UNDERGRADUATE DISSERTATION

COMP39X: Novel Cocrystal Prediction and Visualisation

Author:
Alexander BAGNALL
StudentID: 200995832

Supervisors:
Dr. Vitaliy KURLIN
Dr. Matthew Dyer

A dissertation submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Science

May 17, 2018

Declaration of Authorship

I, Alexander BAGNALL, declare that this dissertation titled, "COMP39X: Novel Cocrystal Prediction and Visualisation" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this dissertation has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this dissertation is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

UNIVERSITY OF LIVERPOOL

Abstract

School of Electrical Engineering, Electronics and Computer Science
Department of Computer Science

Bachelor of Science

COMP39X: Novel Cocrystal Prediction and Visualisation

by Alexander BAGNALL

Cocrystals, a long known but understudied class of crystalline solids, have attracted interest from material scientists. This is largely because they have shown potential to revolutionise many different areas such as the pharmaceutical and energy industries. This project considers whether the Cambridge Structural Database, the world's largest repository of crystal structures, can be used as the basis of a predictive modelling tool to discover new cocrystals. Strategies to create a set of cocrystals containing a common molecule and then use this data to predict the probability of molecules successfully forming cocrystals are assessed in this work. A tool has been created which searches for, extracts, and compares crystal structures stored in the database using inbuilt methods. This tool was used to find groups of organic cocrystals containing buckminsterfullerene whose chemical structures were calculated to be similar using the tanimoto index. The study covers how machine learning can be applied to crystallography and uses predictive methods to harness to potential of the Cambridge Structural Database. While clustering proved a useful first step towards the aim of predicting new cocrystal structures, more complex techniques must be explored.

Contents

Declaration of Authorship i

Abstract ii

1 Introduction 1
- 1.1 Discovering New Materials . . . 1
  - 1.1.1 Importance of Material Science . . . 1
  - 1.1.2 Applications of New Materials . . . 1
    - Pharmaceuticals . . . 1
    - Superconductors . . . 1
  - 1.1.3 Discovering New Materials using Machine Learning . . . 2
    - Current Methods to Discover New Materials . . . 2
    - Current Methods to Calculate Chemical Properties . . . 2
- 1.2 Research Aims: Machine Learning and Material Science . . . 3
- 1.3 Key Challenges . . . 3
- 1.4 Challenge Solutions . . . 4
- 1.5 Content Covered in this Report . . . 4

2 Background 6
- 2.1 Chemistry and Cheminformatics . . . 6
  - 2.1.1 Crystals and Cocrystals . . . 6
  - 2.1.2 Tanimoto Similarity . . . 6
  - 2.1.3 Chemical Similarity . . . 6
  - 2.1.4 Density Functional Theory . . . 7
  - 2.1.5 Packing Efficiency . . . 7
  - 2.1.6 Crystal Infographic File . . . 7
- 2.2 The Cambridge Structural Database . . . 9
  - 2.2.1 The World's Largest Repository of Crystals . . . 9
  - 2.2.2 Contents of an Entry . . . 9
    - Entry Information . . . 9
    - Crystal Information . . . 10
    - Molecule Information . . . 10
  - 2.2.3 ConQuest . . . 10
- 2.3 Computer Science . . . 11
  - 2.3.1 Graph Theory . . . 11
    - Graph . . . 11
    - Adjacency Matrix . . . 11
    - Paths and Cycles . . . 12
    - Tree . . . 12
    - Minimum Spanning Tree . . . 13
  - 2.3.2 Single-Edge Clustering . . . 13

3 Data Required **14**
3.1 Synthetic Data . . . 14
3.1.1 The Cambridge Structural Database . . . 14
3.1.2 Dependencies . . . 14

4 Design **15**
4.1 Anticipated Components of the System . . . 15
4.1.1 Extract entries . . . 15
4.1.2 Create a Similarity Dataset . . . 15
4.1.3 Reduce the Similarity Dataset to a Minimum Spanning Tree . . . 15
4.1.4 Visualise the Tree . . . 15
4.2 Data Structures Used by the System . . . 16
4.2.1 Arrays . . . 16
4.2.2 Matrices . . . 16
4.2.3 Trees . . . 16
4.3 Algorithms used . . . 17
4.3.1 Similarity . . . 17
4.3.2 Compare . . . 17
4.3.3 Kruskal . . . 18
4.3.4 Visualisation . . . 19
4.4 Design of the intended interface . . . 20
4.4.1 User Input Interface . . . 20
4.4.2 Visualisation Interface . . . 20
4.5 Coding Languages Used . . . 20
4.5.1 Python2.7 . . . 20
4.5.2 JavaScript . . . 21

5 Realisation **22**
5.1 Implementation of the Design . . . 22
5.1.1 Extracting Desired Entries from the User Input . . . 22
5.1.2 Creating the Similarity Matrix . . . 22
5.1.3 Creating the Minimum Spanning Tree . . . 23
5.1.4 Visualising the Minimum Spanning Tree . . . 25
5.2 Problems encountered during implementation . . . 26
5.2.1 Usage of python2.7 by the CSD . . . 26
5.2.2 The optimal way of extracting entries . . . 27
5.2.3 Limitations of Python for Visualisation . . . 28
5.3 Changes made to the design during implementation . . . 28
5.3.1 Alteration of user input . . . 28

6 Evaluation **29**
6.1 Evaluation of the System . . . 29
6.1.1 Criteria used to the success of the system . . . 29
6.1.2 Test case: C60 . . . 29
Input and creating a dataset . . . 29
Visualisation . . . 30
Prediction . . . 30
6.2 Strengths and Weaknesses . . . 30
6.2.1 Strengths . . . 30
6.2.2 Weaknesses . . . 32

7 Learning Points **33**
7.1 The CSD can be used as the basis of a predictive tool 33
7.2 Machine Learning Techniques can be applied to the CSD 33
7.3 Other Machine Learning Techniques must be Explored 33

8 Professional Issues **34**

Bibliography **35**

List of Figures

1.1 A molecule of C60, known as Buckminsterfullerene 3

2.1 Contents of a .cif file 8
2.2 Visual representation of the .cif file in 2.1 8
2.3 Growth of the CSD since 1972 [6] 9
2.4 [Entry descriptors] 10
2.5 [Crystal descriptors] 10
2.6 [Molecule descriptors] 10
2.7 An undirected graph 11
2.8 An adjacency matrix for the graph in Fig. 2.7 12
2.9 A tree of 10 nodes and 9 edges 12
2.10 A complete graph 13
2.11 The minimum spanning tree formed from the complete graph 13

5.1 *Code to extract entries* 23
5.2 *Code to find the similarity values* 23
5.3 *Code to fill the matrix* 24
5.4 *Code to find the similarity values* 24
5.5 *Code to find the similarity values* 25
5.6 *Code to obtain the minimum weight set by the user* 26
5.7 *Code to filter the data* 26
5.8 *Code to draw the data* 27

6.1 [C60 Results] 30
6.2 [C60 further results] 31

List of Tables

6.1 [Results: Entry descriptors] 31
6.2 [Results: Molecular descriptors] 31
6.3 [Results: Crystal descriptors] 32

List of Abbreviations

CSD	**C**ambridge **S**tructural **D**atabase
DFT	**D**ensity **F**unctional **T**heory
MST	**M**inimum **S**panning **T**ree
JSON	**J**avaScript **O**bject **N**otation
RF	**R**andom **F**orest
ERT	**E**xtremely **R**andomised **T**ree

Chapter 1

Introduction

1.1 Discovering New Materials

Material science has consistently advanced humanity throughout history. In the present day, machine learning has been used to improve a wide range areas such as transport infrastructure, online retail, and education. Therefore, there is potential to use machine learning techniques to improve the study of material science.

1.1.1 Importance of Material Science

Human civilisation is constantly propelled forward by discoveries and progress made in materials science [22]. In the stone age, the discovery of materials such as fiber and resin allowed humans to attach blades to sticks to create knives and axes [17]. Millenniums later, the discovery of the electron led to the development of the vacuum tube, the solid state transistor, and to microelectronics in general [9]. In the modern age an example of a new material advancing humanity is the two-dimensional material graphene [1], which is similar to a crystal in that it is a solid composed of a single type of molecule in a symmetrical, repeating pattern. Discovered in 2004, the material has been used to create a smart graphene coating that indicates breaks and fractures in a building by changing colour [8]. This is only one of the many applications the discovery of new materials has had.

1.1.2 Applications of New Materials

The discovery of new cocrystals, a crystal made up of two or more molecules, with novel chemical and electronic properties would have many consequences, such as in the development of new pharmaceuticals and super-conductors.

Pharmaceuticals

For example, cocrystals are becoming popular in the pharmaceutical industry as these can offer advantages such as greater bioavailability, .

The investigation of cocrystals can multiply the number and diversity of solid form choices. This in turn increases the probability of identifying suitable development candidates which present solubility, bioavailability, or stability issues [2] .

Superconductors

Superconductivity is an important and highly desirable feature of certain materials. It is a phenomenon where exactly zero electrical resistance and expulsion of magnetic flux fields occur in certain materials, called superconductors, when cooled

below a characteristic critical temperature. There are numerous applications of superconductivity being developed, such as for example to avoid energy waste. Currently, almost all transmission of electrical current is via copper wire. It is common for electricity to be lost during transmission, for instance India reported a 30% loss of electrical current in transition across their utility lines [24]. A much more efficient way of transmission is through the use of HTS powercables , which provides 0% loss of electrical current during transmission. High Temperature Superconductors, such as HTS Powercables, use much cheaper cryogens like liquid nitrogen [13].

These are but some of the important applications of discovering novel cocrystals

1.1.3 Discovering New Materials using Machine Learning

Current Methods to Discover New Materials

When attempting to discover new materials with desired chemical properties chemists use chemical intuition. This involves a trial and error approach of mixing molecules and studying how they react together with the hope that they successfully synthesise and contain the desired properties. This method offers no guarantee, is slow, and solely depends on the ability of the chemist [14].

Current Methods to Calculate Chemical Properties

The ability to anticipate new crystal structures with certain properties, this could advance materials science dramatically, however this is a hard problem. Chemists currently perform simulations based on density functional theory (DFT). To calculate potential crystals properties the electronic structure of many-body systems are investigated. These simulations can foretell the mechanics of the atoms, and thus determine the magnetic and electronic properties of a material. One crucial issue of using this method is that a structure must be known before its properties can be calculated. Therefore, a system which infers both the probability of crystals forming and their potential structure would be of high value. However, such predictions would require the simulation of complex physical systems which would be computationally intractable.

The Cambridge Structural Database (CSD) is both a repository containing validated and curated three-dimensional structural data of crystal structures. This database is the world's most comprehensive and up-to-date database of crysal structures, with over 900,000 curated entries. Moreover, it is continually updated with new structures (more than 50,000 new structures each year) and with improvements to existing entries. Every entry is enriched with bibliographic, chemical, and physical property information, adding further value to the raw structural data. This unique database of accurate 3D structures has become an essential resource to scientists around the world[6].

A new idea would be to use the CSD as the basis of a predictive modelling tool. With such a large amount of data, the CSD would be an ideal choice to apply machine learning techniques to. The program would use information extracted from the CSD to form predictions about the success of certain molecules cocrystallising. Furthermore, it would also show the user which entries would be useful when predicting new structures. Such information could then be used to apply DFT to predict the properties of cocrystals with high chance of formation. Thus allowing the hypothetical and desired cocrystal to then be synthesised.

1.2 Research Aims: Machine Learning and Material Science

Over recent years, machine learning techniques, such as clustering, have been applied to problems where a large dataset has been available. These techniques have been able to infer patterns and features from data, which have allowed for predictions to be made. For example, logistic regression analysis, a statistical method for analyzing a dataset [10], has been used to predict which chemical groups are best for making chiral molecules, which in turn offers a new way of designing chiral crystals [20].

In the following, the problem of whether machine learning can be applied to the CSD is considered. Given the depth and breadth of information contained in the database, enough data is available. This project will use Buckminsterfullerene (C60) to test the software. C60 is a molecule of carbon in the form of a hollow sphere containing 60 carbon atoms, see Figure 1.1. Crucially, when alkali metals are doped into C60 the molecule has the potential to convert to a superconductor[16]. Hence, it would be desirable to find molecules with a high probabilty of forming a new material with C60.

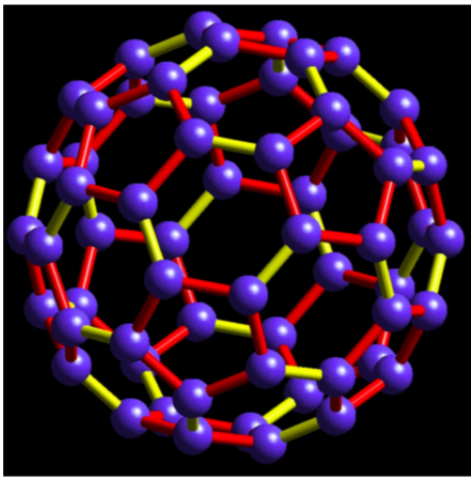


FIGURE 1.1: A molecule of C60, known as Buckminsterfullerene

There are many methods of machine learning which could be applied to the CSD. This project will use clustering to find groups of database entries which are considered similar. To do this areas within cheminformatics will be combined with graph theory. Computer Science and machine learning have been successfully applied to many fields, the aim of this project is to determine if it can be successfully applied to Crystallography.

1.3 Key Challenges

There are three major challenges for the project. First, the project must determine whether the CSD can be used as the basis of a predictive modelling tool. A single entry from the CSD provides information about the molecular weight, packing coecient, temperature the crystal was synthesised at, and more. With such a significant amount of data available the CSD has great potential to provide enough data for the successful application of machine learning algorithms.

Second, this project needs to successfully visualise the data extracted from the CSD in a useful manner. This is necessary as the user must be able to visually interpret the data. This will avoid the issue of the user needing to create their own program to interpret the data as such a requirement would need the user to possess a strong understanding of python. Hence, the data must be modified to clearly show the most important information by separating the data into groups.

Third, this project will attempt to predict which molecules will successfully form a cocrystal. With the relevant data extracted and clustered into groups, patterns can be found. By inspecting what causes these groups to emerge, conclusions can be drawn about how molecules react when mixed.

1.4 Challenge Solutions

First, to determine whether the CSD can be used as the basis of a predictive modelling tool, two areas must be explored: Can user input be serve as useful means to obtain a dataset. Can this dataset be reduced to a more specific set of data. There are many ways to to extract entries from the CSD. For instance, a program can find entries by searching for chemical names or formula. Thus this project must use the most efficient method to create a dataset of entries relevant to the users input. The user must then have the option of reducing this dataset to both reduce computation and the possibility of error. This can be achieved by applying criteria to each entry, for example whether it is an organic molecule, or if it made up of at least two components. This will allow the program to make more stringent predictions and give the user more precise results.

Next, to visualise the data in a useful manner graph theory will be utilised. By presenting each entry as a node and connecting each node depending on a measure, the data can be more easily interpreted. By explaining to the user what the measure which connects nodes is based on and allowing them to interact with the graph, it will then be possible to cluster the nodes. These clustered entries can then reveal vital information about similarities between certain crystals which may not be obvious when reading through data.

Finally, in order to make predictions, the clusters must be inspected. With so much data available there are many different areas this project could attempt to predict. While it would be useful to attempt to predict chemical properties of potential cocrystals this is possible with DFT. However, as DFT requires a known crystal structure it would be beneficial to predict what molecules have a high probability of forming a cocrystal. This would give chemists an idea of a structure of a potential cocrystal which would allow them to apply DFT to predict the properties. As a result they could decide whether to attempt to synthesise the cocrystal. Hence, when inspecting the clusters the molecules which make up the entries will be considered and used to calculate the probability of forming a cocrystal.

1.5 Content Covered in this Report

This report covers the background of the problem, the design and realisation of the solution, an evaluation of the solution, and conclusions of the project. Information

about the data used and professional issues are also included.

First in section 2, relevant background about the project and the methods used are introduced. Next, in section 3 the data required for this project to be completed is . Then, in section 4 the components of the system, an explanation of the data structures used by the system, information about which algorithms were used by the system, what coding languages are used by the system, and examples of the user interface are explained.

Chapter 2

Background

2.1 Chemistry and Cheminformatics

In this section, background about the chemical aspects of this project and the current methods to analyse these are explained.

2.1.1 Crystals and Cocrystals

A crystal is a solid composed of atoms, ions, or molecules arranged in a pattern that is periodic in three dimensions. Crystals are usually presented in terms of their unit cell. The unit cell is the smallest part of a crystal that, if repeated regularly by translation in three dimensions, creates the whole crystal.

A cocrystal is defined as a solid consisting of a crystalline single-phase material composed of two or more different molecular and/or ionic compounds generally in a stoichiometric ratio which are neither solvates nor simple salts.

2.1.2 Tanimoto Similarity

The Jaccard index is a statistic used for comparing the similarity and diversity of sets. Tanimoto similarity is a synonym for the Jaccard index but can be mathematically different. The method of classification is based on a similarity ratio. In the paper "A Computer Program for Classifying Plants", published in October 1960 [19], a classification method based on a similarity ratio, and a derived distance function, are given. This ratio is equivalent to Jaccard similarity, but the distance function is not the same as Jaccard distance.

In the paper, the similarity ratio is given over bitmaps, each bit of an array represents the presence or absence of a characteristic in a sample. The definition of the ratio is the number of common bits of data, divided by the number of bits (which is never zero) in a sample. If samples X and Y are bitmaps, X_i is the ith bit of X, and $\bigwedge$, $\bigvee$ are bitwise and, or logical operators. The similarity ratio, which is a value between 0 and 1, T_s is:

$$T_s(X, Y) = \frac{\sum(X_i \bigwedge Y_i)}{\sum(X_i \bigvee Y_i)}$$

2.1.3 Chemical Similarity

The idea of chemical similarity is one of the most important concepts in chemoinformatics. The similar property principle of Johnson and Maggiora states: compounds

with similar structures have similar properties [15]. This principle plays an important role in modern approaches to predicting the properties of chemical compounds, designing chemicals with a predefined set of properties, and in conducting drug design studies by screening large databases containing structures of available or potentially available chemicals.

Similarity-based virtual screening assumes that all compounds in a database that are similar to a query compound have similar biological activity. To achieve high efficacy of similarity-based screening of databases containing millions of compounds, molecular structures are usually represented by molecular screens (structural keys) or by fixed-size or variable-size molecular fingerprints. Molecular screens and fingerprints can contain both two-dimensional and three-dimensional information.

The most popular similarity measure for comparing chemical structures represented by means of fingerprints is the Tanimoto coefficient T. Two structures are usually considered similar if the tanimoto value is greater than 0.85.

2.1.4 Density Functional Theory

Density functional theory (DFT) is a method used to investigate the electronic structure of many-body systems. These simulations can predict the mechanics of the atoms, for example the magnetic and electronic properties of a material. DFT methods evaluate the electronic arrangements within a structure by constructing a potential acting on a system's electrons [12]. Therefore, a structure must be known before the materials properties can be predicted. Furthermore, chemists performing the simulation must know how the material will pack into space. This is because knowledge of the cell size and how the atoms are packed are needed to predict the most stable state of a calculated structure. For example, these measurements can be applied in molecular modeling simulation.

2.1.5 Packing Efficiency

By considering the arrangement of atoms relative to each other, their coordination numbers (or number of nearest neighbors), inter-atomic distances, types of bonding, etc. It is possible to form a general view of the structures and alternative ways of visualizing them.

2.1.6 Crystal Infographic File

A Crystallographic Infographic File (.cif) is a standard text file for representing crystallographic information. It was developed by the International Union of Crystallography. It is widely used by people handling, viewing, and determining crystal structures. This file type contains definitions about periodic boundary conditions and unit cells. This gives the user the ability to examine the intricacies of a structure to allow deeper analysis of how the results were produced [3]. An example of a .cif file can be seen in Fig. 2.1. This data can be visualised to view the structure of the crystal, this can be in Fig. 2.2.

```
data_CSD_CIF_ASOPOP
_audit_creation_date 2016-08-23
_audit_creation_method CSD-ConQuest-V1
_database_code_CSD ASOPOP
_database_code_depnum_ccdc_archive 'CCDC 1492711'
_chemical_formula_sum 'C75 H12 S8'
_chemical_formula_moiety
;
C60,C14 H12 S6,C1 S2
;
_journal_coden_Cambridge 1500
_journal_volume 6
_journal_year 2016
_journal_page_first 79978
_journal_name_full 'RSC Advances '
loop_
_publ_author_name
"Yantao Sun"
"Zili Cui"
"Lichuan Chen"
"Xiaofeng Lu"
"Yuewei Wu"
"Hao-Li Zhang"
"Xiangfeng Shao"
_chemical_name_systematic
;
1,3,6,8-tetramethylthieno[3,4-b]thieno[3',4':5,6][1,4]dithiino[2,3-e][1,4]dith
iine C60fullerene carbon disulfide
;
_chemical_melting_point 534
_cell_volume   1107.817
_exptl_crystal_colour 'black'
_exptl_crystal_density_diffrn 1.753
_exptl_crystal_description 'block'
_diffrn_ambient_temperature 110
#These two values have been output from a single CSD field.
_refine_ls_R_factor_gt 0.0526
_refine_ls_wR_factor_gt 0.0526
_symmetry_cell_setting triclinic
_symmetry_space_group_name_H-M 'P -1'
_symmetry_Int_Tables_number 2
loop_
_symmetry_equiv_pos_site_id
_symmetry_equiv_pos_as_xyz
```

FIGURE 2.1: Contents of a .cif file

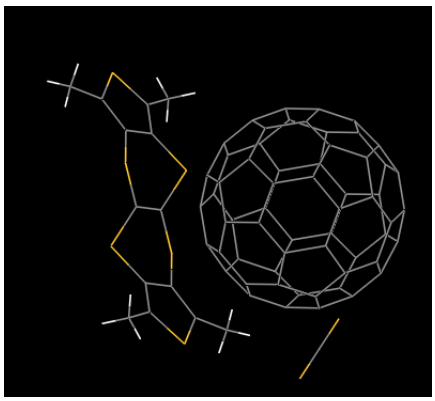


FIGURE 2.2: Visual representation of the .cif file in 2.1

2.2 The Cambridge Structural Database

This section contains information about the CSD is and the contents of the database.

2.2.1 The World's Largest Repository of Crystals

The Cambridge Structural Database (CSD) is a highly curated and comprehensive resource. Established in 1965, the CSD is the world's repository for small-molecule organic and metal-organic crystal structures. Containing over 900,000 entries, this unique database of accurate 3D structures has become an essential resource to scientists around the world. Furthermore, the CSD is continually updated with new structures (more than 50,000 new structures each year) and with improvements to existing entries [6], as seen in Fig. 2.3.

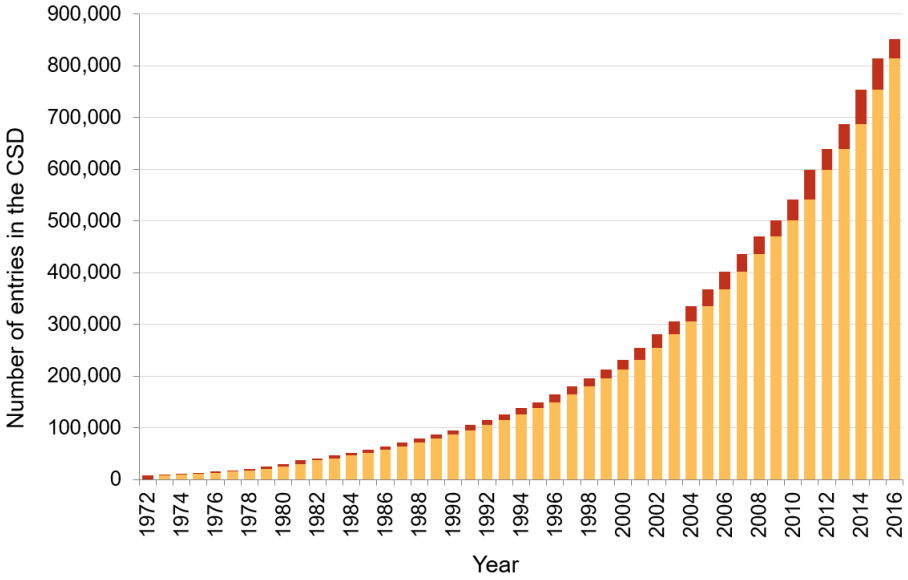


FIGURE 2.3: Growth of the CSD since 1972 [6]

Each crystal structure undergoes extensive validation and cross-checking by expert chemists and crystallographers to ensure that the CSD is maintained to the highest possible standards. Every entry is enriched with bibliographic, chemical, and physical property information, adding further value to the raw structural data. These editorial processes enable scientists to interpret structures in a chemically meaningful way.

2.2.2 Contents of an Entry

Each entry is split into three parts: the entry information, the crystallographic information, and the molecular information.

Entry Information

CSD entries contains additional information about a crystal structure added during the editorial process, such as, for instance, the chemical name and publication details [5], see Fig. 2.4.

```
>>> from ccdc.io import EntryReader
>>> csd_reader = EntryReader('CSD')
>>> pesmbd = csd_reader.entry('PESMBD')
>>> pesmbd.chemical_name
u'hexakis(2-Phenylethylthiomethyl)benzene 1,4-dioxane clathrate'
>>> pesmbd.publication
Citation(authors=u'K.Burns, C.J.Gilmore, P.R.Mallinson, D.D.MacNicol, S.Swanson', journal
="Journal(European Crystallographic Meeting)", volume=u'6', year=1980, first_page=u'23',
doi=None)
>>>
```

FIGURE 2.4: [Entry descriptors]

Crystal Information

Crystals contain crystallographic descriptors as well as the crystal structure for the entry, such as, for example, the unit cell volume and the packing coefficient [5], see Fig. 2.5.

```
>>> from ccdc import io
>>> csd_reader = io.EntryReader('CSD')
>>> crystal_pesmbd = csd_reader.crystal('PESMBD')
>>> crystal_pesmbd.packing_coefficient
0.0
>>> crystal_pesmbd.cell_volume
1481.6656203773591
>>>
```

FIGURE 2.5: [Crystal descriptors]

Molecule Information

Each entry contains information about the crystal's molecular properties, such as, for instance, the molecular weight or descriptors about whether the molecule is organic or not [5], see Fig. 2.6.

```
>>> from ccdc import io
>>> csd_reader = io.EntryReader('CSD')
>>> molecule_pesmbd = csd_reader.molecule('PESMBD')
>>> molecule_pesmbd.molecular_weight
1067.6581999999985
>>> molecule_pesmbd.is_organic
True
>>>
```

FIGURE 2.6: [Molecule descriptors]

2.2.3 ConQuest

ConQuest is the primary program for searching and retrieving information from the CSD. ConQuest provides an extensive range of flexible search options, enabling the user to search for crystal structures using [4]:

- Text and numeric searches - compound name, formula, elemental composition, literature reference and experimental details.

- Chemical substructure searches - finds substrctures present in crystals. This options includes the ability to define chemical constrains such as charge, hybridization state and cyclicity.
- 3D geometric searches - analyse molecular dimensions and determine conformational preferences.
- Intermolecular and non-bonded contact searches - explore interactions of all types and to locate pharmacophoric patterns.

The data obtained from searches can be exported as a .txt, .mol2, or .cif for later use.

2.3 Computer Science

The sections explains the key ideas from computer science, theory, and mathematics used to implement the project.

2.3.1 Graph Theory

Graph

An undirected graph is an ordered pair G $= (V, E)$ comprising a set V of vertices (or nodes) together with a set E of edges (or lines). Edges are 2-element subsets of V i.e. an edge associated with two vertices, where the association takes the form of the unordered pair comprising those two vertices. Furthermore, in a weighted graph, an edge can have an associated weight. For example in Fig2.7, an undirected graph where G $= (6, 11)$.

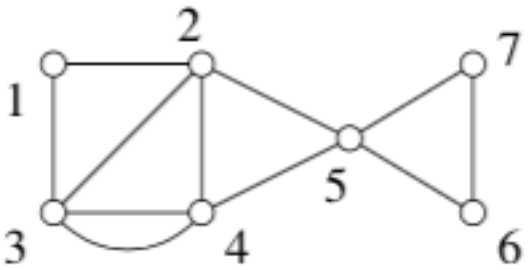


FIGURE 2.7: An undirected graph

Adjacency Matrix

The adjacency matrix of a graph G is a mathematical representation of the graph. For vertex set V, the adjacency matrix is a square matrix A such that its element A_ij is one when there is an edge from vertex i to vertex j, and zero when there is no edge. When no self-loops are allowed, i.e. a vertex cannot be connected to itself, the diagonal elements of the matrix are all zero. An example of an adjacency matrix can be seen in Fig. 2.9.

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 2 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{2.1}$$

FIGURE 2.8: An adjacency matrix for the graph in Fig. 2.7

Paths and Cycles

A path is a sequence of nodes X_1 , X_3 , ... , X_n where any two successive elements are connected by an edge [7].

A cycle is a finite path X_1 , X_3 , ... , X_n such that $X_1 = X_n$ when vertices do not repeat [7]. If repeated vertices are allowed, it is more often called a closed walk. A simple cycle is a closed walk with no repetitions of vertices and edges allowed, other than the repetition of the first and last vertex.

Tree

A tree is a particular type of undirected graph. Here, the idea is that every vertex should be connected, via a path, to any other vertex in the graph. In particular, there must only be one path taking any vertex to any other vertex. This is an example of a connected graph. If there can only be one path, then the graph can not contain cycles because this would mean there are two paths.

Alternatively, a graph is a tree if the graph contains once again no cycles, but adding one edge to it would create a cycle. This means that the original graph was connected with no cycles, and was thus a tree.

For a tree, if the graph has n nodes, then in this case it has $n - 1$ edges. If it had less than $n - 1$ edges, then it could not be connected. If it had more than $n - 1$ edges then there would be a cycle. So a tree has $n - 1$ edges with the extra condition of either being connected or having no cycles.

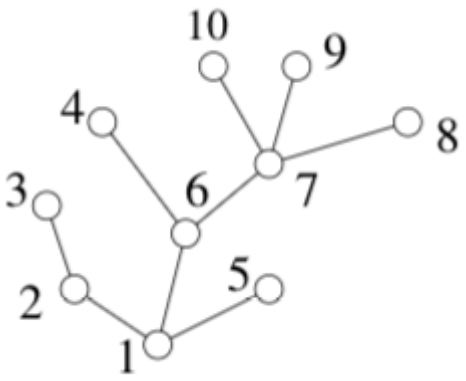


FIGURE 2.9: A tree of 10 nodes and 9 edges

Minimum Spanning Tree

A minimum spanning tree (MST) is a special kind of tree that minimises the total weight, sum of weights, over all edges. For example is a telecommunications company wanting to lay line to multiple neighbourhoods, by minimising the amount of cable laid, the telecommunications company will save money.

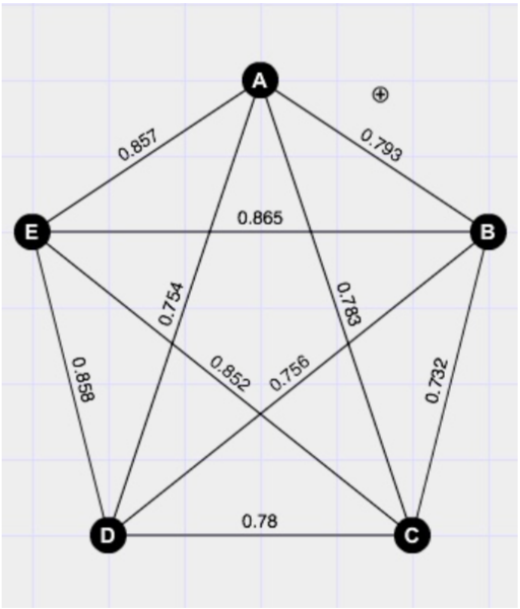


FIGURE 2.10: A complete graph

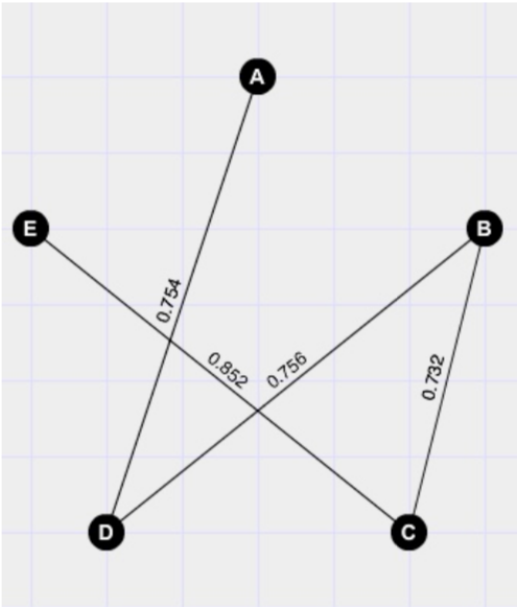


FIGURE 2.11: The minimum spanning tree formed from the complete graph

2.3.2 Single-Edge Clustering

Clustering is when a set of objects is grouped together in such a way that objects in the same group (known as a cluster) are more similar to each other than to those in other groups (clusters). The edges of a minimum spanning tree can be classified into separate groups. This is done by setting a minimum weight threshold and removing edges whose value is less than the threshold.

Chapter 3

Data Required

For this project no human data was required as the user only inputs a list of entries. Therefore, no testing was needed. This project instead relied heavily on synthetic data.

3.1 Synthetic Data

3.1.1 The Cambridge Structural Database

The synthetic data used is the Cambridge Structural Database. To use this resource a license is required. For this project I obtained this license from the University of Liverpool. For the user to use the software they must also have a license. If the user does not have an activated copy of the CSD then they will receive an error message and the code will fail to compile.

3.1.2 Dependencies

As Python2.7 was used to create this project, I have taken advantage of the import module feature. This feature has allowed me to utilise multiple third party packages, all of which are open source. I have also used a package for JavaScript to aid with visualisation.

Chapter 4

Design

4.1 Anticipated Components of the System

4.1.1 Extract entries

Initially the program obtains a set of entries from the CSD. To create this set, the user must provide a common molecule and state the sets inclusion criteria. This criteria will include the minimum and maximum number of components an entry must have. Furthermore, the user must specify what type of crystals are to be included in the set. This can be organic, organometallic, metallic, or a combination of types.

4.1.2 Create a Similarity Dataset

The set of entries will provide the information required to create the similarity dataset. The system will compare each element from the set of entries with every other element in the set. The comparison will use the inbuilt CSD function which calculates the Tanimoto similarity of two entries. Let N is the length of the set of entries. Then, N sets will be created, and each will contain N similarity measures. This information will then be stored and used to create a graph where each entry is a node and each edge is weighted by the similarity value of the two nodes.

4.1.3 Reduce the Similarity Dataset to a Minimum Spanning Tree

The similarity dataset will need to be filtered to find the useful information required for predictions. To do this Kruskal's algorithm is applied. This treats each set of similarity measures as a node and each measure an edge to another set. Kruskal's algorithm will create a minimum spanning tree by minimising the total weight, sum of weights, over all edges. Hence, it will remove all weights which connect dissimilar entries until a path is made. As high similarity measures are desired the data must be manipulated during implementation. This ensures that the minimum spanning tree will contain the highest rather than the lowest similarity measures.

4.1.4 Visualise the Tree

The tree is then visualised for the user to interpret. Initially the tree will be one single cluster. To allow the user to discover clusters of highly similar entries they will be given the option of setting a minimum edge weight. This will redraw the tree, removing the edges below the minimum weight. This will thus result in edges connecting only the entries which are structurally similar.

4.2 Data Structures Used by the System

4.2.1 Arrays

An array is a data structure consisting of collection of elements. Each element is identified by an index key. By convention, the first element of an array is at index 0.

Arrays will be used to store entry identifiers. When the entries are initially extracted the program will only require the identifier names. This is because they can be accessed at any time later on. This array will be used to create the matrix containing similarities and used by the comparison method to fill the matrix.

4.2.2 Matrices

A matrix is a data structure which acts as a two dimensional array. By convention, the first index refers to the row and the second index refers to the column of the matrix.

Matrices will be used to store the similarity values. As each entry must be compared to every other entry in the dataset, a matrix is an ideal choice. Each row and column will correspond to a specific entry in the dataset. As it is computationally intensive to apply the similarity operation on a large set of data, the matrix enables the dataset to become smaller with each iteration, see section 4.3.1. This will increase the speed of the similarity search with each iteration.

Matrices will also be used to store the components of the minimum spanning tree. The similarity matrix is treated as a complete, a graph where every vertex is connected to every other vertex by an edge, graph by Kruskal's algorithm. Every similarity measure is treated as the weight of an edge on this graph. Once the tree is made a matrix is used to store three values: the identifier of the first node, the identifier of the second node, and the weight of the connecting edge. This was necessary for visualisation.

4.2.3 Trees

The tree data structure is used to discover clusters of highly similar entries. A minimum spanning tree will be used to connect only the most similar nodes in the dataset. This will remove unnecessary data obtained when creating the similarity matrix.

The tree will allow clusters to be revealed to the user. To obtain clusters from a tree a set number of weak edges must be removed. However, the creation of a set number of clusters is not guaranteed when removing N edges. To avoid this issue and create a smooth interface the user will be able choose the minimum weight which will trim the tree of all edges below said weight.

4.3 Algorithms used

4.3.1 Similarity

For the similarity matrix, the larger a dataset is the more time is required to construct the matrix. This can result in the program spending longer than desired when comparing molecules. To reduce this issue, the first entry is chosen from the dataset, then compared with every other molecule in the dataset, and then removed. The similarity measures are then stored in the row and column corresponding to the index position of the chosen entry in the original entry array. The entry is then removed from the entry array, see 1. This final operation will increase the speed of the algorithm with every iteration as less data will be compared.

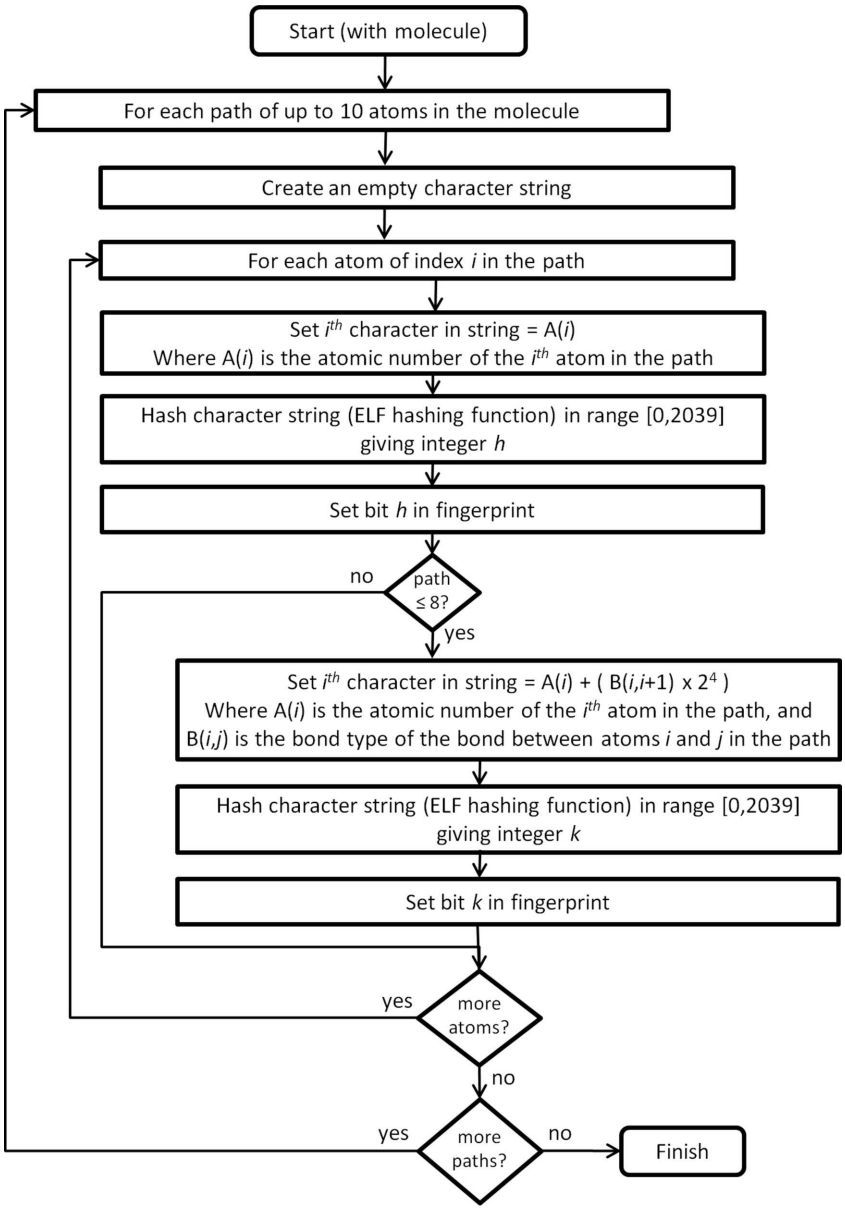
```
Data: Similarity(inputEntries)
Result: Matrix with similarity measures
matrixRow = 0
datasetLength = length(inputEntries)
similarityMatrix = [][]
while matrixRow <datasetLength do
    similarityList = []
    entryToCompare = inputEntries[0]
    for hit = 0 to datasetLength do
        similarityList[hit] = compare(entryToCompare, inputEntries[hit])
    end
    for matrixColumn = matrixRow to datasetLength do
        similarityMatrix = [matrixRow][matrixColumn]
        similarityMatrix = [matrixColumn][matrixRow]
    end
    remove entryToCompare from inputEntries
end
```

Algorithm 1: Create a matrix of similarity values

4.3.2 Compare

The compare algorithm is inbuilt into the CSD. It compares variable sized molecular fingerprints by converting each fingerprint into a string. Each string is then divided into two equal halves, these halves are then combined by a logical OR. This is repeated until the information density, the ratio of how much information it holds compared to how much it could hold, of both strings are equal. Once this has been achieved, the fingerprints are compared by applying the Tanimoto equation, see section 2.1.2.

In the following algorithm, a bond is a force of attraction between molecules which enables the formation of chemical compounds. A path refers to the linear patterns of the chemical structure, this algorithm uses bonded paths of up to 10 atoms. A fingerprint represents a molecule. Each fingerprint is determined by the chemical features of the molecules, for example the atom types, bond types, and bonded paths.



Algorithm 2: Compare two molecules [23]

4.3.3 Kruskal

To create the minimum spanning tree, Kruskal's algorithm is used. This is a greedy algorithm which finds an edge of the least possible weight that connects any two trees in a forest. As a result, a subset of edges forming a tree is created. The total weight of these edges in the tree is minimised. The tree created using Kruskal's algorithm links an entry with the most similar entry in the dataset. There are multiple other algorithms which could be used to implement a minimum spanning tree. Three other algorithms which were equally suitable for this task are Borůvka's, Prim's, and the reverse-delete algorithm. In terms of complexity Borůvka's, which runs in O($m$log$n$) [18], and Prim's do not differ much from Kruskal's. However, the reverse-delete algorithm, which is the reverse of Kruskal's, has time complexity of O($m$log$n$(log log n)^3). Let m is the number of edges and n is the number of vertices. Kruskal's algorithm is the most popular of these algorithms, for example it is used by the python module Scipy. As there is little to no difference in complexity between

these algorithms Kruskal's is a suitable choice.

Data: Kruskal(Graph)
Result: Matrix with similarity measures
A = 0
for *matrixColumn = matrixRow to datasetLength* **do**
  similarityMatrix = [matrixRow][matrixColumn]
  similarityMatrix = [matrixColumn][matrixRow]
end
for *each v ∈G.V* **do**
  MAKE-SET(v)
end
for *each (u, v) ∈ G.E ordered by weight(u, v), increasing:* **do**
  if *FIND-SET(u) ≠ FIND-SET(v):* **then**
    A = A ⋃ (u, v) UNION(u, v)
  end
end
return A

Algorithm 3: Create a minimum spanning tree

The average performance for this algorithm is $O|E|\log|V|$ and the worst case space complexity is $\Omega|E| + |V|$, where E is the number of edges and V is the number of vertices.

4.3.4 Visualisation

The visualisation algorithm reads through a matrix and draws a tree based on the values found, see section 3. Each row of the matrix is read and the third element of the row is compared to the minimumSimilarity value. If the third element is less than this value then the nodes are deemed too dissimilar and thus neither the nodes nor the edge will be added to the graph. If the third element is above this value then the row is added to the matrix used for printing. Once the initial matrix has been fully read through the printMatrix is used. If either of the node identifiers stored in the first two columns of each row are not accounted for then they are added to the node array, otherwise if they are ignored. The edgeMatrix also stores the nodes on each line of the printMatrix. Once every row has been read. The draw function maps every node in the nodeArray onto the page. Then the draw function uses the edgeMatrix to connect the nodes together.

```
Data: Visualisation(minimumSimilarity, edges)
Result: A visualised tree
edges = [nodeFrom, nodeTo, weight]
printMatrix = [][]
nodeArray = []
edgeMatrix = [][]
for row = 0 to length(edges) do
    if edges[row][2] >= minimumSimilarity then
        Add edges[row] to printMatrix
    end
end
for row = 0 to length(printMatrix) do
    Node1 = printMatrix[row][0] Node2 = printMatrix[row][1] if Node1 does
     not exists in nodeArray then
        Add Node1 to nodeArray
    end
    if Node2 does not exists in nodeArray then
        Add Node2 to nodeArray
    end
    edgeMatrix[row][0] = Node1 edgeMatrix[row][1] = Node2
end
draw(edgeMatrix, nodeArray)
```

Algorithm 4: Algorithm for visualising the Minimum Spanning Tree

4.4 Design of the intended interface

4.4.1 User Input Interface

The user interface will be designed for the command prompt. This is because there will be only 3 steps necessary after the user has begun running the program. The first step involves the user inputting the a common molecule which every entry must contains. The second step requires the user to state which type of crystals they want their dataset to made up of. The choices for this step will be: organic, organometallic, metallic, or all of the above. The third step requires the user to state the minimum and maximum number of components each entry in their dataset must have. This will allow more specific datasets to be created.

4.4.2 Visualisation Interface

The visualisation interface will let the user interact with the visualised tree. The user can specify a minimum edge weight to trim the tree so as to reveal clusters. The user also has the option to drag nodes around which will move all of their connected neighbours.

4.5 Coding Languages Used

4.5.1 Python2.7

Python2.7 was used to create this project. The CSD has an inbuilt Python API which only works with Python2.7. Python is a useful coding language as it is quite flexible.

Furthermore, the syntax and conventions it uses are forgiving to novice users. For these reasons the language has been adopted by the the scientific non-Computer Science community. The usage of Python is ubiquitous, with applications created for a wide variety of areas from cheminformatics to machine learning.

One of the understated features of python is the import module feature. This feature is one of the reasons a community has been fostered around the language. It allows users to easily utilise multiple third party packages.

4.5.2 JavaScript

JavaScript is a scripting language used for front-end web design. While python has many advantages its visualisation features are limited at best. JavaScript however is primarily designed to create aesthetically pleasing user interfaces. While it is used for websites no internet connection is required for to work effectively javascript. Hence, a website can be stored locally and maintain full functionality.

To transfer data from python to javascript the JavaScript Object Notation (JSON) will be used. This will treat the python as a database or "back-end" for the javascript. JSON is already an implemented feature of both python2.7 and javascript so no extra dependencies will be required to support this operation.

Chapter 5

Realisation

5.1 Implementation of the Design

5.1.1 Extracting Desired Entries from the User Input

Initially the program's code to compute will take two inputs from the user. For the first, a Crystallographic Information File (.cif) will be given by the user. This will be extracted from the CSDs program ConQuest. Then the formula of a molecule will be input. The second input must be present in every entry that is contained in the first input. The second input will be used to verify if each of the entries in the first input contains a common molecule.

The user will then be presented with two filtering options. First, the user will decide which types of crystals will be considered: organic, organometallic, metallic, or a combination. Then, the user will decide the minimum and maximum number of components an entry included can have. A components is defined as the number of molecules present in the crystal structure. The program will search through every entry in the CSD and store an entry which is in the first input and fits the search criteria into an array.

The array's purpose it to hold a list of entries relevant to the user's search for later steps. This array is sorted into alphabetical order. As the CSD can contain duplicates of entries the program ignores entries whose names are greater than 6 characters. This is because every entry has 6 characters and duplicates contain the same name with a number appended to signify that it is a duplicate. The code describing these method's can be seen in Fig. 5.8

5.1.2 Creating the Similarity Matrix

The dataset of entries is then used to create a matrix. This matrix will store the similarity measures which are essential for the visualisation step. The length of the dataset is stored in the variable numberOfEntries. This is then used to create the matrix of size numberOfEntries $\times$ numberOfEntries. Hence, each row and column in the matrix corresponds to an entry. The element stored when the row index is equal to the column index is the entry which every other entry is compared to for that row and that column.

The first entry in the array is then taken and compared to every other entry in the array using the CSDs inbuilt comparison algorithm. A counter is used to track the number of times these comparisons are made. This is for the two reasons. First,

```
def readIdentifiers(self, filename, inputSet):
    identifiers = open(filename, 'r')
    search.Search.Settings.only_organic = True
    commonMolecule = 'c60'
    csd_reader = io.EntryReader('CSD')
    csd_mol_reader = io.MoleculeReader('CSD')
    for line in identifiers:
        if '_database_code_CSD' in line:
            lineEntry = line.split()
            entryStored = lineEntry[1]
            findEntry = csd_reader.entry(entryStored)
            findMol = csd_mol_reader.molecule(entryStored)
            if (findEntry.is_organic == True and findMol.components >= 2
            and len(entryStored) == 6) and if (commonMolecule in
            entryReader.formula == True):
                inputSet.add(str(entryStored))
    identifiers.close()
```

FIGURE 5.1: *Code to extract entries*

the program uses a while loop operator to read through the array and run the comparison algorithm. The counter ensures every entry is read and the loop terminates as soon as possible. Second, by keeping track of the number of iterations the dataset can be made smaller. By filling in the row and column of each entry with a similarity value, the entry which has just been compared to the entire set can be removed as all of its comparisons have already been computed. This is ideal as the this operation can be quite computationally intensive on large datasets. The counter enables this as the index to begin writing subsequent values in the matrix is updated using the counter. This prevents later iterations from filling rows and overwriting the data stored by previous iterations in the columns.

```
def findSimilarity(self, currentEntry, comparisonList, threshold_setting,
hits):
    csd_mol_reader = io.MoleculeReader('CSD')
    sim_mol = csd_mol_reader.molecule(currentEntry)
    sim_query = search.SimilaritySearch(sim_mol)
    sim_query.threshold = threshold_setting
    for j in comparisonList:
        compareMol = csd_mol_reader.molecule(j)
        hit = sim_query.search_molecule(compareMol)
        hits.append(hit)
    collected = gc.collect()
    return hits
```

FIGURE 5.2: *Code to find the similarity values*

5.1.3 Creating the Minimum Spanning Tree

To create the minimum spanning tree the python module SciKit-Learn was used. This module includes Kruskal's algorithm which treats the the matrix as a graph. Each entry is treated as a node and each similarity measure is treated as an edge. The algorithm removes all but the minimum number of edges needed to create a

```
while identifierMatrixPosition < len(firstInput):
    start_time = datetime.now()
    similarityList = []
    entryRead = firstInputList[0]
    threshold_setting = 0
    item_hits = []
    simFunctions.findSimilarity(entryRead, firstInputList,
    threshold_setting, item_hits)

    for hit in range(0, len(item_hits)):
        identifierMatrix[identifierMatrixPosition][hit+identifierMatrixPosit
        ion] = item_hits[hit].similarity
        identifierMatrix[hit+identifierMatrixPosition][identifierMatrixPosit
        ion] = item_hits[hit].similarity
        similarityList.append(item_hits[hit].similarity)
    similarityList.sort()
    pointer = len(similarityList)
    if pointer == 0:
        median = 0
    elif pointer % 2 == 0:
        int1 = similarityList[pointer/2]
        int2 = similarityList[(pointer/2)-1]
        median = (int1 + int2)/2
    else:
        index = ((float(pointer)/2)-1) + 0.5
        median = similarityList[int(index)]

    firstInputList.remove(entryRead)
    print identifierMatrixPosition
```

FIGURE 5.3: *Code to fill the matrix*

path between two nodes.

As the similarity for the CSD is measured from 1 to 0, where values closer to 1 indicate higher similarity, the similarity matrix must be inverted. If this is not done then the minimum spanning tree will produce a tree connecting entries to the least similar entry in the dataset. To do this another matrix is made and every value from the original matrix is inverted. This is done by taking every value away from 1 and storing it at the same index in the new matrix. This can be seen in 5.4
Now a minimum spanning tree can be created. Kruskal's algorithm is applied and

```
while row < MatrixLength:
    for column in range(0, MatrixLength):
        simValue = float(matrix[row][column])
        #print simValue
        MSTMatrix[row][column] = 1-simValue
    row += 1
```

FIGURE 5.4: *Code to find the similarity values*

the output is stored in a new matrix. In this matrix, the first column contains an entry identifier, the second column contains an entry identifier, and the third column contains similarity weight for the degree connecting the two entries. The index of the row is arbitrary. This matrix is then written into a .txt file. The code to create this can be seen in

```
def findSimilarity(self, currentEntry, comparisonList, threshold_setting,
hits):
    csd_mol_reader = io.MoleculeReader('CSD')
    sim_mol = csd_mol_reader.molecule(currentEntry)
    sim_query = search.SimilaritySearch(sim_mol)
    sim_query.threshold = threshold_setting
    for j in comparisonList:
        compareMol = csd_mol_reader.molecule(j)
        hit = sim_query.search_molecule(compareMol)
        hits.append(hit)
    collected = gc.collect()
    return hits
```

FIGURE 5.5: *Code to find the similarity values*

5.1.4 Visualising the Minimum Spanning Tree

The matrix is then exported to a .txt which must be copied over into the matrix used for visualisation. This enables initial visualisation to occur without user participation. This is not only reduces the possibility of human error effecting the visualisation but also creates a smoother the user experience.

The user will need to copy and paste the information in the .txt file into the matrix in the javascript. A matrix to store the values to draw and two arrays are also created. The arrays are used to store nodes and edges used by the graph. As each row in the matrix contains two entries the program checks if either entry exists in the node array. If an entry does not exist then it is added, if it does then it is ignored. Then the nodes corresponding the entries in the row are connected by an edge, the similarity measure contained in each row of the matrix is the weight of this edge. Once this is finished then the graph is drawn for the user to see.

The initial graph visualised will be a single tree. The user will have the option to specify a minimum weight for edges to be drawn. Once the user has verified the desired minimum weight for drawn edges the program reads through the matrix file again. As the edge weight was inverted when creating the minimum spanning tree the minimum weight value must be inverted in the same manner. The value is taken away from 1. If a row in the matrix file contains an edge weight which is greater than the minimum desired weight then it is not considered. If it the edge weight is less than the minimum desired weight then the information is stored for drawing in the same manner as described earlier. This data is then visualised, if the user wishes

to alter the minimum weight of an edge then they have the option to.

```
function setWeight(){
  var weightedArray = [
  var x = document.getElementById("weightCoefficient").value;
  var minSimilarity = 1 - x
  filterArrayByWeight(minSimilarity, weightedArray, x);
}
```

FIGURE 5.6: *Code to obtain the minimum weight set by the user*

```
function filterArrayByWeight(weight, weightArray, spread) {
  var row;
  var weightIndex = spread;
  var filteredWeightedArray = []
  for (row = 0; row < weightArray.length; row++) {
    if (weightArray[row][2] <= weight) {
      filteredWeightedArray.push(weightArray[row]);
    }
  }
  var x = filteredWeightedArray.length;
  document.getElementById("test").innerHTML = x;
  createEdgesAndNodes(filteredWeightedArray, weightIndex)
}
```

FIGURE 5.7: *Code to filter the data*

5.2 Problems encountered during implementation

5.2.1 Usage of python2.7 by the CSD

The CSD only works with python2.7. The developers of the language announced in 2014 that they would cease support and encouraged users to move to python3 as soon as possible[11]. This poses a problem as most cheminformatic modules for python are only compatible with python3. This limited what tools this project could use.

```
function createEdgesAndNodes(filteredWeightedArray, weight){
  var nodesToAdd = new vis.DataSet();
  var edgesToAdd = new vis.DataSet();
  var edgeCount = 1;
  var weightIndex = weight;
  var nodes = []
  var row;
  for (row = 0; row < filteredWeightedArray.length; row++) {
    var node1 = filteredWeightedArray[row][0];
    var node2 = filteredWeightedArray[row][1];

    if (nodes.indexOf(node1) == -1) {
      var node1String = "Node "
      node1String += node1
      nodesToAdd.add([
          {id: node1, label: node1String},
      ]);
      nodes.push(node1)
    }
    if (nodes.indexOf(node2) == -1) {
      var node2String = "Node "
      node2String += node2
      nodesToAdd.add([
          {id: node2, label: node2String},
      ]);
      nodes.push(node2)
    }
```

FIGURE 5.8: *Code to draw the data*

5.2.2 The optimal way of extracting entries

When the initial stage of the program was being implemented multiple methods were considered. One method explored was to use simplified molecular-input line-entry system (SMILES) codes. This a form of line notation which describes the structure of a chemical species. While this method would have provided the user with the ability to create more specific queries not all entries have a smiles string included. Furthermore, SMILES strings will not necessarily look alike if they represent the same thing. While there are some cheminformatics modules for python which can standardise SMILES strings, for example RDKit, all of these are only compatible with python3. Hence, while a SMILES string search functionality would be ideal the design of the CSD does not allow it to be used.

5.2.3 Limitations of Python for Visualisation

While there are many positives to using python it is limited in its use of visualisation. The project was originally advertised as only to be written in python. However, after experimenting with the modules such as NetworkX and GraphVis it was concluded that python couldn't provide the necessary interactive visualisation desired. This is why javascript was adopted during the design stage.

5.3 Changes made to the design during implementation

5.3.1 Alteration of user input

Due to time issues the user input features were not fully implemented. The user can still specify molecules but there is no command line input. The visualisation also works but the JSON feature was not implemented. This was due to time constraints

Chapter 6

Evaluation

6.1 Evaluation of the System

6.1.1 Criteria used to the success of the system

Two criteria will be used to evaluate the success of the system. First, does the functionality implemented compile. This is essential as the system is intended for users who are not masters of python or javascript. Hence, the user must not be required to debug the software at all. Second, does the system work on a test case. The test case will use the molecule C60 to extract organic crystal structures containing at least two molecules.

6.1.2 Test case: C60

This test case was provided by the supervisors of the project.

Input and creating a dataset

For this test case a .cif file was extracted from ConQuest. Every entry included in this file had C60 in the chemical formula. This was then input into the program and C60 was specified as the common molecule. The criteria for each entry was that they had to have at least two molecules and the entry had to be organic. With these constraints applied, a set of 1470 organic cocrystals containing C60 was produced.

The similarity algorithm, see algorithm 1, was then successfully applied to this dataset of 1470 entries. One issue which was discovered from this test is that the algorithm took a significant amount of time. Initially an iteration would take two minutes, this did gradually decrease at a linear rate but the program took two days to run through the entire set. The length of time taken is a result of the algorithm. This could potentially be reduced by a differet choice of algorithm.

Once the similarity matrix was created Kruskal's algorithm was applied, see algorithm 3. Thus a the data can be represented as a graph with $n = 1470$ nodes. For this graph to be a minimum spanning tree, exactly $n - 1 = 1469$ edges is required. As the results match this criteria, it can be concluded that the algorithm achieved the intended goal.

Visualisation

The minimum spanning tree was successfully visualised by the javascript for the webpage. The minimum similarity measure allowed the tree to be reduced into clusters. One issue that was noticed with this area of the system is that the visualisation isn't exactly clear. This is because the edges of the tree have tendency to overlap. However, the user has the ability drag nodes around the screen to make the image clearer. In Fig. 6.1 the visualisation interface is shown when a minimum similarity of 0.85 has been applied. This is considered similar by the Tanimoto coefficient.

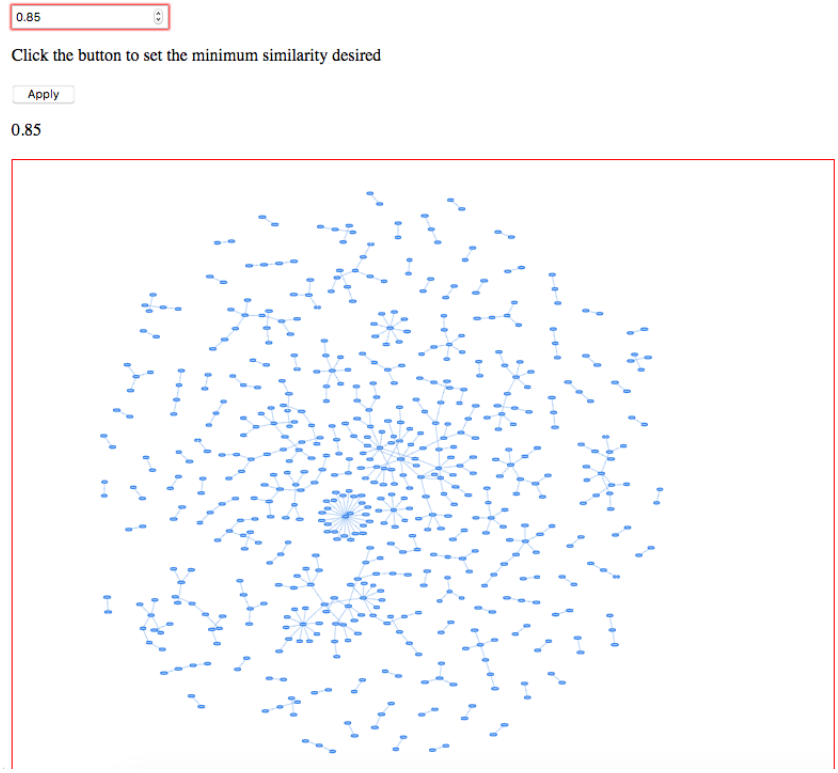


FIGURE 6.1: [C60 Results]

Prediction

For the prediction stage it was concluded that different machine learning techniques would be required. The information the user obtains from the visualisation can be used for such techniques. Currently, the information from the clusters has been used in conjunction with the CSDs descriptors to look at the entry, molecular, and crystallographic properties of each entry in a cluster.

6.2 Strengths and Weaknesses

6.2.1 Strengths

This project demonstrated that the CSD can be used as the basis for a predictive modelling tool. The program has successfully extracted and used entries with a common molecule. This was done through using methods already present within the CSD. It

FIGURE 6.2: [C60 further results]

Refcode	Formula	Colour	Temperature
MIHSON	C60 H52 Ca1 N12 O2,5(C7 H8)	colourless	at 133 K
EHETES	C60 H93 N3 Si3	orange	at 80 K
WEPWIZ	C60 H56 B2 Ca1 N12	colourless	at 123 K
XOWDUI	C60 F36,1.75(C7 H8)	yellow	at 173 K
YELKIK	C60 H42 N6 O18,6(C1 H1 Cl3)	yellow	at 113 K
NADHIL	C60 H70 N6 O10,2(C2 H3 N1),H2 O1	orange	at 150 K
IRUPOB	C60 H134 Si14 Te3,0.5(C6 H14)	orange-red	at 103 K
RUCYIZ	C60 H52 O4 P2 Se2,4(C3 H6 O1)	colourless	at 110 K

TABLE 6.1: [Results: Entry descriptors]

Refcode	Molecular Weight
MIHSON	1473.9005
EHETES	940.6533
WEPWIZ	1006.8648
XOWDUI	1496.7225
YELKIK	1731.8912
NADHIL	1135.348
IRUPOB	1717.8724
RUCYIZ	1289.2336

TABLE 6.2: [Results: Molecular descriptors]

has also been shown that each entry has a breadth data which can be used for analysis by the user, see Table 6.1, 6.2, 6.3.

A major strength of this project is that the data has been visualised in a manner which is interactive and useful for the user. The inclusion of a minimum similarity measure removes unnecessary edges which connect dissimilar entries. As the user can view the actual entry names of each node they can use this information for their own work, see 6.1.

Machine learning techniques have also been successfully applied. It has been shown that clusters can be made of highly similar entries. This allows these groups

Refcode	Calculated Density	Packing Coefficient
MIHSON	1.20478288084	0.66635027975
EHETES	1.06794557255	0.691944927995
WEPWIZ	2.55424069453	0.677088522024
XOWDUI	1.88285093995	0.797342589296
YELKIK	1.49680715157	0.718594441973
NADHIL	1.21964650498	0.676000966626
IRUPOB	1.30943170081	0.65996074066
RUCYIZ	1.35056043302	0.69614376281

TABLE 6.3: [Results: Crystal descriptors]

to be studied and constitutes the first step towards explaining what more sophisticated methods could achieve. Furthermore, it demonstrated that the CSD lends itself to such techniques.

6.2.2 Weaknesses

The final output returned to the user is a minor weakness of this project. This project has attempted to show the user the probability of molecules successfully forming cocrystals. However, due the methods used this has not been fully achieved. The user will receive information about which entries are highly similar. This information can be used for predictions. To take full advantage of this information more sophisticated tools and research wold be required for this.

The time required to process a large set of data is a weakness of this project. While the construction of the similarity matrix gets quicker with every iteration the process can take an uncomfortable amount of time. The test case of C60 shows this as the similarity matrix takes two days to be constructed. This issue cannot be avoided as the similarity algorithm used is inbuilt into the CSD, as the CSD is incompatible with most major cheminformatics kits this algorithm is the best choice available. Thus this is a weakness related to the CSD, hopefully it will soon be updated to be compatible with python3.

Chapter 7

Learning Points

7.1 The CSD can be used as the basis of a predictive tool

This project has successfully shown that machine learning methods can be applied to the CSD. The CSD's data has been successfully extracted and clustering has been applied to group highly similar entries. Without the inbuilt similarity method it would have been significantly harder to create these clusters as more external modules may have been required. If this was not possible a similarity algorithm would have been required to be made from scratch. Such a problem would require it's own project.

7.2 Machine Learning Techniques can be applied to the CSD

The data contained within the CSD can now be interpreted in a new way. The program has attempted to make predictions about which molecules successfully form cocrystals but more complex techniques must be employed. The natural step forward from this project would be to apply extremely randomised trees (ERT) and random forests (RF). A study published in 2017 explored different machine learning methods to predict the outcomes of organic reactions between molecules [21]. The paper found that the most successful methods were RFs and ERTs. As this project successfully clustered highly similar entries using MSTs the data could be used for regression testing to calculate the probability of molecules successfully forming cocrystals.

7.3 Other Machine Learning Techniques must be Explored

This project has shown that the CSD can be successfully used as the basis of a predictive modelling tool. Therefore, it is possible that a convulsion neural network could be applied to the CSD. As previously stated, the CSDs depth of data indicates its worth. This data could be utilised for a neural network to not only calculate the probability of molecules successfully forming cocrystals but also provide an in depth analysis of the crystal structure as well as predict the potential properties. This would provide chemists with the necessary information to apply DFT and show them the utility applying DFT to predictions.

Chapter 8

Professional Issues

For this project the customers were computational chemists. Active communication was maintained to ensure a healthy relationship was sustained. This was crucial as implemented parts of the system could be refined when certain customer requests were misinterpreted. Frequent discussions also enabled future steps to be devised. Areas which were unfamiliar to the developers, specifically chemistry and cheminformatics, were explained by the customers to ensure the correct approach was implemented.
When research was performed for this project, the benefits were assessed. This project will prove beneficial to materials scientists working at the Materials Innovation Factor at the University of Liverpool. The results of this project are in no way negative. It assess the application of machine learning techniques to discover new materials. While conclusions have been drawn the project does not

As the software uses a variety of dependencies this project had to ensure there were no compatibility issues. The dependencies used were combined using the anaconda package. This package manager is recommended by the CSD on installation and ensures compatibility between dependencies. The usage of cheminformatics kits was avoided they all had been written for python3.

Bibliography

[1] 2018.

[2] Nicholas Blagden et al. "Crystal engineering of active pharmaceutical ingredients to improve solubility and dissolution rates". In: *Advanced drug delivery reviews* 59.7 (2007), pp. 617–630.

[3] I David Brown and Brian McMahon. "CIF: the computer language of crystallography". In: *Acta Crystallographica Section B: Structural Science* 58.3 (2002), pp. 317–324.

[4] Cambridge Crystallographic Data Centre. *ConQuest*. URL: https://www.ccdc.cam.ac.uk/solutions/csd-system/components/conquest/.

[5] Cambridge Crystallographic Data Centre. *Quick primer to using the CSD Python API*. URL: https://downloads.ccdc.cam.ac.uk/documentation/API/descriptive_docs/primer.html.

[6] Cambridge Crystallographic Data Centre. *The Cambridge Structural Database (CSD)*. URL: https://www.ccdc.cam.ac.uk/solutions/csd-system/components/csd/.

[7] Reinhard Deistel. *Graph Theory*. Springer, 2016.

[8] Yinhu Deng et al. "Variable structural colouration of composite interphases". In: *Materials Horizons* 4.3 (2017), pp. 389–395.

[9] LOREN BUTLER FEFFER. *J. J. Thomson, the Discovery of the Electron, and the Study of Atomic Structure*. URL: https://www.encyclopedia.com/science/encyclopedias-almanacs-transcripts-and-maps/j-j-thomson-discovery-electron-and-study-atomic-structure.

[10] David A Freedman. *Statistical models: theory and practice*. cambridge university press, 2009.

[11] Sue Gee. "Python 2.7 To Be Maintained Until 2020". In: *IProgrammer* (Apr. 14, 2014). URL: http://www.i-programmer.info/news/216-python/7179-python-27-to-be-maintained-until-2020.html.

[12] Dorian AH Hanaor et al. "Ab initio study of phase stability in doped TiO 2". In: *Computational Mechanics* 50.2 (2012), pp. 185–194.

[13] HTS-powercables. *HTS-powercables*. URL: http://www.hts-powercables.nl/.

[14] 2013 Jogalekar Ashutosh on May 24. "What is chemical intuition?" In: *Scientific American* (May 24, 2013). URL: https://blogs.scientificamerican.com/the-curious-wavefunction/what-is-chemical-intuition/.

[15] Mark A Johnson and Gerald M Maggiora. *Concepts and applications of molecular similarity*. Wiley, 1990.

[16] E. A. Katz. "Fullerene Thin Films as Photovoltaic Material". In: *Nanostructured Materials for Solar Energy Conversion*. Ed. by Tetsuo Soga. Elsevier, 2006. Chap. 13, 361–443.

[17] PRB Kozowyk, GHJ Langejans, and JA Poulis. "Lap shear and impact testing of ochre and beeswax in experimental Middle Stone Age compound adhesives". In: *PloS one* 11.3 (2016), e0150436.

[18] Seth Pettie and Vijaya Ramachandran. "An optimal minimum spanning tree algorithm". In: *Journal of the ACM (JACM)* 49.1 (2002), pp. 16–34.

[19] David J Rogers and Taffee T Tanimoto. "A computer program for classifying plants". In: *Science* 132.3434 (1960), pp. 1115–1118.

[20] Eri Shimono et al. "Logistic regression analysis for the material design of chiral crystals". In: *Chemistry Letters* 47.5 (2018), pp. 611–612.

[21] G Skoraczyński et al. "Predicting the outcomes of organic reactions via machine learning: are current descriptors sufficient?" In: *Scientific reports* 7.1 (2017), p. 3582.

[22] Nicola A Spaldin. "Fundamental Materials Research and the Course of Human Civilization". In: *arXiv preprint arXiv:1708.01325* (2017).

[23] Ian R Thomas et al. "WebCSD: the online portal to the Cambridge Structural Database". In: *Journal of applied crystallography* 43.2 (2010), pp. 362–366.

[24] Robert Wilson. *How much electricity is lost in transmission?* URL: https://carboncounter.wordpress.com/2015/02/15/how-much-electricity-is-lost-in-transmission/.