sánchez-Castellanos, M. M. Flores-Leonar, Z. Mata-Pinzón, H. G. Laguna, K. M. García-Ruiz, S. S. Rozenel, V. M. Ugalde-Saldívar, R. Moreno-Esparza, J. J. Pijpers, C. Amador-Bedolla, "Theoretical exploration of 2, 2'-bipyridines as electro-active compounds in flow batteries", *Physical Chemistry Chemical Physics* **2019**, *21*, 15823–15832.

[34] L. Cheng, R. S. Assary, X. Qu, A. Jain, S. P. Ong, N. N. Rajput, K. Persson, L. A. Curtiss, "Accelerating electrolyte discovery for energy storage with high-throughput screening", *The journal of physical chemistry letters* **2015**, *6*, 283–291.

[35] S. D. Pineda Flores, G. C. Martin-Noble, R. L. Phillips, J. Schrier, "Bio-inspired Electroactive organic molecules for aqueous redox flow batteries. 1. Thiophenoquinones", *The Journal of Physical Chemistry C* **2015**, *119*, 21800–21809.

[36] B. Huskinson, M. P. Marshak, C. Suh, S. Er, M. R. Gerhardt, C. J. Galvin, X. Chen, A. Aspuru-Guzik, R. G. Gordon, M. J. Aziz, "A metal-free organic–inorganic aqueous flow battery", *Nature* **2014**, *505*, 195–198.

[37] J. E. Bachman, L. A. Curtiss, R. S. Assary, "Investigation of the redox chemistry of anthraquinone derivatives using density functional theory", *The Journal of Physical Chemistry A* **2014**, *118*, 8852–8860.

[38] Z. Yang, L. Tong, D. P. Tabor, E. S. Beh, M.-A. Goulet, D. De Porcellinis, A. Aspuru-Guzik, R. G. Gordon, M. J. Aziz, "Alkaline benzoquinone aqueous flow battery for large-scale storage of electrical energy", *Advanced Energy Materials* **2018**, *8*, 1702056.

[39] C. DeBruler, B. Hu, J. Moss, X. Liu, J. Luo, Y. Sun, T. L. Liu, "Designer two-electron storage viologen anolyte materials for neutral aqueous organic redox flow batteries", *Chem* **2017**, *3*, 961–978.

[40] M. Krejčík, A. Viček, "Electrochemical formation of dianions of 2, 2'-bipyridine and related compounds", *Journal of electroanalytical chemistry and interfacial electrochemistry* **1991**, *313*, 243–257.

[41] H. Erhard, W. Jaenicke, "Kinetics and mechanism of the electrochemical reduction of 2, 2'-bipyridine: Part III. Accurate kinetics and thermodynamics of

the base catalyzed consecutive reactions", *Journal of Electroanalytical Chemistry and Interfacial Electrochemistry* **1977**, *81*, 89–96.

[42] B. Mennucci, "Polarizable continuum model", *Wiley Interdisciplinary Reviews: Computational Molecular Science* **2012**, *2*, 386–404.

[43] A. V. Marenich, C. J. Cramer, D. G. Truhlar, "Universal solvation model based on solute electron density and on a continuum model of the solvent defined by the bulk dielectric constant and atomic surface tensions", *The Journal of Physical Chemistry B* **2009**, *113*, 6378–6396.

[44] J. Yuan, Z.-Z. Pan, Y. Jin, Q. Qiu, C. Zhang, Y. Zhao, Y. Li, "Membranes in non-aqueous redox flow battery: A review", *Journal of Power Sources* **2021**, *500*, 229983.

[45] M. L. Perry, J. D. Saraidaridis, R. M. Darling, "Crossover mitigation strategies for redox-flow batteries", *Current Opinion in Electrochemistry* **2020**, *21*, 311–318.

[46] Z. Liang, N. H. Attanayake, K. V. Greco, B. J. Neyhouse, J. L. Barton, A. P. Kaur, W. L. Eubanks, F. R. Brushett, J. Landon, S. A. Odom, "Comparison of separators vs membranes in nonaqueous redox flow battery electrolytes containing small molecule active materials", *ACS Applied Energy Materials* **2021**, *4*, 5443–5451.

[47] R. A. Potash, J. R. McKone, S. Conte, H. D. Abruna, "On the benefits of a symmetric redox flow battery", *Journal of The Electrochemical Society* **2016**, *163*, A338.

[48] W. Duan, R. S. Vemuri, J. D. Milshtein, S. Laramie, R. D. Dmello, J. Huang, L. Zhang, D. Hu, M. Vijayakumar, W. Wang, et al., "A symmetric organic-based nonaqueous redox flow battery and its state of charge diagnostics by FTIR", *Journal of Materials Chemistry A* **2016**, *4*, 5448–5456.

[49] D. Zhang, E. J. Dufek, E. L. Clennan, "Syntheses, Characterizations, and Properties of Electronically Perturbed 1, 1 '-Dimethyl-2, 2 '-bipyridinium Tetrafluoroborates", *The Journal of Organic Chemistry* **2006**, *71*, 315–319.

[50] N. A. Richard, C. K. Khor, S. M. Hetherington, S. L. Mills, A. Decken, C. A. Dyker, "Iminophosphorano-Substituted Bispyridinylidenes: Redox Potentials and Substituent Constants from Tolman Electronic Parameters", *Chemistry–A European Journal* **2020**, *26*, 17371–17375.

[51] S. S. Hanson, N. A. Richard, C. A. Dyker, "Powerful Bispyridinylidene Organic Reducing Agents with Iminophosphorano π-Donor Substituents", *Chemistry–A European Journal* **2015**, *21*, 8052–8055.

[52] D. Deng, M. G. Kim, J. Y. Lee, J. Cho, "Green energy storage materials: Nanostructured TiO 2 and Sn-based anodes for lithium-ion batteries", *Energy & Environmental Science* **2009**, *2*, 818–837.

[53] M. S. Whittingham, "Electrical energy storage and intercalation chemistry", *Science* **1976**, *192*, 1126–1127.

[54] X. Wang, Z. Wen, Y. Liu, X. Wu, "A novel composite containing nanosized silicon and tin as anode material for lithium ion batteries", *Electrochimica Acta* **2009**, *54*, 4662–4667.

[55] D. Deng, "Li-ion batteries: basics, progress, and challenges", *Energy Science & Engineering* **2015**, *3*, 385–418.

[56] L. Carrette, K. A. Friedrich, U. Stimming, "Fuel cells: principles, types, fuels, and applications", *ChemPhysChem* **2000**, *1*, 162–193.

[57] B. C. Steele, A. Heinzel, "Materials for fuel-cell technologies", *Nature* **2001**, *414*, 345–352.

[58] A. Bauen, D. Hart, "Assessment of the environmental benefits of transport and stationary fuel cells", *Journal of power sources* **2000**, *86*, 482–494.

[59] M. Aneke, M. Wang, "Energy storage technologies and real life applications–A state of the art review", *Applied Energy* **2016**, *179*, 350–377.

[60] W. F. Pickard, "The history, present state, and future prospects of underground pumped hydro for massive energy storage", *Proceedings of the IEEE* **2011**, *100*, 473–483.

[61] C. Blair, Gravitricity–Storing Power as well as Energy, **2016**.

[62] J. Liu, G. Cao, Z. Yang, D. Wang, D. Dubois, X. Zhou, G. L. Graff, L. R. Pederson, J.-G. Zhang, "Oriented nanostructures for energy conversion and storage", *ChemSusChem: Chemistry & Sustainability Energy & Materials* **2008**, *1*, 676–697.

[63] P. A. Taylor, J. L. Miller, P. C. Butler, Energy Storage Opportunities Analysis Phase II Final Report: A Study for the DOE Energy Storage Systems Program, tech. rep., Sandia National Lab.(SNL-NM), Albuquerque, NM (United States); Sandia ..., **2002**.

[64] Z. Yang, J. Zhang, M. C. Kintner-Meyer, X. Lu, D. Choi, J. P. Lemmon, J. Liu, "Electrochemical energy storage for green grid", *Chemical reviews* **2011**, *111*, 3577–3613.

[65] I. H. Sarker, A. Kayes, P. Watters, "Effectiveness analysis of machine learning classification models for predicting personalized context-aware smartphone usage", *Journal of Big Data* **2019**, *6*, 1–28.

[66] J. Han, M. Kamber, J. Pei, *Data Mining: Concepts and Techniques*, Elsevier, Amsterdam, **2011**.

[67] I. H. Sarker, A. Kayes, S. Badsha, H. Alqahtani, P. Watters, A. Ng, "Cybersecurity data science: an overview from machine learning perspective", *Journal of Big data* **2020**, *7*, 1–29.

[68] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions", *SN computer science* **2021**, *2*, 160.

[69] F. Zheng, Z. Zhu, J. Lu, Y. Yan, H. Jiang, Q. Sun, "Predicting the HOMO-LUMO gap of benzenoid polycyclic hydrocarbons via interpretable machine learning", *Chemical Physics Letters* **2023**, *814*, 140358.

[70] F. Wang, J. Li, Z. Liu, T. Qiu, J. Wu, D. Lu, "Computational design of quinone electrolytes for redox flow batteries using high-throughput machine learning and theoretical calculations", *Frontiers in Chemical Engineering* **2023**, *4*, 1086412.

[71] Q. Chen, M. R. Gerhardt, L. Hartle, M. J. Aziz, "A quinone-bromide flow battery with 1 W/cm2 power density", *Journal of the Electrochemical Society* **2015**, *163*, A5010.

[72] Z. Xiong, D. Wang, X. Liu, F. Zhong, X. Wan, X. Li, Z. Li, X. Luo, K. Chen, H. Jiang, et al., "Pushing the boundaries of molecular representation for drug discovery with the graph attention mechanism", *Journal of medicinal chemistry* **2019**, *63*, 8749–8760.

[73] T. A. Halgren, "Merck molecular force field. III. Molecular geometries and vibrational frequencies for MMFF94", *Journal of computational chemistry* **1996**, *17*, 553–586.

[74] RDKit: Open-Source Cheminformatics.

[75] G. M. Barca, C. Bertoni, L. Carrington, D. Datta, N. De Silva, J. E. Deustua, D. G. Fedorov, J. R. Gour, A. O. Gunina, E. Guidez, et al., "Recent developments in the general atomic and molecular electronic structure system", *The Journal of chemical physics* **2020**, *152*.

[76] N. M. O'Boyle, M. Banck, C. A. James, C. Morley, T. Vandermeersch, G. R. Hutchison, "Open Babel: An open chemical toolbox", *Journal of cheminformatics* **2011**, *3*, 1–14.

[77] The Open Babel Package.

[78] F. Jensen, *Introduction to computational chemistry*, John wiley & sons, **2017**.

[79] N. Mardirossian, M. Head-Gordon, "Thirty years of density functional theory in computational chemistry: an overview and extensive assessment of 200 density functionals", *Molecular physics* **2017**, *115*, 2315–2372.

[80] T. H. Dunning Jr, "Gaussian basis sets for use in correlated molecular calculations. I. The atoms boron through neon and hydrogen", *The Journal of chemical physics* **1989**, *90*, 1007–1023.

[81] N. Elgrishi, K. J. Rountree, B. D. McCarthy, E. S. Rountree, T. T. Eisenhart, J. L. Dempsey, "A practical beginner's guide to cyclic voltammetry", *Journal of chemical education* **2018**, *95*, 197–206.

[82] W. Ogilvie, N. Ackroyd, C. S. Browning, G. Deslongchamps, E. Sauer, *Organic Chemistry: Mechanistic Patterns*, Nelson, **2018**.

[83] D. E. Manolopoulos, J. C. May, S. E. Down, "Theoretical studies of the fullerenes: C34 to C70", *Chemical Physics Letters* **1991**, *181*, 105–111.

[84] P. Ertl, A. Schuffenhauer, "Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions", *Journal of cheminformatics* **2009**, *1*, 1–11.

Appendix A

Python Script for the Random Generation of Organic Molecules

```
import random
import os
import fileinput

import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Chem import rdmolfiles

from autocombinatorial import build

initial_mol_filepath = './buildingblocks/startingmolfile.mol'
max_number_n = 4
max_number_fg = 2
max_number_db = 3
double = True

for i in range(1):
    print('molecule number?')
```

```
    molecule_number = input()
    molecule_number = str(molecule_number)
    molecule_name = '%s-' %(molecule_number)
    molecule = build.autocombinatorial(initial_mol_filepath=
   initial_mol_filepath,max_number_n=max_number_n,max_number_fg=
   max_number_fg,max_number_db=max_number_db,double=double,
   molecule_name=molecule_name)
    molecule.add_n()
    molecule.add_fg()
    molecule.add_db()
    molecule.symmetrize()
    molecule.linker()
    molecule.show()
    molecule.remove()

import random
import os
import fileinput

import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Chem import rdmolfiles

print('molecule number?')
molecule_number = input()
molecule_number = str(molecule_number)

optimized_filename = './molecules/optimized/%s-final-optimized.mol'
   %(molecule_number)

for file in os.listdir('./molecules/mol-files/'):
    if file.startswith('%s-' %(molecule_number)):
```

```
        output_file = './molecules/mol-files/' + file

        mol = Chem.MolFromMolFile(output_file)
        mol = Chem.AddHs(mol)
        molecule=Chem.MolToMolBlock(mol)
        AllChem.EmbedMolecule(mol)
        AllChem.MMFFOptimizeMolecule(mol,maxIters=5000)
        Chem.rdmolfiles.MolToMolFile(mol,optimized_filename)

converted_filename = './molecules/converted/%s-final-converted.inp'
   %(molecule_number)

os.system('obabel ' + optimized_filename + ' -O ' +
   converted_filename)

neutgas_filename = './molecules/gamess-files/%s-neutgas.inp' %(
   molecule_number)
neutdmf_filename = './molecules/gamess-files/%s-neutdmf.inp' %(
   molecule_number)
dicatdmf_filename = './molecules/gamess-files/%s-dicatdmf.inp' %(
   molecule_number)
diandmf_filename = './molecules/gamess-files/%s-diandmf.inp' %(
   molecule_number)

file = open(neutgas_filename,mode='w',encoding='utf-8')

for line in fileinput.FileInput(converted_filename, inplace=0):
    if '$CONTRL' in line:
        line = line.replace(line, ' $CONTRL SCFTYP=RHF RUNTYP=
   HESSIAN EXETYP=RUN DFTTYP=B3LYP' + '\n' 'MAXIT=120 ICHARG=0 MULT
   =1 ISPHER=1 $END' + '\n')
        file.write(line)
        file.write(' $BASIS GBASIS=CCD $END' +'\n')
```

```
        file.write(' $SYSTEM TIMLIM=1300 MWORDS=500 PARALL=.TRUE.
    $END'+'\n')
        file.write(' $SCF DIRSCF=.TRUE. $END '+'\n')
        file.write(' $STATPT HESS=READ $END'+'\n')
for line in fileinput.FileInput(converted_filename, inplace=0):
    if ' $CONTRL COORD=CART' not in line:
        file.write(line)
file.close()

file = open(neutdmf_filename,mode='w',encoding='utf-8')

for line in fileinput.FileInput(converted_filename, inplace=0):
    if '$CONTRL' in line:
        line = line.replace(line, ' $CONTRL SCFTYP=RHF RUNTYP=
    HESSIAN EXETYP=RUN DFTTYP=B3LYP' + '\n' 'MAXIT=120 ICHARG=0 MULT
    =1 ISPHER=1 $END' + '\n' ' $PCM SOLVNT=DMF SMD=.TRUE. $END' + '\
    n' ' $FORCE METHOD=SEMINUM $END' + '\n')
        file.write(line)
        file.write(' $BASIS GBASIS=CCD $END' +'\n')
        file.write(' $SYSTEM TIMLIM=1300 MWORDS=500 PARALL=.TRUE.
    $END'+'\n')
        file.write(' $SCF DIRSCF=.TRUE. $END '+'\n')
        file.write(' $STATPT HESS=READ $END'+'\n')
for line in fileinput.FileInput(converted_filename, inplace=0):
    if ' $CONTRL COORD=CART' not in line:
        file.write(line)
file.close()

file = open(dicatdmf_filename,mode='w',encoding='utf-8')

for line in fileinput.FileInput(converted_filename, inplace=0):
    if '$CONTRL' in line:
```

```
        line = line.replace(line, ' $CONTRL SCFTYP=RHF RUNTYP=
    HESSIAN EXETYP=RUN DFTTYP=B3LYP' + '\n' 'MAXIT=120 ICHARG=2 MULT
    =1 ISPHER=1 $END' + '\n' ' $PCM SOLVNT=DMF SMD=.TRUE. $END' + '\
    n' ' $FORCE METHOD=SEMINUM $END' + '\n')
        file.write(line)
        file.write(' $BASIS GBASIS=CCD $END' +'\n')
        file.write(' $SYSTEM TIMLIM=1300 MWORDS=500 PARALL=.TRUE.
    $END'+'\n')
        file.write(' $SCF DIRSCF=.TRUE. $END '+'\n')
        file.write(' $STATPT HESS=READ $END'+'\n')
for line in fileinput.FileInput(converted_filename, inplace=0):
    if ' $CONTRL COORD=CART' not in line:
        file.write(line)
file.close()

file = open(diandmf_filename,mode='w',encoding='utf-8')

for line in fileinput.FileInput(converted_filename, inplace=0):
    if '$CONTRL' in line:
        line = line.replace(line, ' $CONTRL SCFTYP=RHF RUNTYP=
    HESSIAN EXETYP=RUN DFTTYP=B3LYP' + '\n' 'MAXIT=120 ICHARG=-2
    MULT=1 ISPHER=1 $END' + '\n' ' $PCM SOLVNT=DMF SMD=.TRUE. $END'
    + '\n' ' $FORCE METHOD=SEMINUM $END' + '\n')
        file.write(line)
        file.write(' $BASIS GBASIS=CCD $END' +'\n')
        file.write(' $SYSTEM TIMLIM=1300 MWORDS=500 PARALL=.TRUE.
    $END'+'\n')
        file.write(' $SCF DIRSCF=.TRUE. $END '+'\n')
        file.write(' $STATPT HESS=READ $END'+'\n')
for line in fileinput.FileInput(converted_filename, inplace=0):
    if ' $CONTRL COORD=CART' not in line:
        file.write(line)
file.close()
```

```
os.system('rm *converted*')

for file_a in os.listdir('./molecules/removed/mol-files-removed'):
    if file_a.endswith('-REM0.mol'):
        if file_a.startswith('%s-' %(molecule_number)):
            output_a_file = './molecules/removed/mol-files-removed/'
    + file_a
            print(output_a_file)
            optimized_a_filename = './molecules/removed/optimized-
   removed/%s-a-removed-optimized.mol' %(molecule_number)

            mol = Chem.MolFromMolFile(output_a_file)
            mol = Chem.AddHs(mol)
            molecule=Chem.MolToMolBlock(mol)
            AllChem.EmbedMolecule(mol)
            AllChem.MMFFOptimizeMolecule(mol,maxIters=5000)
            Chem.rdmolfiles.MolToMolFile(mol,optimized_a_filename)

            converted_a_filename = './molecules/removed/converted-
   removed/%s-a-removed-converted.inp' %(molecule_number)

            os.system('obabel ' + optimized_a_filename + ' -O ' +
   converted_a_filename)

            neutdmf_a_filename = './molecules/removed/gamess-files-
   removed/%s-a-neutdmf.inp' %(molecule_number)
            dicatdmf_a_filename = './molecules/removed/gamess-files-
   removed/%s-a-dicatdmf.inp' %(molecule_number)
            diandmf_a_filename = './molecules/removed/gamess-files-
   removed/%s-a-diandmf.inp' %(molecule_number)
```

```
        file = open(neutdmf_a_filename,mode='w',encoding='utf-8'
)

        for line in fileinput.FileInput(converted_a_filename,
inplace=0):
            if '$CONTRL' in line:
                line = line.replace(line, ' $CONTRL SCFTYP=RHF
RUNTYP=HESSIAN EXETYP=RUN DFTTYP=B3LYP' + '\n' 'MAXIT=120 ICHARG
=0 MULT=1 ISPHER=1 $END' + '\n' ' $PCM SOLVNT=DMF SMD=.TRUE.
$END' + '\n' ' $FORCE METHOD=SEMINUM $END' + '\n')
                file.write(line)
                file.write(' $BASIS GBASIS=CCD $END' +'\n')
                file.write(' $SYSTEM TIMLIM=1300 MWORDS=500
PARALL=.TRUE. $END'+'\n')
                file.write(' $SCF DIRSCF=.TRUE. $END '+'\n')
                file.write(' $STATPT HESS=READ $END'+'\n')
        for line in fileinput.FileInput(converted_a_filename,
inplace=0):
            if ' $CONTRL COORD=CART' not in line:
                file.write(line)
        file.close()

        file = open(dicatdmf_a_filename,mode='w',encoding='utf-8
')

        for line in fileinput.FileInput(converted_a_filename,
inplace=0):
            if '$CONTRL' in line:
                line = line.replace(line, ' $CONTRL SCFTYP=RHF
RUNTYP=HESSIAN EXETYP=RUN DFTTYP=B3LYP' + '\n' 'MAXIT=120 ICHARG
=2 MULT=1 ISPHER=1 $END' + '\n' ' $PCM SOLVNT=DMF SMD=.TRUE.
$END' + '\n' ' $FORCE METHOD=SEMINUM $END' + '\n')
                file.write(line)
```

```
                file.write(' $BASIS GBASIS=CCD $END' +'\n')
                file.write(' $SYSTEM TIMLIM=1300 MWORDS=500
PARALL=.TRUE. $END'+'\n')
                file.write(' $SCF DIRSCF=.TRUE. $END '+'\n')
                file.write(' $STATPT HESS=READ $END'+'\n')
        for line in fileinput.FileInput(converted_a_filename,
inplace=0):
            if ' $CONTRL COORD=CART' not in line:
                file.write(line)
        file.close()

        file = open(diandmf_a_filename,mode='w',encoding='utf-8'
)

        for line in fileinput.FileInput(converted_a_filename,
inplace=0):
            if '$CONTRL' in line:
                line = line.replace(line, ' $CONTRL SCFTYP=RHF
RUNTYP=HESSIAN EXETYP=RUN DFTTYP=B3LYP' + '\n' 'MAXIT=120 ICHARG
=-2 MULT=1 ISPHER=1 $END' + '\n' ' $PCM SOLVNT=DMF SMD=.TRUE.
$END' + '\n' ' $FORCE METHOD=SEMINUM $END' + '\n')
                file.write(line)
                file.write(' $BASIS GBASIS=CCD $END' +'\n')
                file.write(' $SYSTEM TIMLIM=1300 MWORDS=500
PARALL=.TRUE. $END'+'\n')
                file.write(' $SCF DIRSCF=.TRUE. $END '+'\n')
                file.write(' $STATPT HESS=READ $END'+'\n')
        for line in fileinput.FileInput(converted_a_filename,
inplace=0):
            if ' $CONTRL COORD=CART' not in line:
                file.write(line)
        file.close()
```

```
for file_b in os.listdir('./molecules/removed/mol-files-removed/'):
    if file_b.endswith('-REM1.mol'):
        if file_b.startswith('%s-' %(molecule_number)):
            output_b_file = './molecules/removed/mol-files-removed/'
    + file_b
            print(output_b_file)
            optimized_b_filename = './molecules/removed/optimized-
   removed/%s-b-removed-optimized.mol' %(molecule_number)

            mol = Chem.MolFromMolFile(output_b_file)
            mol = Chem.AddHs(mol)
            molecule=Chem.MolToMolBlock(mol)
            AllChem.EmbedMolecule(mol)
            AllChem.MMFFOptimizeMolecule(mol,maxIters=5000)
            Chem.rdmolfiles.MolToMolFile(mol,optimized_b_filename)

            converted_b_filename = './molecules/removed/converted-
   removed/%s-b-removed-converted.inp' %(molecule_number)

            os.system('obabel ' + optimized_b_filename + ' -O ' +
   converted_b_filename)

            neutdmf_b_filename = './molecules/removed/gamess-files-
   removed/%s-b-neutdmf.inp' %(molecule_number)
            dicatdmf_b_filename = './molecules/removed/gamess-files-
   removed/%s-b-dicatdmf.inp' %(molecule_number)
            diandmf_b_filename = './molecules/removed/gamess-files-
   removed/%s-b-diandmf.inp' %(molecule_number)

            file = open(neutdmf_b_filename,mode='w',encoding='utf-8'
   )
```

```
        for line in fileinput.FileInput(converted_b_filename,
inplace=0):
            if '$CONTRL' in line:
                line = line.replace(line, ' $CONTRL SCFTYP=RHF
RUNTYP=HESSIAN EXETYP=RUN DFTTYP=B3LYP' + '\n' 'MAXIT=120 ICHARG
=0 MULT=1 ISPHER=1 $END' + '\n' ' $PCM SOLVNT=DMF SMD=.TRUE.
$END' + '\n' ' $FORCE METHOD=SEMINUM $END' + '\n')
                file.write(line)
                file.write(' $BASIS GBASIS=CCD $END' +'\n')
                file.write(' $SYSTEM TIMLIM=1300 MWORDS=500
PARALL=.TRUE. $END'+'\n')
                file.write(' $SCF DIRSCF=.TRUE. $END '+'\n')
                file.write(' $STATPT HESS=READ $END'+'\n')
        for line in fileinput.FileInput(converted_b_filename,
inplace=0):
            if ' $CONTRL COORD=CART' not in line:
                file.write(line)
        file.close()

        file = open(dicatdmf_b_filename,mode='w',encoding='utf-8
')

        for line in fileinput.FileInput(converted_b_filename,
inplace=0):
            if '$CONTRL' in line:
                line = line.replace(line, ' $CONTRL SCFTYP=RHF
RUNTYP=HESSIAN EXETYP=RUN DFTTYP=B3LYP' + '\n' 'MAXIT=120 ICHARG
=2 MULT=1 ISPHER=1 $END' + '\n' ' $PCM SOLVNT=DMF SMD=.TRUE.
$END' + '\n' ' $FORCE METHOD=SEMINUM $END' + '\n')
                file.write(line)
                file.write(' $BASIS GBASIS=CCD $END' +'\n')
                file.write(' $SYSTEM TIMLIM=1300 MWORDS=500
PARALL=.TRUE. $END'+'\n')
```

```
                file.write(' $SCF DIRSCF=.TRUE. $END '+'\n')
                file.write(' $STATPT HESS=READ $END'+'\n')
        for line in fileinput.FileInput(converted_b_filename,
inplace=0):
            if ' $CONTRL COORD=CART' not in line:
                file.write(line)
        file.close()

        file = open(diandmf_b_filename,mode='w',encoding='utf-8'
)

        for line in fileinput.FileInput(converted_b_filename,
inplace=0):
            if '$CONTRL' in line:
                line = line.replace(line, ' $CONTRL SCFTYP=RHF
RUNTYP=HESSIAN EXETYP=RUN DFTTYP=B3LYP' + '\n' 'MAXIT=120 ICHARG
=-2 MULT=1 ISPHER=1 $END' + '\n' ' $PCM SOLVNT=DMF SMD=.TRUE.
$END' + '\n' ' $FORCE METHOD=SEMINUM $END' + '\n')
                file.write(line)
                file.write(' $BASIS GBASIS=CCD $END' +'\n')
                file.write(' $SYSTEM TIMLIM=1300 MWORDS=500
PARALL=.TRUE. $END'+'\n')
                file.write(' $SCF DIRSCF=.TRUE. $END '+'\n')
                file.write(' $STATPT HESS=READ $END'+'\n')
        for line in fileinput.FileInput(converted_b_filename,
inplace=0):
            if ' $CONTRL COORD=CART' not in line:
                file.write(line)
        file.close()

    else:
        pass
```

A.1 build.py

```
from calendar import c
import random
import fileinput
from re import A
import numpy as np
import os

from .utils import utils

class autocombinatorial():
    def __init__(self,initial_mol_filepath='',mol_string='',
   max_number_n=6,max_number_fg=6,max_number_db=6,double=False,
   molecule_name=''):

        for line in fileinput.FileInput(initial_mol_filepath,inplace
   =0):
            mol_string = mol_string + line

        if double == True:
            double = random.choice((True,False,False,False,False,
   False))

        self.mol_string = mol_string
        self.max_number_n = max_number_n
        self.max_number_fg = max_number_fg
        self.max_number_db = max_number_db
        self.double = double
        self.molecule_name = molecule_name

    def show(self):
        print('max nitrogens:', self.max_number_n)
```

```
        print('double linker', self.double)
        print('molecule file:', self.mol_string)
        print('N positions', self.n_positions)
        print('fg positions', self.fgs)
        print('db position', self.db_positions)

        self.fgs = np.array(self.fgs)
        self.n_positions = np.sort(self.n_positions)
        self.db_positions = np.sort(self.db_positions)

        if len(self.fgs) > 0:
            self.fgs = self.fgs[self.fgs[:,0].argsort()]

        molecule_id = self.molecule_name + 'N'
        for i in range(len(self.n_positions)):
            molecule_id = molecule_id + '-' + str(self.n_positions[i
])

        molecule_id = molecule_id + '-DB'
        for i in range(len(self.db_positions)):
            molecule_id = molecule_id + '-' + str(self.db_positions[
i])

        molecule_id = molecule_id + '-FG'
        for i in range(len(self.fgs)):
            molecule_id = molecule_id + '--' + str(self.fgs[i][0]) +
 ',' + str(self.fgs[i][1])

        molecule_id = molecule_id + '-L'+str(self.linker)

        self.molecule_id = molecule_id
        save_mol = open('molecules/mol-files/'+molecule_id+'.mol',
mode='w')
```

```
        save_mol.write(self.mol_string)

    def add_n(self):

        number_N_to_add = random.randint(0,self.max_number_n)

        self.n_positions = []
        if number_N_to_add == 0:
            pass

        if number_N_to_add > 6:
            number_N_to_add = 6

        if number_N_to_add > 0:
            if self.double == True:
                position_to_add = random.sample((1,2,3,4,5),
number_N_to_add)
            else:
                position_to_add = random.sample((1,2,3,4,5,6),
number_N_to_add)

            for i in range(len(position_to_add)):

                available = utils.check_valency(position_to_add[i],
self.mol_string,self.double)

                self.n_positions.append(position_to_add[i])

                if available == True:
                    modified_string = ''
                    count_lines_to_position = 0
                    for line in self.mol_string.split('\n'):
                        if count_lines_to_position == 3 +
```

```
position_to_add[i]:
                        line = line.replace(line,line[0:30]+' N
 ' + line[34:70])
                    modified_string = modified_string + line + '
\n'
                    count_lines_to_position =
count_lines_to_position + 1

                self.mol_string = modified_string

 def add_fg(self):

     number_fg = random.randint(1,self.max_number_fg)

     self.benchmark_atom_number = 6
     self.benchmark_bond_number = 6

     self.fgs = []
     self.save_fg_index = []

     for i in range(number_fg):

         if self.double == True:
             position_to_add = random.choice((1,2,3,4,5))
         else:
             position_to_add = random.choice((1,2,3,4,5,6))

         random_fg = random.choice((1,2,3,4,5,6,7,8,9))
         random_fg = str(random_fg)
         fg_data_file = './buildingblocks/fgroup/%s.mol' %(
random_fg)

         available = utils.check_valency(position_to_add,self.
```

```
mol_string,self.double)

            if available == True:
                self.fgs.append([int(position_to_add),int(random_fg)
])

                self.mol_string, self.benchmark_atom_number, self.
benchmark_bond_number, fg_index = utils.build_fg(self.mol_string
,fg_data_file,position_to_add,self.benchmark_atom_number,self.
benchmark_bond_number)

                self.save_fg_index.append(fg_index)

    def add_db(self):

        number_db = random.randint(0, self.max_number_db)

        self.db_positions = []

        for i in range(number_db):
            position_for_double = random.choice((1,2,3,4,5,6))

            neighbors = np.zeros((2))
            if position_for_double == 1:
                neighbors[0] = 6
                neighbors[1] = 2
            if position_for_double == 2:
                neighbors[0] = 1
                neighbors[1] = 3
            if position_for_double == 3:
                neighbors[0] = 2
                neighbors[1] = 4
            if position_for_double == 4:
```

```
                neighbors[0] = 3
                neighbors[1] = 5
            if position_for_double == 5:
                neighbors[0] = 4
                neighbors[1] = 6
            if position_for_double == 6:
                neighbors[0] = 5
                neighbors[1] = 1

            neighbor1 = int(neighbors[0])
            neighbor2 = int(neighbors[1])

            available_position = utils.check_valency(
position_for_double,self.mol_string,self.double)
            available_neighbor = utils.check_valency(neighbor2,self.
mol_string,self.double)

            if available_position == True and available_neighbor ==
True:
                modified_string = ''
                for line in self.mol_string.split('\n'):
                    if ' ' + str(position_for_double) + ' ' in line
[0:7] and ' ' + str(neighbor2) + ' ' in line[0:7] and int(line
[8]) != 2:
                        line = line.replace(line, line[0:7] + ' 2 '
+ line[10:14])
                        self.db_positions.append(position_for_double
)

                    modified_string = modified_string + line + '\n'

                self.mol_string = modified_string
```

```
def symmetrize(self):

    mol_string = self.mol_string
    number_atoms = self.benchmark_atom_number
    number_bonds = self.benchmark_bond_number

    atoms = []
    count_lines = 0
    count_atoms = 1
    for line in self.mol_string.split('\n'):
        if count_lines > 3:
            if count_atoms == 2:
                if 'N' in line:
                    atoms.append('N')
                if 'C' in line:
                    atoms.append('C')
            if count_atoms == 6:
                if 'N' in line:
                    atoms.append('N')
                if 'C' in line:
                    atoms.append('C')
            if count_atoms == 3:
                if 'N' in line:
                    atoms.append('N')
                if 'C' in line:
                    atoms.append('C')
            if count_atoms == 5:
                if 'N' in line:
                    atoms.append('N')
                if 'C' in line:
                    atoms.append('C')
            count_atoms = count_atoms + 1
```

```
            count_lines = count_lines + 1


        count_lines = 0
        new_string = ''
        for line in self.mol_string.split('\n'):
            if count_lines == 3 + 2:
                line = line.replace(atoms[0],atoms[3])
            if count_lines == 3 + 6:
                line = line.replace(atoms[3],atoms[0])
            if count_lines == 3 + 3:
                line = line.replace(atoms[1],atoms[2])
            if count_lines == 3 + 5:
                line = line.replace(atoms[2],atoms[1])


            new_string = new_string + line + '\n'
            count_lines = count_lines + 1


        new_string = new_string.strip()
        count_lines = 0
        new_string2 = ''
        for line in new_string.split('\n'):
            if 3+number_atoms < count_lines <= 3+number_atoms +
number_bonds and 'END' not in line:
                int_number1 = int(line[1:4])
                int_number2 = int(line[4:7])
                new_number1_int = int_number1 + number_atoms
                new_number2_int = int_number2 + number_atoms
                new_number1 = str(new_number1_int)
                new_number2 = str(new_number2_int)
                if new_number1_int < 10 and new_number2_int < 10:
                    line = line.replace(line[0:7], '  '+ new_number1
 + '  ' + new_number2+' ')
                if new_number1_int < 10 and new_number2_int >= 10:
```

```
                line = line.replace(line[0:7], '  ' +
new_number1 + ' ' + new_number2+' ')
            if new_number1_int >= 10 and new_number2_int < 10:
                line = line.replace(line[0:7], ' ' + new_number1
 + '  ' + new_number2+' ')
            if new_number1_int >= 10 and new_number2_int >= 10:
                line = line.replace(line[0:7], ' ' + new_number1
 + ' ' + new_number2+' ')
        new_string2 = new_string2 + line + '\n'
        count_lines = count_lines + 1

    count_lines = 0
    new_string3 = ''
    for line in new_string2.split('\n'):
        replaced_2 = False
        replaced_3 = False
        if 3+number_atoms < count_lines <= 3+number_atoms +
number_bonds:
            if ' ' + str(2+number_atoms) +' ' in line[0:7] or '
' + str(3+number_atoms) +' ' in line[0:7]:
                if 2+number_atoms < 10 and 6+number_atoms >= 10:
                    line = line.replace(' ' + str(2+number_atoms
) +' ', str(6+number_atoms)+' ')
                    replaced_2 = True
                else:
                    line = line.replace(' ' + str(2+number_atoms
) +' ',' ' + str(6+number_atoms)+' ')
                    replaced_2 = True
                if 3+number_atoms < 10 and 5+number_atoms >= 10:
                    line = line.replace(' ' + str(3+number_atoms
) +' ', str(5+number_atoms)+' ')
                    replaced_3 = True
                else:
```

```
                    line = line.replace(' ' + str(3+number_atoms
) +' ',' '+ str(5+number_atoms)+' ')
                    replaced_3 = True

            if ' ' + str(5+number_atoms) +' ' in line[0:7] or '
' + str(6+number_atoms) +' ' in line[0:7]:
                if replaced_2 == False:
                    if 6+number_atoms >= 10 and 2+number_atoms
<10:
                        line = line.replace(str(6+number_atoms)
+' ', ' ' + str(2+number_atoms) +' ')
                    else:
                        line = line.replace(' ' + str(6+
number_atoms) +' ', ' ' + str(2+number_atoms) +' ')
                if replaced_3 == False:
                    if 5+number_atoms >=10 and 3+number_atoms <
10:
                        line = line.replace(str(5+number_atoms)
+' ', ' ' + str(3+number_atoms) +' ')
                    else:
                        line = line.replace(' ' + str(5+
number_atoms) +' ', ' ' + str(3+number_atoms) +' ')
        new_string3 = new_string3 + line + '\n'
        count_lines = count_lines + 1

    final_string = ''
    count_lines = 0
    for line in self.mol_string.split('\n'):
        if count_lines <= 3:
            if '  6  6' in line[0:7]:
                if number_atoms >= 10 and number_bonds >= 10:
                    line = line.replace(' 6  6',str(2*
number_atoms)+' '+str(2*number_bonds))
```

```
                if number_atoms >= 10 and number_bonds < 10:
                    line = line.replace(' 6  6',str(2*
number_atoms)+' '+str(2*number_bonds))
                if number_atoms < 10 and number_bonds >= 10:
                    line = line.replace(' 6  6','' + str(2*
number_atoms)+' '+str(2*number_bonds))
                if number_atoms < 10 and number_bonds < 10:
                    line = line.replace(' 6  6','' + str(2*
number_atoms)+' '+str(2*number_bonds))
            final_string = final_string + line + '\n'

        if 3 < count_lines <= 3 + number_atoms:
            float_x = float(line[2:11])
            new_float_x = float_x - 4.3234233
            new_float_x = round(new_float_x,4)
            new_float_x = str(new_float_x)

            line = line.replace(line[3:10],''+new_float_x)
            final_string = final_string + line + '\n'
        count_lines = count_lines + 1

    count_lines = 0
    for line in new_string3.split('\n'):
        if 3 < count_lines <= 3 + number_atoms:
            final_string = final_string + line + '\n'
        count_lines = count_lines + 1

    count_lines = 0
    for line in self.mol_string.split('\n'):
        if  3 + number_atoms < count_lines <= 3+ number_atoms+
number_bonds:
            final_string = final_string + line + '\n'
        count_lines = count_lines + 1
```

```
        count_lines = 0
        for line in new_string3.split('\n'):
            if  3 + number_atoms < count_lines <= 3+ number_atoms+
number_bonds:
                final_string = final_string + line + '\n'
            count_lines = count_lines + 1

        final_string = final_string + 'M  END'

        self.mol_string = final_string

    def linker(self):

        number_bonds = self.benchmark_bond_number
        number_atoms = self.benchmark_atom_number

        number_bonds = number_bonds*2 + 1

        if self.double == True:
            random_linker = 100
            random_linker = str(random_linker)
            linker_data_file = './buildingblocks/linker/%s.mol' %(
random_linker)

            self.linker = random_linker
        else:
            random_linker = random.choice((1,2,3,5))
            random_linker = str(random_linker)
            linker_data_file = './buildingblocks/linker/%s.mol' %(
random_linker)

            self.linker = random_linker
```

```
        count_lines = 0
        modified_string = ''
        for line in self.mol_string.split('\n'):
            if count_lines == 3:
                line = line.replace(line[0:6],' '+ str(number_atoms
*2) + ' ' + str(number_bonds))
            if count_lines == 3+2*number_atoms+1:
                for line2 in fileinput.FileInput(linker_data_file,
inplace=0):
                    if 'a' in line2:
                        if number_atoms+6-4 < 10:
                            line2 = line2.replace('a',' '+ str(
number_atoms+6-4))
                        else:
                            line2 = line2.replace('a', str(
number_atoms+6-4))
                    if 'm' in line2:
                        if number_atoms*2+1 < 10:
                            line2 = line2.replace('m',' ' + str(
number_atoms*2+1))
                        elif number_atoms*2+1 >= 10:
                            line2 = line2.replace('m', str(
number_atoms*2+1))
                    if 'n' in line2:
                        if number_atoms*2+2 < 10:
                            line2 = line2.replace('n',' ' + str(
number_atoms*2+2))
                        elif number_atoms*2+2 >= 10:
                            line2 = line2.replace('n', str(
number_atoms*2+2))
                    if 'o' in line2:
                        if number_atoms*2+3 < 10:
```

```
                        line2 = line2.replace('o',' ' + str(
number_atoms*2+3))
                    elif number_atoms*2+3 >= 10:
                        line2 = line2.replace('o', str(
number_atoms*2+3))
                if 'p' in line2:
                    if number_atoms*2+4 < 10:
                        line2 = line2.replace('p',' ' + str(
number_atoms*2+4))
                    elif number_atoms*2+4 >= 10:
                        line2 = line2.replace('p', str(
number_atoms*2+4))
                modified_string = modified_string + line2
            break
        modified_string = modified_string + line + '\n'
        count_lines = count_lines + 1

    modified_string = modified_string + '\n'
    for line in self.mol_string.split('\n'):
        if ' 1  0 ' in line or ' 2  0 ' in line or ' 3  0 ' in
line:
            modified_string = modified_string + line + '\n'
        if 'END' in line:
            modified_string = modified_string + line + '\n'

    self.mol_string = modified_string

 def finalize(self):

    number_atoms = 0
    number_bonds = 0
    count_lines = 0
    for line in self.mol_string.split('\n'):
```

```
            if count_lines > 4:
                if 'C' in line or 'N' in line or 'O' in line or 'F'
in line or 'Cl' in line or 'S' in line or 'P' in line:
                    number_atoms = number_atoms + 1
                if '1  0' in line[8:12] or '2  0' in line[8:12] or '
3  0' in line[8:12]:
                    number_bonds = number_bonds+1
            count_lines = count_lines + 1

        final_string = ''
        count_lines = 0
        for line in self.mol_string.split('\n'):
            if '0999' in line:
                line = line.replace(' '+ line[1:3]+' '+line[4:6],' '
 + str(number_atoms)+' ' + str(number_bonds))
            final_string = final_string+ line + '\n'
            count_lines = count_lines + 1

        self.number_atoms = number_atoms
        self.number_bonds = number_bonds
        self.mol_string = final_string

    def find_fg(self):

        for file in os.listdir('buildingblocks/fgroup/'):

            fg_lines = []
            for line in fileinput.FileInput('buildingblocks/fgroup/'
+file,inplace=0):
                if ' C ' in line or ' N ' in line or ' O ' in line
or ' Cl ' in line:
                    fg_lines.append(line[0:71])
```

```
            mol_string = self.mol_string.split('\n')

            fg = True
            for index in range(len(fg_lines)):
                if fg_lines[index] in mol_string:
                    mol_string_index = mol_string.index(fg_lines[
index])
                if fg_lines[index] not in mol_string:
                    fg = False
                    break

            if fg == True:
                print('file',file,'index',mol_string_index)

 def remove(self):

     if len(self.fgs) > 0:
            for i in range(len(self.fgs)):

                filename = './buildingblocks/fgroup/' + str(self.fgs
[i][1]) +'.mol'

                fg_numb_atoms = utils.count_fg_atoms(filename)
                indices_to_remove = [x for x in range(self.
save_fg_index[i],self.save_fg_index[i]+fg_numb_atoms)]

                new_string = ''
                count_lines = 0
                shift1 =0
                for line in self.mol_string.split('\n'):

                    if count_lines < 4:
```

```
                    new_string = new_string + line + '\n'

                if  3 < count_lines < self.number_atoms + 4 and
count_lines-3 not in indices_to_remove:
                    new_string = new_string + line + '\n'

                if  self.number_atoms + self.number_bonds + 3 >=
 count_lines > self.number_atoms + 3 and 'END' not in line:
                    if int(line[1:3]) in indices_to_remove or
int(line[4:6]) in indices_to_remove:
                        shift1 = shift1 + 1

                    if int(line[1:3]) not in indices_to_remove
and int(line[4:6]) not in indices_to_remove:
                        new_string = new_string + line + '\n'

                if 'END' in line:
                    new_string = new_string + line + '\n'

                count_lines = count_lines+1

            final_string = utils.adjust(new_string,
indices_to_remove[len(indices_to_remove)-1],len(
indices_to_remove),self.number_atoms,self.number_bonds, shift1)

            save_mol = open('molecules/removed/mol-files-removed
/'+self.molecule_id+'-REM'+str(i)+'.mol',mode='w')
            save_mol.write(final_string)
```

A.2 stability.py

```
import os
```

```
class remove_fg():
    def __init__(self,mol_dir=''):

        filenames = os.listdir(mol_dir)

        mol_files = []
        for file in filenames:
            if 'N' in file and 'DB' in file and '.mol' in file:
                mol_files.append(file)

        self.mol_files = mol_files

    def remove()
```

A.3 utils.py

```
import random
import fileinput
from re import L

def check_valency(position_to_add,mol_string,double):

    valency = 0

    available = False

    nitrogen_present = False
    count_lines = 0
    for line in mol_string.split('\n'):
        if count_lines == 3 + position_to_add:
            if ' N ' in line:
```

```
                nitrogen_present = True
        count_lines = count_lines + 1

    if nitrogen_present == True:
        valency =  valency + 1

    if position_to_add == 6 and double == True:
        valency = valency + 2

    if position_to_add == 6 and double == False:
        valency = valency + 1

    count_lines = 0
    for line in mol_string.split('\n'):
        if count_lines > 3:
            if ' ' + str(position_to_add) + ' ' in line[0:7]:
                valency = valency + int(line[8])
        count_lines = count_lines + 1

    if valency < 4:
        available = True
    if valency >= 4:
        available = False

    return available

def build_fg(mol_string, fg_data_file, position_for_fg,
   benchmark_atom_number,benchmark_bond_number):

    print(mol_string)
    number_added_atoms = 0
    number_added_bonds = 0
    modified_string = ''
```

```
 for line in mol_string.split('\n'):
     if ' 1  0 ' in line or ' 2  0 ' in line or ' 3  0 ' in line:
         for line2 in fileinput.FileInput(fg_data_file,inplace=0)
:
             if 'a' in line2:
                 if benchmark_atom_number >= 9:
                     line2 = line2.replace('a',' ' +str(
position_for_fg))
                 elif benchmark_atom_number < 9:
                     line2 = line2.replace('a', ' ' + str(
position_for_fg))
             if 'm' in line2:
                 if benchmark_atom_number >= 9:
                     line2 = line2.replace('m', str(
benchmark_atom_number+1))
                 elif benchmark_atom_number < 9:
                     line2 = line2.replace('m', ' ' + str(
benchmark_atom_number+1))
             if 'n' in line2:
                 if benchmark_atom_number >= 8:
                     line2 = line2.replace('n', str(
benchmark_atom_number+2))
                 elif benchmark_atom_number < 8:
                     line2 = line2.replace('n', ' ' + str(
benchmark_atom_number+2))
             if 'o' in line2:
                 if benchmark_atom_number >= 7:
                     line2 = line2.replace('o', str(
benchmark_atom_number+3))
                 elif benchmark_atom_number < 7:
                     line2 = line2.replace('o', ' ' + str(
benchmark_atom_number+3))
             if 'p' in line2:
```

```
                if benchmark_atom_number >= 6:
                    line2 = line2.replace('p', str(
benchmark_atom_number+4))
                elif benchmark_atom_number < 6:
                    line2 = line2.replace('p', ' ' + str(
benchmark_atom_number+4))
            if 'q' in line2:
                if benchmark_atom_number >= 5:
                    line2 = line2.replace('q', str(
benchmark_atom_number+5))
                elif benchmark_atom_number < 5:
                    line2 = line2.replace('q', ' ' + str(
benchmark_atom_number+5))
            if 'r' in line2:
                if benchmark_atom_number >= 4:
                    line2 = line2.replace('r', str(
benchmark_atom_number+6))
                elif benchmark_atom_number < 4:
                    line2 = line2.replace('r', ' ' + str(
benchmark_atom_number+6))
            if 's' in line2:
                if benchmark_atom_number >= 3:
                    line2 = line2.replace('s', str(
benchmark_atom_number+7))
                elif benchmark_atom_number < 3:
                    line2 = line2.replace('s', ' ' + str(
benchmark_atom_number+7))
            if ' 1  0 ' in line2 or ' 2  0 ' in line2 or ' 3  0
' in line2:
                number_added_bonds = number_added_bonds + 1
            if ' C  ' in line2 or ' N  ' in line2 or ' Cl  ' in
line2 or ' O  ' in line2 or ' F  ' in line2 or ' S  ' in line2:
                number_added_atoms = number_added_atoms + 1
```

```
                modified_string = modified_string + line2
            break
        modified_string = modified_string + line + '\n'
    for line in mol_string.split('\n'):
        if ' 1  0 ' in line or ' 2  0 ' in line or ' 3  0 ' in line:
            modified_string = modified_string + line + '\n'
        if 'END' in line:
            modified_string = modified_string + line + '\n'

    save_fg_index = benchmark_atom_number + 1
    benchmark_atom_number = benchmark_atom_number +
   number_added_atoms
    benchmark_bond_number = benchmark_bond_number +
   number_added_bonds

    mol_string = modified_string
    print(mol_string)
    return mol_string, benchmark_atom_number, benchmark_bond_number,
    save_fg_index

def count_fg_atoms(filename):

    fg_numb_atoms = 0
    for line in fileinput.FileInput(filename,inplace=0):
        if ' C  ' in line or ' N  ' in line or ' O  ' in line or '
   Cl  ' in line or ' S ' in line:
            fg_numb_atoms = fg_numb_atoms + 1

    return fg_numb_atoms

def adjust(string,index,shift,number_atoms,number_bonds,shift1):

    print('ATOMS',number_atoms)
```

```
 print('BONDS',number_bonds)

 number_atoms = number_atoms - shift
 number_bonds = number_bonds - shift1

 print('SHIFT',shift)
 print('SHIFT1',shift1)
 print('ATOMS',number_atoms)
 print('BONDS',number_bonds)

 new_string = ''
 count_lines = 0
 for line in string.split('\n'):
     full_line = line
     if (number_atoms+number_bonds+3) >= count_lines > (
number_atoms+3) and 'END' not in line:
         line1 = line.replace(line,line[0:3])
         line2 = line.replace(line,line[3:12])
         print(line)
         if int(line1[1:3]) > index:
             if int(line1[1:3])-shift1 < 10:
                 line1 = line1.replace(line1[1:3],' '+str(int(
line1[1:3])-shift1))
             if int(line1[1:3])-shift1 >= 10:
                 line1 = line1.replace(line1[1:3],str(int(line1
[1:3])-shift1))
         if int(line2[1:3]) > index:
             if int(line2[1:3])-shift1 < 10:
                 line2 = line2.replace(line2[1:3],' '+str(int(
line2[1:3])-shift1))
             if int(line2[1:3])-shift1 >= 10:
                 line2 = line2.replace(line2[1:3],str(int(line2
[1:3])-shift1))
```

```
            full_line = line1 + line2
        new_string = new_string + full_line + '\n'
        count_lines = count_lines + 1

    final_string = ''
    count_lines = 0
    for line in new_string.split('\n'):
        if '0999' in line:
            line = line.replace(' '+ line[1:3]+' '+line[4:6],' ' +
str(number_atoms)+' ' + str(number_bonds))
        final_string = final_string+ line + '\n'
        count_lines = count_lines + 1

    return final_string
```

Appendix B

Python Script for the Application of a Genetic Algorithm

```
 def crossover(self,double=False):
     parent1, parent2 = random.choices(os.listdir('./molecules_GA
/mol-files/'), k=2)
     parent1 = str(parent1)
     parent2 = str(parent2)

     #SECTIONS OF PARENT 1
     start = parent1.index('N')
     end = parent1.index('D',start+1)
     N_1 = parent1[start+1:end]

     start = parent1.index('B')
     end = parent1.index('F',start+1)
     DB_1 = parent1[start+1:end]

     start = parent1.index('G')
     end = parent1.index(',',start+1)
     FG_pos_1 = parent1[start+1:end]
```

```
        start = parent1.index(',')
        end = parent1.index('L',start+1)
        FG_1 = parent1[start+1:end]

        start = parent1.index('L')
        end = parent1.index('.',start+1)
        L_1 = parent1[start+1:end]

        #SECTIONS OF PARENT 2
        start = parent2.index('N')
        end = parent2.index('D',start+1)
        N_2 = parent2[start+1:end]

        start = parent2.index('B')
        end = parent2.index('F',start+1)
        DB_2 = parent2[start+1:end]

        start = parent2.index('G')
        end = parent2.index(',',start+1)
        FG_pos_2 = parent2[start+1:end]

        start = parent2.index(',')
        end = parent2.index('L',start+1)
        FG_2 = parent2[start+1:end]

        start = parent2.index('L')
        end = parent2.index('.',start+1)
        L_2 = parent2[start+1:end]

        def crossover_uniform(gene1,gene2,prop):
            c1 = copy.deepcopy(gene1)
            c2 = copy.deepcopy(gene2)
```

```
            for i in range(len(gene1)):
                if random.random() < prop:
                    c1[i], c2[i] = gene2[i], gene1[i]

            return[c1, c2]

        random.seed()

        ##[N atom position, double bond position, substituent
position, substituent, linker]
        gene1 = ['N'+N_1,'DB'+DB_1,'FG'+FG_pos_1+','+FG_1,'L'+L_1]
        gene2 = ['N'+N_2,'DB'+DB_2,'FG'+FG_pos_2+','+FG_2,'L'+L_2]

        offspring = crossover_uniform(gene1, gene2, 0.5)
        child1 = offspring[0]
        child2 = offspring[1]

        print('Parent 1:',parent1)
        print('Parent 2:',parent2)
        print('Child 1:',child1[0]+child1[1]+child1[2]+child1[3])
        print('Child 2:',child2[0]+child2[1]+child2[2]+child2[3])

        start = child1[2].index('G')
        end = child1[2].index(',',start+1)
        FG_pos_1 = child1[2][start+1:end]

        start = child1[2].index(',')
        end = child1[2].index('-',start+1)
        FG_1 = child1[2][start+1:end]

        file = open('./molecules_GA/crossover_GA.txt',mode='a',
encoding='utf-8')
        file.write('\n'+'Parent1:'+parent1+'\n')
```

```
        file.write('Parent2:'+parent2+'\n')
        file.close()

        #N atom positions
        global position_N_to_add
        position_N_to_add = child1[0]
        position_N_to_add = str(position_N_to_add)
        position_N_to_add = re.findall('[0-9]+',position_N_to_add)
        position_N_to_add = [eval(i) for i in position_N_to_add]
        position_N_to_add = position_N_to_add

        #double bond positions
        global position_db_to_add
        position_db_to_add = child1[1]
        position_db_to_add = str(position_db_to_add)
        position_db_to_add = re.findall('[0-9]+',position_db_to_add)
        position_db_to_add = [eval(i) for i in position_db_to_add]
        position_db_to_add = position_db_to_add

        #substituent positions
        global position_fg_to_add
        position_fg_to_add = FG_pos_1
        position_fg_to_add = str(position_fg_to_add)
        position_fg_to_add = re.findall('[0-9]+',position_fg_to_add)
        position_fg_to_add = [eval(i) for i in position_fg_to_add]
        position_fg_to_add = position_fg_to_add

        #substituent
        global fg_to_add
        fg_to_add = FG_1
        fg_to_add = str(fg_to_add)
        fg_to_add = re.findall('[0-9]+',fg_to_add)
        fg_to_add = [eval(i) for i in fg_to_add]
```

```
        fg_to_add = fg_to_add

        #linker
        global linker_to_add
        linker_to_add = child1[3]
        linker_to_add = str(linker_to_add)
        linker_to_add = re.findall('[0-9]+',linker_to_add)
        linker_to_add = [eval(i) for i in linker_to_add]
        linker_to_add = linker_to_add

        print('N atoms:', position_N_to_add)
        print('double bonds:', position_db_to_add)
        print('fgroup position:', position_fg_to_add)
        print('fgroup:', fg_to_add)
        print('linker', linker_to_add)

        if linker_to_add == [100]:
            double = True
        else:
            double = False

        self.double = double

        print(self.double)

        return position_N_to_add,position_db_to_add,
    position_fg_to_add,fg_to_add,linker_to_add
```

Curriculum Vitae

Candidate's full name: Emily Claire Anderson

Universities attended:

BSc Chemistry (First Class Honours), University of New Brunswick, 2017-2021

MSc Chemistry, University of New Brunswick, 2021-2024

Publications: N/A

Conference Presentations:

"Machine learning towards automated discovery of organic molecules for use in non-aqueous redox flow batteries" E. C. Anderson, S. De Baerdemacker, and A. El-Samman, CSC Canadian Chemistry Conference and Exhibition 2022, Calgary, AB, June 2022 (Poster).

"Machine learning towards automated discovery of organic molecules for use in non-aqueous redox flow batteries" E. C. Anderson, 2nd workshop of AARMS CRG on Mathematical Foundations of Scientific Machine Learning, Fredericton, NB, August 2023 (Talk).