



A Big Data Approach for Clustering Large Chemical Datasets

JURGEN CASSAR

Supervised by Dr Charlie Abela,
Dr Jean-Paul Ebejer

Department of AI
Faculty of ICT
University of Malta

September, 2018

A dissertation submitted in partial fulfilment of the requirements for the degree of M.Sc. AI.



L-Università
ta' Malta

University of Malta Library – Electronic Thesis & Dissertations (ETD) Repository

The copyright of this thesis/dissertation belongs to the author. The author's rights in respect of this work are as defined by the Copyright Act (Chapter 415) of the Laws of Malta or as modified by any successive legislation.

Users may access this full-text thesis/dissertation and can make use of the information contained in accordance with the Copyright Act provided that the author must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the prior permission of the copyright holder.

Statement of Originality

I, the undersigned, declare that this is my own work unless where otherwise acknowledged and referenced.

Candidate Jurgen Cassar

Signed _____

Date June 22, 2019

Acknowledgements

A number of people have helped me during my life and especially during the period of my dissertation. Special thanks goes to my girlfriend, family and friends for their constant support throughout the duration of this dissertation. Their support ensured that I continued to strive forward even when times were difficult. Also a big thanks goes to my supervisors Dr Charlie Abela and Dr Jean-Paul Ebejer for their constant guidance and help through the period of the dissertation.

Abstract

Physically testing compounds for their biological activity with respect to a target protein is an expensive and time consuming problem in the drug discovery process. Clustering is one of the techniques that enables a more efficient method of selecting compounds for testing. This is done by grouping similar molecules together with the advantage of testing only the compounds from the clusters which contain compounds which exhibit some activity. However, large molecular datasets pose a challenge to efficiently cluster the dataset. Hierarchical clustering techniques are shown to be the most effective in separating active compounds from inactive ones, however the time and space complexity make them impractical for large datasets. Distribution of clustering algorithms may be a possible solution, with Big Data techniques enabling large scale distribution of tasks. In this research, D-Butina a distributed version of Butina clustering algorithm was implemented. The algorithm was extended to create DLSH-Butina algorithm which uses approximation method to identify neighbours. Both implementations obtain satisfactory results, with D-Butina implementation providing increasing speedup of 2.4 and 3.9 when using 5 and 10 distributed nodes, while DLSH-Butina achieves a speedup of 4.1 and 8.4 respectively over the serial approach. Additionally, the clusters achieved by the D-Butina and DLSH-Butina algorithms achieve better separation of actives within the clusters generated than Bisecting k-means.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Aims and Objectives	4
1.3	Proposed Solution	4
1.4	Document Structure	5
2	Background & Literature Overview	7
2.1	Computational Representation of Molecules	7
2.1.1	Graph Representations	7
2.1.2	Connection Tables	8
2.1.3	Linear Notations	9
2.2	Molecular Similarity	10
2.2.1	Molecular Structure	10
2.2.2	One dimensional Molecular Descriptors	11
2.2.3	Two dimensional Molecular Descriptors	12
2.2.4	Three dimensional Molecular Descriptors	14
2.2.5	Similarity Measures	15
2.3	Clustering	16
2.3.1	Non-Hierarchical Clustering	16
2.3.2	Hierarchical Clustering	21
2.3.3	Big Data Clustering	25

2.4	Big Data Paradigm - Map Reduce	28
2.5	Big Data Techniques	29
2.5.1	Hadoop MapReduce	29
2.5.2	Apache Spark	30
2.6	Evaluation Criteria	31
2.6.1	Evaluating Quality of Clusters	31
2.6.2	Evaluating Performance	34
2.7	Related Work	35
2.7.1	Clustering Small Molecules	35
2.7.2	Clustering Large Datasets	37
2.8	Chapter Summary	40
3	Methodology	41
3.1	Decision of Proposed Approach	41
3.1.1	Experiment Setup	42
3.1.2	Scalability Results	43
3.1.3	Clustering Quality Results	45
3.2	Approach Overview	49
3.3	Spark Framework	51
3.4	Datasets	52
3.4.1	DUD-E	52
3.4.2	ZINC	52
3.5	D-Butina	54
3.5.1	Neighbours Identification Phase	54
3.5.2	Clustering Phase	57
3.6	DLSH-Butina	68
3.6.1	Locality Sensitive Hashing	70
3.6.2	Clustering Phase	73
3.7	Chapter Summary	73
4	Results & Evaluation	75
4.1	Experiments Design	75
4.2	Cloud Infrastructure	76

4.3	Evaluation Dataset	77
4.4	Results	78
4.4.1	Selecting the Butina Similarity Threshold	78
4.4.2	Locality Sensitive Hashing (LSH) Parameters	83
4.5	Evaluation	85
4.5.1	Clustering Efficiency	85
4.5.2	Clustering Quality Results	90
4.6	Chapter Summary	94
5	Conclusion	95
5.1	Contributions	96
5.2	Critique and Limitations	97
5.3	Future Work	98
5.4	Final Remarks	100
A	Unknown Activity molecules	101
B	CD Contents	105
C	Installation Instructions	106
C.1	Environment Setup	106
C.2	Running D-Butina	107
C.3	Running DLSH-Butina	107
C.4	Running Output Analysis	108
	Bibliography	109

List of Figures

1.1	Small-molecule binding example	2
2.1	Vanillin molecule	8
2.2	Connection Table for Vanillin molecule.	9
2.3	Similarity calculation of two SMILES strings.	11
2.4	Image showing iterations to create Extended Connectivity Fingerprints.	14
2.5	Classification of clustering algorithms	16
2.6	Exclusion Sphere Example	20
2.7	Dendrogram for hierarchical clustering algorithms.	23
2.8	Example showing Bisecting k-means clustering results.	25
2.9	Example showing the steps of Locality Sensitive Hashing process.	28
2.10	Lineage Graph	31
3.1	Scalability analysis on four serial clustering algorithms.	44
3.2	Comparison of QPI results for the four selected serial clustering algorithms.	47
3.3	Comparison of F-measure results for the four selected serial clustering algorithms.	48
3.4	High level diagram of the presented distributed clustering approaches.	50
3.5	Architecture of the distributed Spark clusters	51
3.6	D-Butina process flow.	55
3.7	Example showing the calculation of the complete similarity matrix distributed among the worker nodes	56

3.8	Example showing the clustering scenario in the Butina approach. . .	59
3.9	Example showing local computation and synchronisation of the clustering approach.	66
3.10	Example showing the clustering scenario in the Butina approach. . .	67
3.11	DLSH-Butina process flow.	69
3.12	Minhash process example.	71
3.13	Locality Sensitive Hashing example	72
3.14	Process flow of neighbour identification process using LSH in a distributed environment.	74
4.1	Parameter estimation using QPI results	79
4.2	Parameter estimation using F-measure results	81
4.3	Speed performance comparison in logarithmic time for five workers	88
4.4	Speed performance comparison in logarithmic time for ten workers .	89

List of Tables

2.1	Example of Fingerprint Representations of SMILES strings.	13
2.2	Review of clustering algorithms applied to clustering small-molecules.	38
4.1	Characteristics of evaluation datasets	78
4.2	F-measure results for Butina on Renin dataset	81
4.3	F-measure results for Butina on THB dataset	82
4.4	F-measure results for Butina on ABL1 dataset	82
4.5	Locality Sensitive Hashing parameter estimation	84
4.6	D-Butina processing time.	86
4.7	DLSH-Butina processing time.	87
4.8	Bisecting k-means processing time.	87
4.9	Quality based comparison of clustering results.	91
A.1	Comparison of unknown activity molecules with activity potential .	104

Introduction

The process of getting a drug to market takes on average 14 years to complete with costs reaching approximately 1.8 billion USD [1, 2]. This affects both pharmaceutical companies with the aim of investing in new drugs and patients in a negative way. One way of improving the process is through the use of Chemoinformatics. Chemoinformatics can be defined as the application of computational methods to the process of drug discovery to aid with gaining knowledge from the information available [3]. Clustering is one of the techniques used, with the aim of grouping large datasets of small-molecules, to enable informed selection of molecules having the properties required to become drugs.

1.1 Motivation

One of the first steps in the drug discovery process is the identification of a subset of small-molecules having the potential of being active with respect to the target protein. A target protein is the compound that is responsible for a disease state [2]. For a drug to function against such disease, a small molecule must bind to the protein and alter its behaviour as shown in Figure 1.1. A small molecule with the necessary properties to bind to a target protein is called an active molecule, with a molecule being only active against a small number of proteins. Conversely, a molecule which does not bind with a target protein is inactive with respect to that protein. Molecular activity is one of the first criteria



University of Malta
L-Universita' ta' Malta

University of Malta Library – Electronic Thesis & Dissertations (ETD) Repository

The copyright of this thesis/dissertation belongs to the author. The author's rights in respect of this work are as defined by the Copyright Act (Chapter 415) of the Laws of Malta or as modified by any successive legislation.

Users may access this full-text thesis/dissertation and can make use of the information contained in accordance with the Copyright Act provided that the author must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the prior permission of the copyright holder.

to determine the application of a molecule on a target, however additional tests must be performed to determine its applicability and viability to become a drug.

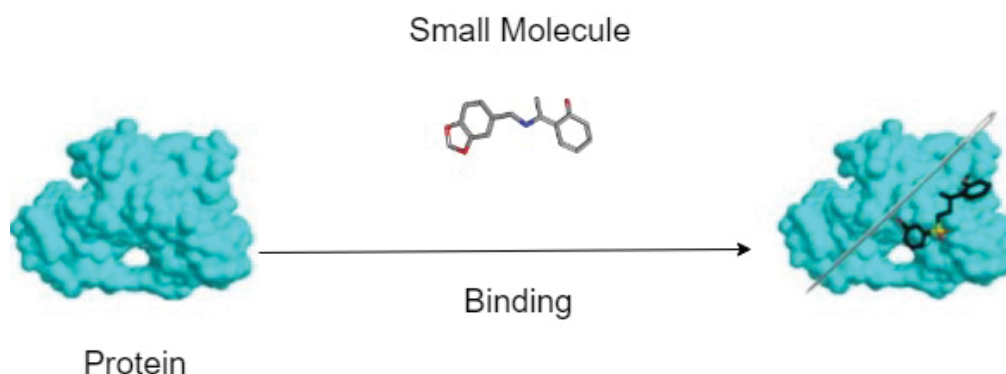


Figure 1.1: An image showing a target protein, a small molecule and binding of the two molecules. Adapted from [4].

The methods of High Throughput Screening (HTS) and Virtual Screening (VS) are two of the methods employed to test for small-molecule activity. HTS is a manual method whereby molecules are manually synthesised and screened. This process is both costly and time consuming. VS is a computational process using computational models to test molecules' binding ability with respect to the target protein. Since it is a computational process, costs are kept lower, however it is still a time consuming process as large number of molecules require testing against each separate target protein. Additionally VS is not as accurate as HTS methods and therefore they may be used together to further enhance accuracy [5].

Not all active compounds end up being marketable drugs. In fact [6] highlighted that the success rate of identifying a drug from an initial number of tested compounds is 0.001%. Therefore a large number of molecules need to be tested for their activity until a successful batch is found. The selection of molecules for testing is an important process as it is done from a larger library of compounds. A random selection of compounds will lead to longer times of activity testing as more molecules are selected than may be required. Additionally, important molecules may be missed altogether. Therefore structured selection may be done to select molecules with a higher chance of achieving the

desired aim, by testing less compounds [7]. Selection should occur depending on the aim of the test. Initial screening happens when no actives are available for a given target protein. It should focus on getting a diverse subset such that the activity of the chemical space is covered. However, should actives be known, more focused approach can be done to select similar molecules to known actives [6].

This structured selection is only possible if the dataset is clustered such that similar molecules are grouped together. Clustering is technique used in data mining with the aim of grouping highly related data points together. It aims to generate groups that have high intra cluster similarity between the points inside the cluster while having a high inter cluster dissimilarity among clusters [8, 9]. This is required to create segregations in data, with similar items being represented in the same cluster, to enable better analysis and faster retrieval of the data having the required properties. It is used in various scenarios such as pattern recognition, drug discovery and community detection to identify new patterns in the data which may not be obvious to the users, thus enabling better predictive analysis on the data [8].

A challenge in this regard is that dataset sizes are increasing at fast rates. The size of the virtual chemical library is around 10^{60} [10, 7]. Although this is not used in its entirety, publicly and privately available datasets reach up to million of compounds and are increasing yearly, therefore making the initial subset selection a difficult one. This amount can be reduced by filtering only those having the correct ADMET (absorption, distribution, metabolism, excretion and toxicity) properties, however this amount is still huge and therefore each small molecule cannot be tested in a brute force manner with respect to every target protein as it is time consuming. Even after years of research, there are molecules that have not yet been tested and could potentially contain solutions for today's not yet cured diseases. The volume of the dataset make the clustering process more difficult as standard algorithms applied in this field do not cater for large datasets [11]. Therefore clustering algorithms require adaption to process the datasets in a distributed manner. This would enable the use of cloud infrastructure to scale the algorithm and handle larger datasets.

1.2 Aims and Objectives

The aim of the research is to apply Big Data paradigms to cluster large small-molecule datasets. The drug discovery process is an expensive and time consuming one, requiring sampling of compounds to identify active compounds with respect to a target protein. Therefore this research aims to investigate and implement a distributed clustering algorithm to increase the efficiency of a clustering method while keeping satisfactory cluster quality of the algorithm.

The following are the objectives of this study:

- Investigate a number of clustering algorithms within Big Data context
- Investigate Big Data techniques to be used as a distribution framework
- Implement a distributed version of the algorithm using the Big Data framework's processing capabilities.
- Apply approximation to the distributed approach to increase the performance of the algorithm

1.3 Proposed Solution

The proposed solution is a clustering algorithm enabling large scale diversification of molecular data by their properties to enable more accurate identification of active molecules. Research is ongoing in separating active compounds from non active ones to enable faster molecular screening. Current advances enable the clustering of small datasets, thus limiting the advantages of such systems. The proposed research will investigate and apply methods in clustering larger datasets, creating a more scalable solution through distributed means using Big Data Technologies. The results will be investigated in relation to the quality of the clusters produced. The proposed solution should be one which maintains a satisfactory level of cluster results while providing better scaling abilities. The extent to which distribution affects the clustering results will then be evaluated with respect to results of research.

An investigation of molecular descriptors will be performed, reviewing the current standards in representing molecules to better discriminate active molecules from non active ones. As found in the research, the choice of descriptors highly affects both the clustering results and the performance of the clustering in terms of speed, therefore correct molecular representation is important to subset the dataset successfully [12, 13]. Multiple descriptor types have been identified, with structural and pharmacophoric fingerprints being extensively used. Combination of fingerprints were also used to represent multiple property groups of the data with the aim of increasing accuracy of the clustering results. Clustering techniques will then be analysed for two groups of properties:

1. The applicability of the algorithm to molecular datasets, producing separation of active from inactive molecules.
2. The algorithm's applicability to the big data context

These properties enable the algorithm to remain relevant as the dataset increases, with algorithm chosen being updated to increase its performance with respect to the identified areas.

1.4 Document Structure

Following the Introduction section, the document is divided into five other sections. Chapter 2 Background & Literature Overview section provides explanation of the information that is required to understand the subsequent parts of the document. This is followed by a review of the literature found, discussing and comparing research previously performed by other researchers while providing a basis for the arguments and decisions taken during the dissertation. Chapter 3, the Methodology section contains an in depth explanation of the proposed solutions, highlighting the architecture, frameworks and datasets used. It also provides explanation of the distributed algorithms implemented. Chapter 4 then presents results obtained throughout the research to provide a basis for decisions taken. It is followed by the evaluation of the approaches created, compared to the serial algorithm and another approach chosen from literature to

enable comparison. In the end, final remarks of the document are done within the Conclusion section, highlighting the limitations of the research and future work that is worth investigating.

Background & Literature Overview

Clustering is based on similarity calculations among the elements to enable the grouping of the molecules together. This chapter details the concepts of how molecules can be represented to enable processing by machines, followed by the various ways molecules can be transformed to enable comparisons and the clustering applications that have been successfully applied in this context. Finally a review of the literature on the applications of such methods and the use of clustering datasets is done.

2.1 Computational Representation of Molecules

The first thing required is to represent molecules in a machine readable manner such that these can be read, stored and processed by computers. Ideally the representation needs to be small, to reduce storage requirements for millions of molecules and enable fast processing.

2.1.1 Graph Representations

One way of representing Chemical Structures is through graphs. A graph is a structure that has been widely used in Computer Science and applied to a number of areas, to represent entities and their relationship. In Chemoinformatics, graphs are used to map atoms as vertices while bonds between atoms are represented as edges. Ring compounds are represented using a cyclic graph while

trees may be used to represent acyclic structures. The topology of the resultant graph is then used as a representation of the chemical structure [14, 5].

2.1.2 Connection Tables

Connection Tables is a way to represent molecules in a tabular format. A table having two sections is created, with the first one using rows to represent the atoms in the chemical structure, while the second one identifies the bonds among the atoms. Vanillin molecule shown in Figure 2.1 is represented in a connection table in Figure 2.2.

For each atom, the connection table includes the coordinates of that atom in 2D or 3D space, depending on the options defined, to allow reconstruction. Additional columns are available for properties pertaining to the atom which may be included depending on the format of the connection table. Most chemical structure datasets provide the data in this format with separate molecules being in mol files, or combinations of mol files provided in sdf format [15].

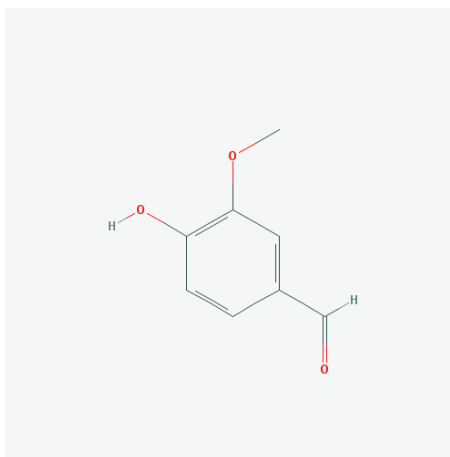


Figure 2.1: Vanillin molecule image from Pubchem¹. This is normally used as a flavouring agent in food.

¹<https://pubchem.ncbi.nlm.nih.gov/compound/vanillin> (last accessed 25th May 2019)

```

OpenBabel106291815203D
19 19 0 0 0 0 0 0 0 0999 v2000
1.4763 -0.2680 -0.6877 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2.4994 0.1233 0.2223 O 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
3.0612 -0.8733 0.9777 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4.0708 -0.4541 1.8542 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4.7299 -1.3443 2.6954 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4.3749 -2.6913 2.6745 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
3.3736 -3.1243 1.7987 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2.7173 -2.2236 0.9471 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
3.0100 -4.5510 1.7846 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2.3207 -5.0302 0.8880 O 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4.4360 0.8638 1.9014 O 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1.1255 0.6320 -1.2017 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.6215 -0.7035 -0.1595 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1.8670 -0.9550 -1.4454 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5.5128 -0.9864 3.3581 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4.8909 -3.3892 3.3289 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1.9447 -2.6036 0.2860 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
3.3773 -5.1710 2.6150 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
3.8638 1.3097 1.2463 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 2 1 0 0 0 0
1 12 1 0 0 0 0
1 13 1 0 0 0 0
1 14 1 0 0 0 0
2 3 1 0 0 0 0
3 4 2 0 0 0 0
3 8 1 0 0 0 0
4 5 1 0 0 0 0
4 11 1 0 0 0 0
5 6 2 0 0 0 0
5 15 1 0 0 0 0
6 7 1 0 0 0 0
6 16 1 0 0 0 0
7 8 2 0 0 0 0
7 9 1 0 0 0 0
8 17 1 0 0 0 0
9 10 2 0 0 0 0
9 18 1 0 0 0 0
11 19 1 0 0 0 0
M END

```

Figure 2.2: Connection Table in mol2² format for the molecule Vanillin generated using OpenBabel software. The first table shows 3D aromatic coordinates of the atoms, with the other properties being 0. The second table shows the bonds between the atoms.

2.1.3 Linear Notations

Linear Notations are a representation of a chemical structure by using a string of alphanumeric characters in a sequence. One standard notation is the Simplified Molecular Input Line Entry Specification (SMILES). It is a compact representation of molecules, using upper case letters for aliphatic atoms and lower case letters

²http://www.csb.yale.edu/userguides/datamanip/dock/DOCK_4.0.1/html/Manual.41.html (last accessed 25th May 2019)

for aromatic atoms. Bonds are represented by having either adjacent atoms, in case of single bonds, or through the use of '=' and '#' symbols for double and triple bonds. Parenthesis enclosures are used to represent branching from one atom into separate paths [16].

An example SMILES string for the molecule Vanillin shown in Figure 2.1 is COC1=C(C=CC(=C1)C=O)O. The construction of the molecule is done by parsing the string and reading each atom only once. Considering its compactness, SMILES format is the ideal method to store and transfer large datasets of chemical structures.

2.2 Molecular Similarity

Molecular similarity provides a score representing the similarity between two molecules, with two identical molecules having a score of 1. This similarity computation is an important step in the clustering process as it enables to identify similar molecules to group together, thus having an important impact on the final clustering result. Molecules are first converted into a format to represent the properties requires, which then is used in a similarity measure to calculate the score of similarity as seen in Figure 2.3. For this to be successful, the correct representation of the data in conjunction with the correct similarity comparison must be selected to convey the required meaning of similarity. Two main molecular representation methods exist for the calculation of molecular similarity. These are graph representation of the molecular structure, with the similarity being calculated using the topological structure or molecular descriptors. Descriptors can be divided into three further groups depending on the data being represented, one dimensional, two dimensional and three dimensional descriptors.

2.2.1 Molecular Structure

All the molecular representations can be converted into molecular graphs for the purpose of similarity comparison. Graphs enable the representation of small molecules without losing information, representing all of its atoms and bonds.

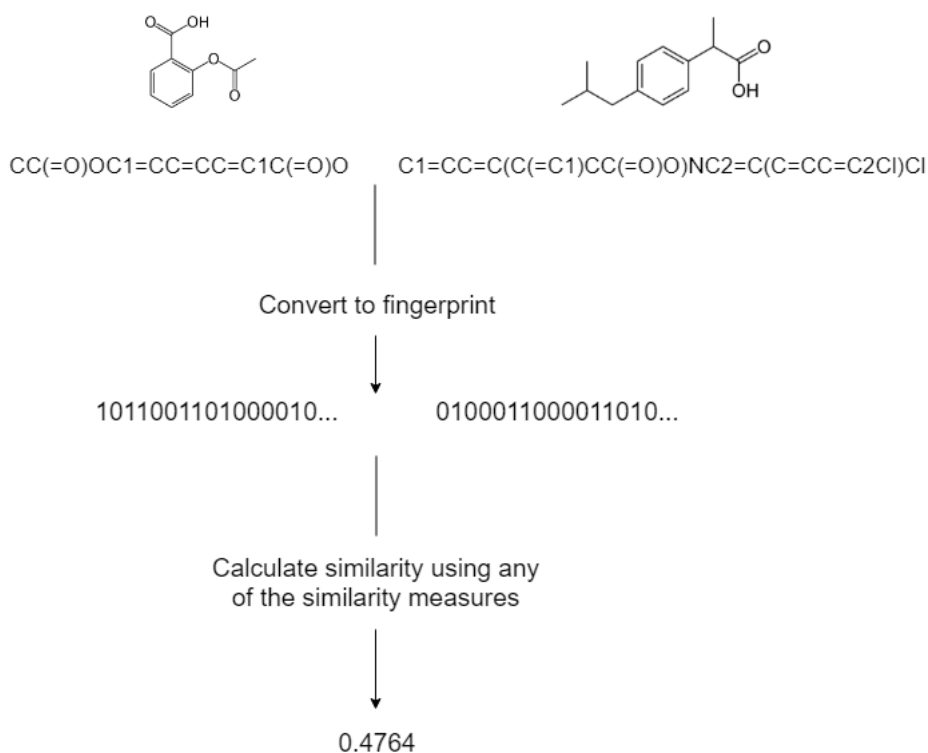


Figure 2.3: A schematic diagram, based on a diagram by [17], of the steps required to calculate the similarity between two molecules starting from their SMILES string.

Molecular similarity is then calculated by matching the sub structure overlap between two molecules by using Maximum Common Substructure (MCS) method. One advantage of such method is that there are no properties that are ignored when computing such similarity, and the original components are being compared. Additionally, the comparison can be visualised in an intuitive manner, making it ideal and easy to understand. MCS is found using graph isomorphism, however such computation is expensive and considered as an NP-complete problem [18]. Therefore it may not be ideal for large datasets [19].

2.2.2 One dimensional Molecular Descriptors

One dimensional molecular descriptors are single values calculated from the molecular structure to represent one of the properties of the molecule. These can

be simple counts, representing elements from the structure of the molecule such as number of hydrogen bond donors, hydrogen bond acceptors, aromatic rings, rotatable bonds or molecular weight. Other descriptors may be more complex to calculate however they offer more value as a representation of the molecule. One such value is the hydrophobicity represented by $\log P$. This represents how much a small molecule is repelled from water. It can be used as one of the properties to indicate the molecule's binding chances to a protein, and is a good indicator of biological activity. Many different descriptors were proposed, each highlighting different properties [5].

Since similarity is subjective, the selection of the descriptors is determined by the user and the aim of the similarity calculation. When changing the descriptor used, the similarity between two molecules may drastically change. A similarity between two molecules based on the number of atoms is different from a similarity based on hydrophobicity. Additionally, one such value by itself is most of the times useless in offering enough discrimination between different molecules. Therefore combination of descriptors are normally used to get averaged results in similarities [20, 21]

2.2.3 Two dimensional Molecular Descriptors

Two dimensional descriptors encode data from the 2D structure of a molecule. These can be either a single values calculated from a substructure or properties of the molecule, or a longer representation encoding more information.

Two types of string based descriptors are the Atom pairs and Topological Torsions. Atom pairs encode the shortest path between all pairs of atoms in a molecule. Similarly Topological Torsion encode sequences of components connected together. The result is a string representing the paths and the atoms found while traversing it [5].

Fingerprints are a type of descriptor that are commonly used to represent combinations of 2D molecular data. Fingerprints take the form of either bit-strings or count strings with the aim of encoding more data and provide a better representation of the molecular properties as seen in Table 2.1. Two types of 2D fingerprints exist, fragment dictionary and hashed fingerprints. Fragment dic-

	Smiles String	Fingerprint
Mol1	C11H12N2O2	001011001100010001100000
Mol2	C1CCCCC1	001011001010000000100010

Table 2.1: Example of Fingerprint Representations of SMILES strings.

tionary fingerprints represented as bitstrings are created in a way such that each bit position represents the presence or absence of a structure fragment from the molecule, with count based fingerprints represent the number of instances of each structure. Hashed fingerprints represent molecule substructures and properties in a hashed manner making an interpretation of the results more difficult. These are normally used in cases where the possible combinations of properties to represent are very large and therefore it is impossible to assign a unique bit for each pattern. Therefore each combination serves as an input to a hashing function that returns a bit pattern of 4 or 5 bits [17].

Fingerprints can be of varying length, although normally they are found to be either 1024 or 2048 bits. The flexibility of fingerprints enable them to encode variety of formats. In fact, even atom pairs and topological torsions can be themselves represented as part of a fingerprint, with each bit position representing the presence of an atom pair or topological torsion combination. Chemically Advanced Template Search (CATS) are one such type of descriptor representing this data [5].

Extended Connectivity Fingerprints are another type of 2D fingerprint, encoding circular substructure data by representing atoms using their neighbouring atoms in form of a radius. A variant of Morgan algorithm is used to calculate the extended connectivity of atoms, with the ability to vary the number of bonds or radius extension to use [22]. Extended Connectivity Fingerprints were created with the intent of representing molecular biological activity, with two popular fingerprints in this group being Extended Connectivity Fingerprints (ECFP) and Functional Connectivity Fingerprints (FCFP) [10, 23].

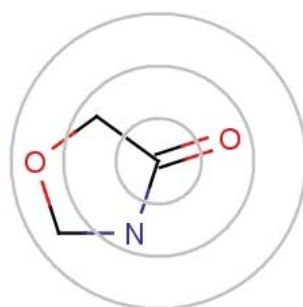


Figure 2.4: An example of how Extended Connectivity Fingerprints are created, with each circle representing the neighbours considered in each iteration to generate the fingerprint [22]. Image from Chemaxon³.

Daylight⁴ fingerprints are another widely used type of hashed fingerprints. A pattern is created for each atom, atoms and their bonds to neighbours and patterns for groups of atoms and bonds. The patterns are then used to generate a fixed hash for every input pattern that are then combined [17].

2.2.4 Three dimensional Molecular Descriptors

3D Descriptors are fingerprints that are generated from the three dimensional molecular structure. Fingerprints encoding 3D data are also used, representing spatial features and relationships of elements in molecules. These encode data such as distances and angles between atoms, position of rings and planes. A number of 3D descriptors exists, however one drawback with respect to the 2D counterparts is that 3D descriptors are more time consuming to generate and therefore they are less ideal to use for large datasets [5].

Pharmacophoric keys are a type of 3D fingerprints that encode molecular properties such as steric and electronic features. These are properties of molecules that are thought to be important for molecular binding with target proteins. These have been applied both in Virtual Screening (refer to Introduction Section 1.1) and molecular docking scenarios to try and predict small molecule binding [24].

³<https://docs.chemaxon.com/display/docs/Extended+Connectivity+Fingerprint+ECFP> (last accessed 25th May 2019)

⁴<http://www.daylight.com/dayhtml/doc/theory/theory.finger.html> (last accessed 25th May 2019)

2.2.5 Similarity Measures

Once the required descriptor is selected, a way of calculating similarity between them is required. This is done by using one of the similarity measures available. The most commonly measures used are presented in Equations 2.2-2.4, showing Euclidean Distance, Cosine Similarity and Hamming Distance. The variables p_i and q_i represent element at position i of the two descriptor vectors that are being compared [11, 10]. However, the most popular and most widely used metric is the Tanimoto Coefficient (also known as Jaccard coefficient) as shown in Equation 2.1, with a representing the number of 1 bits in fingerprint 1, b representing the number on 1 bits in fingerprint 2 and c representing the number of common 1 bits between fingerprints 1 and 2. This stems from the fact that based on the theory by [25], molecules having similar properties also tend to have similar activity, therefore similarity based on the overlap of properties provides the most reliable measures for activity prediction. This is infact what Tanimoto Coefficient does, which measures similarity by dividing the amount of similar properties (the intersection) by the total number of properties of both entities (the union) [7, 26].

$$\text{TanimotoCoefficient} = \frac{c}{a + b - c} \quad (2.1)$$

$$\text{EuclideanDistance} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2} \quad (2.2)$$

$$\text{CosineSimilarity} = \frac{\sum_{i=1}^n (p_i q_i)}{\sqrt{\sum_{i=1}^n p_i^2 \sum_{i=1}^n q_i^2}} \quad (2.3)$$

$$\text{HammingDistance} = \sum_{i=1}^n |p_i - q_i| \quad (2.4)$$

2.3 Clustering

Clustering is a term used to represent unsupervised learning techniques that group items based on their similarity with each other. When grouping elements, the aim is that of maximising intra cluster similarity while minimising inter cluster similarity, with the result being diverse groups with each group having components as similar as possible. Clustering is normally done at initial stages of the process to understand the data and provide information to proceed forward. Clustering algorithms are divided in two main categories, non-hierarchical and hierarchical clustering techniques with the two groups being further divided into subgroups as shown in Figure 2.5. Both approaches have been successfully applied to cluster small molecules [11].

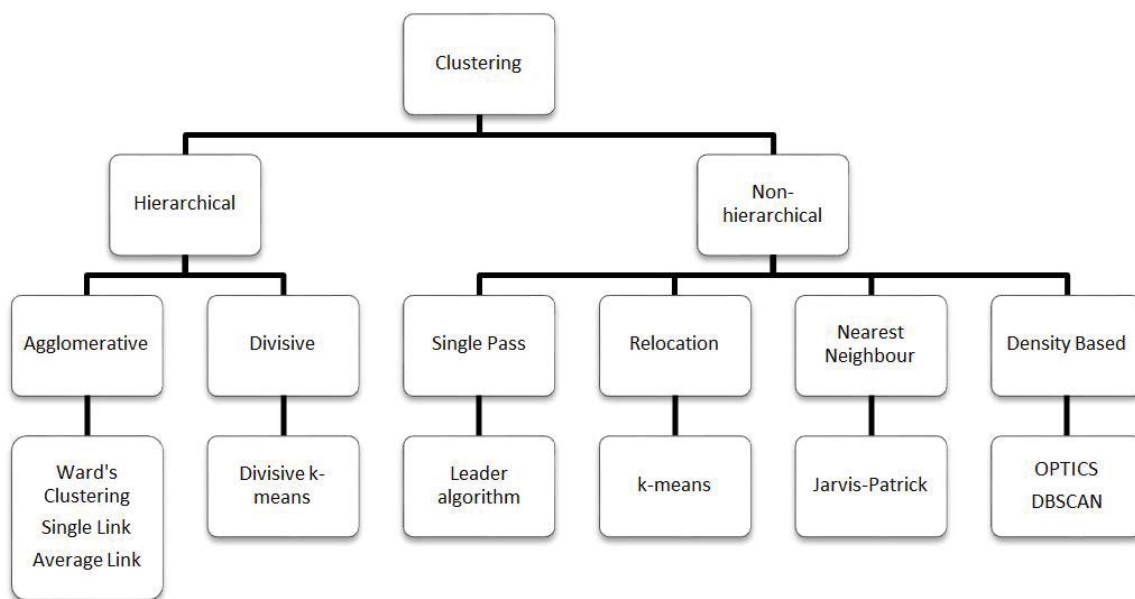


Figure 2.5: Classification of clustering algorithms adapted from [11]

2.3.1 Non-Hierarchical Clustering

Non-hierarchical clustering algorithms are those methods which do not generate a hierarchy of clusters as part of their result. Their output is the partitioned data, with no relations between the partitions. Non-hierarchical techniques are

further divided into multiple groups, determined by the process used to build the clusters, with the most popular ones as presented by [11] being:

- Relocation techniques
- Nearest Neighbours techniques
- Density Based techniques
- Single Pass techniques

Relocation techniques work by moving points within different clusters with the aim of improving an objective criteria. Data points are moved multiple times around clusters until the optimal cluster is found. K-means is the most popular algorithm within this group and has been widely applied to variety of contexts both because of its $O(kn)$ scalability and its performance in identifying the correct clusters, with k being the number of clusters and n the number of data points [11]. Single Pass methods tend to be the most efficient unless some preprocessing step, such as sorting, is required. Single Pass techniques aim to identify the correct cluster by visiting each data point only once, however at times they are undeterministic.

Density based clustering techniques identify the distribution of the data with the aim of finding dense regions to cluster them together. Normally a threshold density is used to add members to clusters. Nearest Neighbours techniques calculate the neighbours of each datapoint and use that information to cluster common neighbours together [27, 28].

Non-hierarchical clustering algorithms have been used to cluster small molecules because of their speed when compared to hierarchical counterparts. The most popular within this group is the Jarvis Patrick algorithm, having obtained successful results [29]. Variants of k-means have also been applied. Exclusion sphere clustering and other algorithms built on its concept have also been used both for clustering and diversity selection, with Butina clustering being one of the main algorithms created for small-molecule clustering [11].

Jarvis Patrick Clustering

Jarvis Patrick is a non-hierarchical clustering technique using the nearest neighbour approach to create non overlapping clusters. Jarvis Patrick works in two phases. In the first phase, the top k neighbours for each compound are identified, with k being a user defined parameter.

In the second phase, clusters are created. This is done by going through the list of nearest neighbours created and for each molecules x and y , y is added to cluster x if it satisfies three conditions.

1. y is in the list of nearest neighbours of x , calculated in phase one
2. x is in the list of nearest neighbours of y , calculated in phase one
3. x and y have k_{min} number of common neighbours, with k_{min} being a parameter chosen by the user whose value cannot be greater than k .

Any two compounds satisfying these conditions are clustered together, thus using the concept of neighbourhood as a basis of clustering [30, 29]. The parameter of k , to determine the number of neighbours calculated, affects the performance of the algorithm in both speed and quality of clusters. Smaller values lead to a faster algorithm, but smaller clusters, while more neighbours lead to slower computation and larger clusters.

This technique has been used in various scenarios in chemical clustering [17, 29], mainly due to the fact of having the ability to scale better than its hierarchical counterparts with better results compared to other non-hierarchical methods. However, when compared to hierarchical methods, the effectiveness of the algorithm was not as good. Jarvis Partick however, still requires $O(N^2)$ time to compute, while requiring $O(N)$ space to store the intermediate results [31].

A major problem as identified by [18] is that the two parameters k and k_{min} need to be changed for every dataset. Therefore no ideal parameter exists for chemical information, but the values need to be found for every dataset used. This may create a problem as an element of trial and error is included in the process, which for large datasets may not be an ideal solution.

Butina Clustering

The Butina clustering algorithm is an adaptation of the Single Pass technique Exclusion Sphere algorithm, with the aim of inducing an order independent step. This ensures that multiple runs of the same algorithm produce consistent results.

Exclusion Sphere algorithm works by having three sets. The first set contains all the unvisited data points, which at first would contain all the initial elements. A second set representing cluster centroids is initially empty and a third set containing the visited points and their assigned cluster.

An initial element is selected at random from the list of unassigned data points. This is assigned to be a cluster centroid and is added to the respective list. Similarity is then calculated to all other unassigned elements in the dataset. Any elements having a similarity equal to or greater than a user defined threshold is assigned to the cluster of the current selected element, added to the set of assigned data points and removed from the unassigned set. This excludes them from further processing [9]. Another element is then selected at random from the unassigned list of molecules to become cluster centroid and is compared to the remaining elements, repeating the same process of assigning similar elements to the current cluster. This is done until all elements have been assigned to a cluster. The result achieved would be non overlapping clusters as can be observed in Figure 2.6.

Although this method is efficient, it has the drawback of being non deterministic [9]. It is dependent on the order of selected cluster centroids. Therefore Butina was created based on the concept of Exclusion Sphere algorithm, with an added preprocessing step to create order in the way centroids are selected.

A stage is added before Exclusion Sphere clustering with the aim of identifying elements that have potential of being the best cluster centroids. Butina argues that molecules having a high number of neighbours are good candidates for cluster centroids as they are similar to a number of other molecules. Therefore as an initial step, all the neighbours for each data point are calculated, using a user defined threshold to define neighbourhood. Once all the neighbours are identified, all the elements are sorted by their number of neighbours in a descending manner, thus having the elements with most neighbours as high

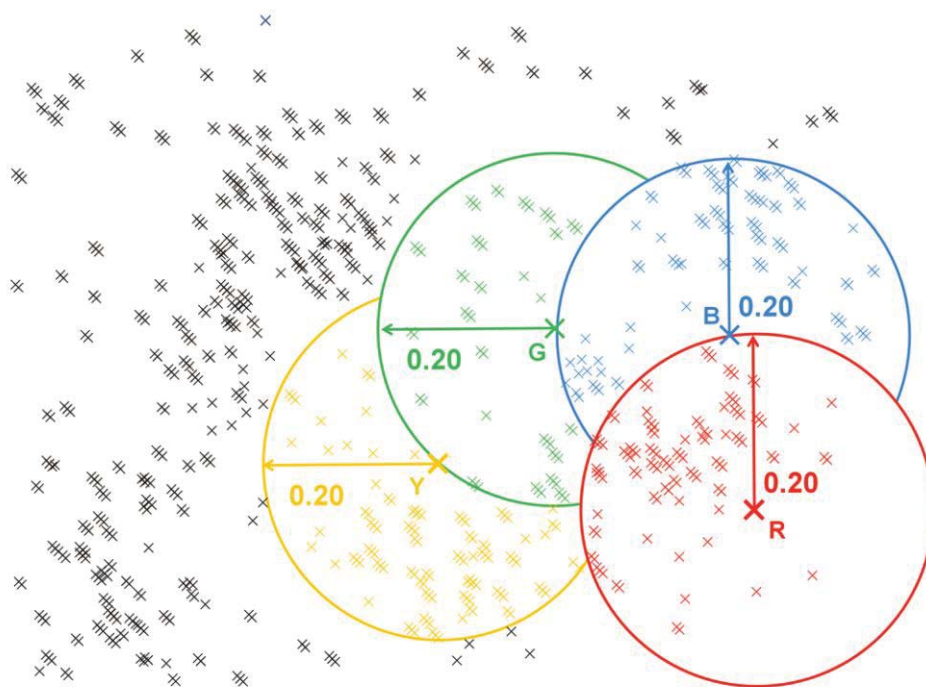


Figure 2.6: Exclusion Sphere Example using 0.2 similarity as represented by [33]

candidates for cluster centres. The second step is Exclusion Sphere clustering, choosing as centroid a not yet assigned datapoint having most neighbours at this stage. The potential members of the cluster are the neighbours previously calculated. However for each new cluster centroid, the neighbours that have already been assigned to other clusters are removed from the current cluster, as they would be already taken by a more strong centroid having more neighbours.

An important observation in this process is that, not all singletons generated are actual singletons with no neighbours at the given range. However, some singletons may actually have neighbours that were already added to clusters of previous centroids, thus ending being singletons even if they are similar to other elements [32].

Butina clustering algorithm guarantees that all centroids are at least equal to or similar up to a threshold to each molecule in the cluster. Therefore it increases the homogeneity within the clusters. Additionally, the threshold similarity is the only input required for the algorithm, thus making it relatively easy to use [17].

K-Means Clustering

K-means clustering is a popular relocation clustering algorithm that is used in a number of fields. It works by randomly selecting a user defined k number of cluster centroids. Each other object in the dataset is assigned to the closest cluster centroid. Once each object is assigned to a cluster, the cluster centroid is recalculated to become a better representation of the cluster. This is done by selecting a new cluster centroid from the cluster that better represents the mean distances of the cluster, with the aim of minimising the distance of each cluster member to the centroid [34, 11]. All the members of the clusters are again assigned to the nearest cluster centroids, which could have changed from the previous ones.

The process continues in iteration until the cluster centroids do not change position or no object changes cluster membership. This approach is efficient, having $O(kn)$ time complexity, with k being the number of clusters and n the number of elements in the dataset. However the drawback of such method is the selection of k and the random selection of the centroids that may affect the final clustering results [11].

2.3.2 Hierarchical Clustering

Hierarchical algorithms are divided into two groups, agglomerative and divisive algorithms. Agglomerative hierarchical methods start with all data points as singleton clusters. The clustering works in iterations, with each iteration combining the two most similar clusters into a single one. A tree relationship between the newly created cluster and the previous separate clusters is then created with the new cluster being represented as a parent of the joined ones. This process is repeated until all clusters are merged into a single cluster containing all the data set [3, 11].

Divisive Hierarchical clustering starts with one cluster containing all the data points, with the aim being that of dividing the cluster rather than combining. At each iteration a cluster chosen by the condition established, is divided into two or more separate clusters, also creating a relationship between the old and new clusters [3, 35, 21].

The most common hierarchical agglomerative clustering algorithms are known as sequential agglomerative hierarchical non-overlapping methods (SAHN), such as Single Link, Group Average Link and Ward's Clustering. These are similar in their process of clustering, with the only difference being the condition to determine which clusters to join at a given step [11]. This can be based on the distance of the centroids between the clusters, the average distance of all the data points or the distance between the nearest or farthest points, as deemed appropriate. Other hierarchical clustering methods such as K-means hierarchical clustering implemented by [35] created a custom way of clustering with the aim of creating a hierarchy of clusters.

Hierarchical clustering is usually implemented using the stored matrix approach, which requires that a matrix of all the pairwise distances is calculated for all the datapoints. At each iteration, the stored matrix is updated to reflect the new clusters, therefore such algorithms have $O(N^3)$ performance and $O(N^2)$ memory requirements. Performance can be improved by using the Reciprocal Nearest Neighbour (RNN) approach, requiring $O(N^2)$ time.

RNN is a way of identifying the clusters to join at any step in an efficient way compared to a whole scan of the similarity matrix. This works by initially setting all points as unused. A random unused point I is selected as a starting point. A chain of nearest neighbours is calculated until a pair of reciprocal nearest neighbours P and Q are found. P and Q are merged in a single cluster, Q is marked as used, while P is updated to be represented by the centroid on the newly created cluster. The nearest neighbour process is continued from the point before P or a new starting point, should P be the first point chosen [30].

As a result of the relationships created by hierarchical algorithms, a dendrogram, as in Figure 2.7 is created, representing all the relationships of the clusters. This is in the form of a tree or inverted tree depending whether the algorithm is agglomerative or divisive. For agglomerative clustering, the topmost layer contains the separate elements while the bottom layer contains the single large cluster. Dendrograms allow for better data visualisation, facilitating the inspection of similar clusters and providing the functionality to drill into more fine grained clusters to obtain more constrained results [11].

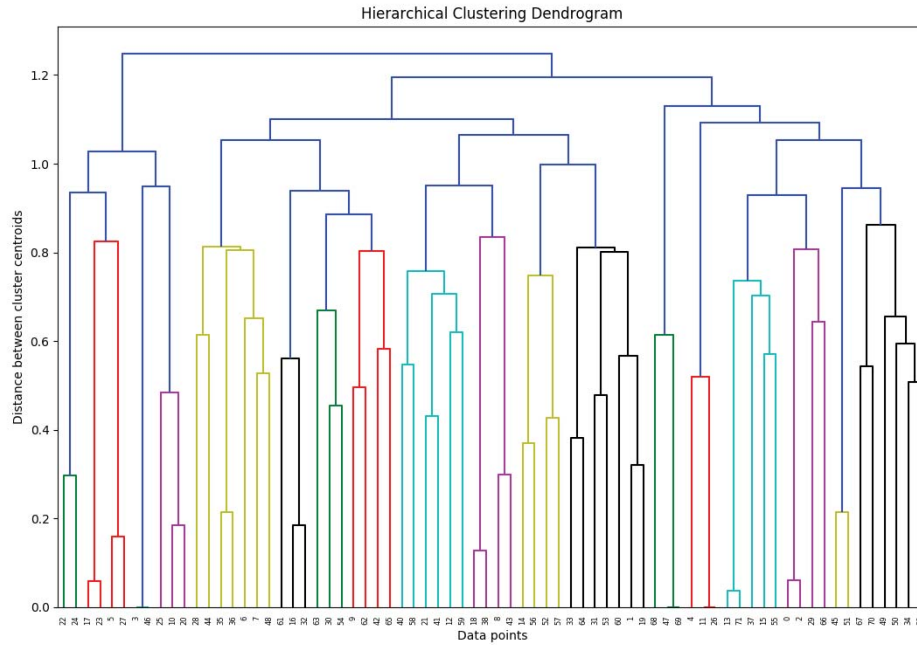


Figure 2.7: Dendrogram for hierarchical clustering algorithms. The y-axis represents the distance between cluster centers at that level, while the x-axis represents the data points being clustered. Adapted from [3].

Ward's Clustering

Ward's Clustering is a hierarchical agglomerative clustering method. The merging condition defined for Ward's clustering is the error sum of squares (or within cluster variance) e^2 , with the optimal value being 0 [36]. This property represents the loss of information incurred when representing the members of a cluster as a whole group and is represented by the equation:

$$ESS(\text{onegroup}) = \sum_{i=1}^n x_i^2 - \frac{1}{n} \left(\sum_{i=1}^n x_i \right)^2 \quad (2.5)$$

where x_i is the score of the data point at position i and n being the number of elements in the dataset. Using this objective function, Ward's clustering aims to create clusters whose data points are close around the center of the cluster, as adding point further from the cluster center would increase the error. At each

iteration, Ward's clustering algorithm checks for the two clusters which when joined, result in having the least error sum of squares thus emphasizing on the reduction of intracluster (within cluster) variance and increase of inter cluster variance.

Ward's clustering has been widely used in clustering chemical compounds, with successful results in separating active from inactive molecules (refer to Introduction Section 1.1). Its results have been consistently better than competing algorithms, thus being considered as the standard algorithm within this context [8]. The drawback of the algorithm is its time and space complexity. Considering its efficiency it tends to be useful only for small datasets, as it would be intractable for larger ones [36, 31, 11].

Bisecting K-means

Bisecting K-means approach is divisive hierarchical clustering method that uses k-means clustering (see Section 2.3.1) algorithm to divide clusters at each hierarchy level. The divided clusters create a parent child relationship with the original cluster, creating the hierarchy dendrogram as shown in Figure 2.7.

Bisecting K-means works by selecting a cluster to divide into two splits. A popular implementation by [37] and is implemented in Apache Spark⁵, divides the selected cluster multiple times into two sub clusters, creating P pairs of sub clusters, of which only one will be selected. The two sub clusters that produce the best results based on the overall similarity among the cluster members are selected. P is a user defined value to determine the number of splits to be done, from which the most optimal split is then selected. This process of selecting a cluster and dividing it into two sub clusters is repeated until k , user defined, number of clusters are obtained. The cluster to split can be chosen based on different criteria. One typical implementation is to select the largest cluster at the current hierarchical level.

Although it has not been widely used in chemical clustering, bisecting k-means was evaluated in comparison to Ward's clustering by [38] showing that it produces more quality clusters than standard K-means and obtaining better

⁵<http://spark.apache.org/> (last accessed 25th May 2019)

scalability than Ward's clustering. [35] also used a similar approach however it included variations in the bisecting process at each iteration and the stopping criteria.

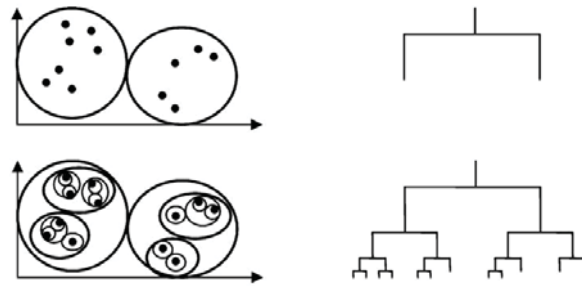


Figure 2.8: Bisecting k-means clustering diagram taken from [35], with left images showing nested clusters, while right images showing the hierarchical structure.

2.3.3 Big Data Clustering

When choosing a clustering algorithm for big data contexts, it must be ensured that it is able to cater for the properties of the data in use, such as the size of the dataset, its changing nature in time or the use of multiple format simultaneously. It may not be practical to employ clustering algorithms used for small datasets directly on big data. This is not feasible as some techniques employ methods that may be inefficient for that context. Some clustering algorithms iterate multiple times over the dataset while others may require to store temporary data in memory, such as similarity matrix, which may be quadratic to the number of elements in size. Other techniques may not be able to handle changing datasets and therefore may not be directly applicable to datasets that change over time [39].

As data grows, issues are encountered, with these becoming more prominent in scenarios of big data. Therefore when selecting or implementing an algorithm for big datasets, some criteria as discussed by [40] and [41], must be considered:

- Scalability: Algorithms should be scalable to ensure that they can cater for

the large datasets, enabling the discovery of the clusters present in reasonable time frames.

- **Stability:** an ideal property would be to provide consistent results over multiple runs. If the algorithm is non deterministic, then non optimal clusters may be returned in different runs, with no indication of how to choose the ideal clusters.
- **Handle Dimensionality of the data:** when having high dimensionality, data becomes sparser therefore the clustering algorithm and similarity measure should handle such data to have relevant clusters. This issue is represented by the idea of the curse of dimensionality, where as dimensions increase the density of the data at any point becomes low.
- **Handle Outliers:** outliers should be handled effectively such as not to negatively affect the resultant clusters.
- **Reduced dependency on input parameters:** the setting of parameters affects the clustering quality and speed performance. Therefore it would be ideal to reduce this dependency, limiting the cases where incorrect parameters are chosen, thus negatively affecting the clustering results.

As the size of the dataset increases, different methods to handle the large datasets are required. Distributed clustering techniques are one way of dealing with this issue. Distributed techniques work by dividing the data and share it among different nodes, performing local computations on subsets of the data, with additional steps to agglomerate the results.

This method enables scaling out, as nodes may be added to the task as required. However care must still be taken when implementing such systems, as communication between nodes is expensive and therefore should be kept to a minimum. Although speed of a clustering algorithms may increase with distribution, these optimisations may lead to a reduced accuracy when compared to serial implementations [42]. This occurs as the algorithm could be altered to cater for reduced communication. The algorithm at each node would not have

full visibility of the dataset and therefore the result would not be globally optimum.

Although clustering should be deterministic, approximation techniques are still used because of their performance enhancement. Approximation methods offer a trade off between speed and accuracy for the clustering produced, and even though not desirable, they are used because of the size of the data at hand. One such technique is Locality Sensitive Hashing (LSH).

Approximation Method - Locality Sensitive Hashing

LSH is a similarity searching method popularly used in contexts of high dimensionality and has been applied to contexts such as document similarity searching and audio files [43]. LSH technique uses hash functions with a high probability of hashing similar items in the same bucket, while dissimilar items in different buckets. Objects in the same bucket are then considered as candidate neighbours and therefore reducing the similarity search space [43, 44]. LSH process for Tanimoto similar objects is divided into two sections, Minhash and LSH.

Minhash is a fast approximate calculation for the Tanimoto distance between two objects. It is used for high dimensionality objects, with the aim of speeding up the performance [44]. Given a fingerprint $F1$, X random permutations for the positions of the fingerprints are generated. The fingerprint bits are ordered according to the permutations produced, each time noting the position of the first '1' bit found. The minhash result is the list of signatures representing '1' bit positions for each permutation.

The process of LSH starts by grouping signatures from minhash into Y bins. Each group is hashed into a single value using a hashing function where similar items are hashed together. Each hash result corresponds to a dictionary key (or bucket), with the values being fingerprints or data objects that hash to that result [46]. The complete process can be seen in Figure 2.9.

To identify similar elements, a new fingerprint is hashed and the buckets it hashes into are identified. The other elements in the buckets are considered as potential neighbours, with similarity being computed with them rather than the whole dataset.

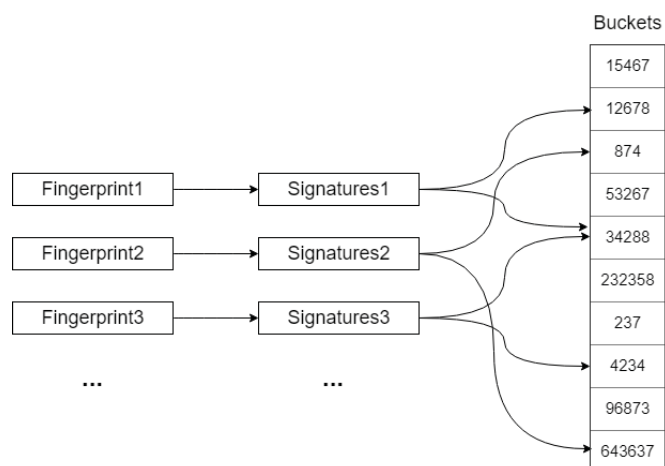


Figure 2.9: LSH process starting by converting fingerprints into signatures using Minhash method. Then signatures are used to hash the values inside into buckets. A signature will hash to multiple buckets according to the number of minhash values that make up the signature and the amount of minhash values that are grouped together for every hash. Image based on [45].

2.4 Big Data Paradigm - Map Reduce

Map Reduce is a concept that enables the processing of large datasets on distributed clusters [47]. This paradigm defines two main processing phases, the Map phase and the Reduce phase. Input tuple data in the form of $\langle key, value \rangle$ is provided to the model. Data is then automatically split and shared among the nodes to allow for distributed processing [48].

The Map phase takes the input data and performs a user defined action on each separate tuple, with the output being an intermediate $\langle key, value \rangle$ pair. The intermediate output is then shuffled and passed on to the reduce stage. Shuffling occurs in a way such that all tuples with the same key are grouped together into the same reducer (processing node) [47].

The Reduce phase then aggregates the values by key. When the data is passed on to the reducer, a function is defined to determine how the grouping of the values pertaining to the same key is done. Since Map Reduce ensures that all tuples with the same key are collected in the same node, local processing can be done without the need to communicate with other nodes. Finally, this results

in a final $\langle key, value \rangle$ pair as output [48].

2.5 Big Data Techniques

Through the use of cloud computing, the process of adding processing power for distributed systems can be done on demand, thus reducing the setup costs. However, this change in setup also requires a different way of thinking to exploit the distributed environment. Two of such frameworks that implement the Map Reduce paradigm to handle large datasets and that have recently increased in popularity both in academia and industry are Hadoop Map Reduce⁶ and Apache Spark⁷. Both frameworks were created as an abstraction of the communication framework between the nodes, to enable and empower developers to focus on the algorithm implemented rather than on the communication between them. Therefore these frameworks aim to reduce time and cost of implementation of distributed algorithms.

2.5.1 Hadoop MapReduce

Hadoop MapReduce is a distributed processing framework implementing the Map Reduce concept. It defines a master node being a job tracker and worker nodes. Hadoop MapReduce handles the process of splitting the task among the nodes to enable running in a distributed manner, monitoring of the job and reassignment of the failed tasks to new nodes. Input is taken from the file system and sent to the map tasks, with the output being shuffled and redirected to the reduce task. The output of such task is again written on the file system.

Map Reduce operations can be chained to perform more complex algorithms. At each iteration data is stored and retrieved from the file system. This can be of a bottleneck in data heavy operations.

⁶https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html (last accessed 25th May 2019)

⁷<http://spark.apache.org/> (last accessed 25th May 2019)

2.5.2 Apache Spark

Spark is a framework that builds upon the concept of Map Reduce with the aim of extending its capabilities. A Spark cluster consist of master nodes and worker nodes, with masters mainly handling work that is related to synchronisation among the workers such as shuffling of the data among the nodes, or recollecting and distributing the data. Workers on the other hand are the main computation nodes that Spark uses to perform the distributed work. Both masters and worker are long lived, meaning that once created they are alive through multiple queries [49].

One main noticeable difference from Hadoop MapReduce is the treatment of data in Spark, where the abstract concept of Resilient Distributed Datasets (RDDs) was created. RDDs enable easier implementation by abstracting distributed data as a normal list. Transformations, such as *filtering*, *map*, and *grouping* can then be applied on this dataset, with the processing being automatically distributed by the Spark Framework. These type of transformations are lazily evaluated until actions are called. Actions such as collect and reduceByKey are commands that retrieve the data and execute the transformations until that stage. This concept of lazy evaluation enables Spark to optimise steps of transformations done by the user by combining multiple operations together or thinking ahead for subsequent steps [50].

One performance optimisation Spark has over Hadoop MapReduce, is the diminished reliance of the framework on persistent storage. In Spark, main memory is utilised to store intermediate results while processing the data. In contrast Map Reduce must store and retrieve results from the file system, thus making it inefficient in tasks heavy on data operations.

Additionally Spark does not use replication as a way to implement fault tolerance. Instead, Spark uses lineage graphs. Lineage Graphs are graphs storing the transformations that are done on the data in the order occurred, as seen in Figure 2.10. When a node is lost, it reuses the lineage graph to reconstruct the lost section by rerunning the same operations on that subset of the data. This is more efficient when compared to replication as less storage and communication with file system is used [50, 51].

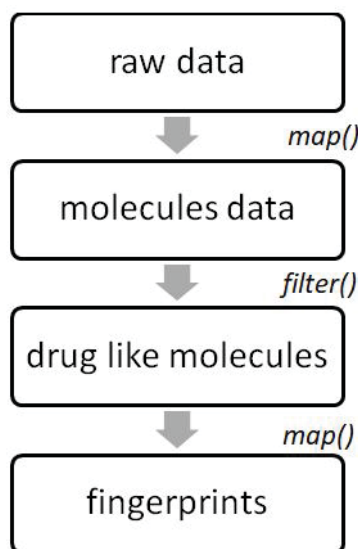


Figure 2.10: Lineage Graph. Boxes represent RDDs and arrows represent operations on the RDDs, adapted from [50].

2.6 Evaluation Criteria

Evaluation of clustering techniques on large datasets should focus on two aspects, the effectiveness of the clustering method and the performance of the algorithm. Effectiveness represents the quality of clusters produced, such that the grouping should be one where similar elements are grouped together while being separate from other less similar items [13]. In small molecules clustering context, effectiveness represents the extent which active small molecules are grouped together and separate from non active ones. Performance of clustering deals with the aspect of speed and the algorithm's ability to cater for increasing data, evaluating its ability to scale for larger datasets.

2.6.1 Evaluating Quality of Clusters

Various methods have been employed to determine the correctness of the grouping process, with some researchers using visual methods [52]. Standard methods that are used in the clustering context, such as Precision, Recall and F-Measure have also been widely used, while others have adapted existing meth-

ods to specifically cater for the given context of molecular clustering.

Precision, Recall, F-Measure

As discussed in [27], F-measure was used in the evaluation process of clustering documents. This was also used, less frequently, in chemical compounds clustering. To calculate F-measure, Precision and Recall are used. Precision represents the proportion of actives with respect to the total number of compounds in the cluster, while Recall is used to represent the proportion of actives clustered together out of all the actives available. Given that the total compounds in a cluster is represented using n , active compounds in a cluster using a and total active compounds in the dataset as A . Then Precision and Recall can be calculated using the following equations:

$$P = \frac{a}{n} \quad (2.6)$$

$$R = \frac{a}{A} \quad (2.7)$$

The results for Precision and Recall are then used to calculate the F-measure, representing the harmonic mean of the two results. The is calculated using:

$$F = \frac{2PR}{P + R} \quad (2.8)$$

In their application by [53] and [23], these three measures were calculated for every cluster produced and the highest F-value was used as the final result.

Quality Partition Index

Quality Partition Index (QPI) was first presented by [31], where algorithms were compared on their ability to separate active compounds from inactive ones. This was done by identifying the proportion of active compounds that were grouped in active clusters.

In their paper, [31] define an active cluster as a non singleton cluster which contains at least one active compound and active cluster subset as all the compounds within active clusters. The proportion of actives in the active cluster

subset is calculated, and is compared to the proportion of actives in the whole dataset. The increase, if any of the proportion is considered as an increase in the quality of the subset compared to the whole dataset.

[13] formalised the Quality Partition Index. However, the definition of active cluster was changed to a non singleton cluster which contains a higher proportion of actives compared to the whole dataset. This was done to reduce the negative effect that one active may have if it is misplaced in a large cluster [13, 23, 53].

Four properties are then defined, with p representing the number of active molecules in active clusters, q representing the number of inactives in active clusters, r representing actives in inactive clusters and s representing singleton active clusters [13, 23, 53]. QPI is then defined as:

$$QPI = \frac{p}{p + q + r + s} \quad (2.9)$$

The QPI measure incorporates the two types of errors in it. These are:

1. False Negatives, which are the active molecules treated as inactives and are represented by r in the QPI measure.
2. False Positives, showing the inactive molecules treated as actives and are represented by q in the QPI measure.

These are mentioned as *Error1* and *Error2* respectively by [54]. However, for *Error2* to be calculated correctly, the activity of every molecule with respect to the target protein must be known. When the activity for some molecules is unknown, then an unknown activity molecule being clustered with active molecules cannot be classified as an incorrect result unless manual testing is done. This was discussed by [54] where tests on big datasets were performed. In their research, actives with respect to one target protein are mixed with a large set of other molecules whose activity is unknown, with the aim of clustering active molecules together. The number of no activity compounds being clustered in active clusters was not directly treated as an error. They were later visually inspected to determine whether they were similar to the active compounds with which they were clustered.

2.6.2 Evaluating Performance

As data gets bigger, performance evaluation becomes more important to determine the applicability of the clustering algorithms on larger datasets. It enables the comparison of algorithms and how they would perform, in a theoretical context, on different dataset sizes. The two most popular measures are Scalability and Speedup.

Scalability

Evaluating scalability is done by tracking the algorithm's performance with respect to a changing dimension of the system. One common way is to divide the dataset into subsets of data instances of increasing sizes. Efficiency readings in seconds or minutes are then taken, to determine the speed of the algorithm for each subset of data, identifying the scalability of the method proposed [19, 55, 56, 57].

This method is frequently used when evaluating parallel and distributed systems. As seen in [40, 58, 51] and [59], this method of evaluating efficiency is used to determine how scalable the proposed methods are. In these scenarios, the number of cores or nodes used for distribution are taken as the variable to change, to see how their increase affects the performance of the method. This shows how increasing the number of nodes in the system affects the performance and provides insight on any overheads that may exist [51].

Speedup

Speedup is another way of representing the efficiency of the system in comparison to a standard method whose time is represented by T_S . The time taken for the newly proposed method is represented by T_P [56, 55]. The final result of speedup is a ratio representing the magnitude of increase or decrease in performance of a new algorithm with respect to a reference algorithm. Speedup is calculated using using:

$$S = \frac{T_S}{T_P} \quad (2.10)$$

2.7 Related Work

Clustering small molecules takes two dimensions. One is the creation of clustering methods, used on small datasets to test the quality of the clusters produced. The other is the creation or adaptation of clustering techniques to handle large datasets, which focuses also on the scalability and performance of the algorithm, coupled with the quality of the clusters.

2.7.1 Clustering Small Molecules

Most research has focused on clustering of small datasets with the aim of increasing the quality of the clusters. Various clustering approaches have been tried, with the most effective methods being hierarchical clustering algorithms [5]. Early studies by [30] and [31] compared different clustering algorithms. Although being slow, the hierarchical approaches used; Ward's, Group Average and Guenoche Divisive clustering obtained better results when compared to non-hierarchical ones. Results have been confirmed in other research by [52] and [60] for hierarchical clustering.

From the hierarchical approaches, Ward's clustering is considered as the state of the art for small molecules clustering, producing best separation for actives and inactives. It is repetitively used as a base algorithm on which to compare results, as can be seen used repetitively in Table 2.2, representing the use of clustering algorithms for small molecules. The effectiveness of this algorithm is attributed to the generation of clusters having high intra cluster similarity, prioritising the combination of clusters that minimise the distance between all the elements. Comparison to consensus clustering by [23], showed that Ward's still produced better results. It was found that multiple runs of the same algorithm do not perform consistently better than Ward's for varying datasets. Although at some particular cluster amount, one consensus clustering technique would obtain better results, a comparison on different number of cluster runs would prove otherwise.

On the other hand, a Weighted Voting based consensus clustering by [53] showed promising results. The Weighted consensus clustering managed to in-

crease QPI results minimally depending on the number of iterations for the algorithm and the algorithms chosen. This is a small increase with respect to Ward's clustering, resulting through the aggregated results of multiple clustering algorithms. Although the single algorithms chosen may be efficient such as k-means clustering, their repeated runs may make this approach not viable for large datasets. Notwithstanding the performance of hierarchical clustering algorithms, their quadratic time and space complexity make them unable to cluster large datasets.

Other researchers have investigated different methods of hierarchical clustering to try and improve efficiency results. [35] have implemented a way of creating a hierarchical clustering technique using k-means algorithm. This enables the algorithm to improve its running time with expected average case of $O(n \log n)$. This continued improving in later research, proposing an improved algorithm to reduce the number of singletons by applying fusion of clusters and singletons based on MCS methods.

Non-hierarchical techniques are normally employed when larger datasets are to be clustered. Jarvis Patrick clustering is regarded as the standard clustering for molecules in this group, with k-means and modifications of it being also employed [31]. However even though Jarvis Patrick is popularly used, [18] argued that the two parameters k and k_{min} for Jarvis Patrick need to be found for each new dataset. This makes it difficult and time consuming to use as each dataset would require multiple runs of the algorithm to identify the correct values.

A linearly scalable stochastic method was presented by [21], where initially diverse probes are identified, which are then used as cluster representatives. Evaluation showed that clusters represented by an active probe have an increased chance of having other active members when compared to clusters represented by inactive ones. In [32], Exclusion Sphere Algorithm was implemented, however the combination of structure extraction has made the algorithm slow, and thus not scalable for datasets having millions of compounds. This was enhanced by [17] by using fingerprints as comparison method and adding a pre-processing step to add determinism to the clustering process. An important advantage of such algorithm is that the representative compound in the cluster

is always within a threshold similarity to all the other compounds in the cluster. This enables the use of representative compounds for faster searches. [57] implemented Leaders' algorithm for fast clustering. The main drawback of the original algorithm was its nondeterministic nature, however the proposed algorithm sorts the input compounds such that different runs of the same procedure always returns consistent results. One drawback of the proposed algorithm is that the focus was on the speed rather than on the effectiveness of the results produced. Therefore further studies need to be done to determine how results compare to current methods.

2.7.2 Clustering Large Datasets

Public available datasets are getting larger in size at increasing rates. In 2005, ChemDB⁸ had 4.1 million compounds available while ChemNavigator⁹ had 10 million compounds [63]. Within 10 years, Pubchem¹⁰ had 16 million available structures, while ZINC15¹¹ had 120 million drug like compounds [64]. Nowadays ZINC15 has more than 700 million available drug like compounds [65].

This fast growth in data requires algorithms that are able to meet the demand to process such data. In fact, some research has started targeting increasing dataset sizes with methods such as approximation methods, distribution and parallelisation techniques. These aim to enable scaling of the algorithms.

Notwithstanding the increasing datasets, only few researchers have yet started applying clustering algorithms for large molecular datasets. [61] implemented a Map Reduce version of Ward's clustering, however this was only applied to small datasets. It managed to increase performance by around 60% when using 6 map tasks, enabling better performance than the serial version. On the other hand, approaches by [54, 26] and [19] distribute a custom implementation, following a similar pattern of identifying or creating an initial set of coarse grained partitions on which further clustering can then be done. [54] created a divisive hierarchical clustering with the aim of clustering 2 million compounds. Itera-

⁸<http://www.chemdb.com/> (last accessed 25th May 2019)

⁹<https://www.chemnavigator.com/> (last accessed 25th May 2019)

¹⁰<https://pubchem.ncbi.nlm.nih.gov/> (last accessed 25th May 2019)

¹¹<http://zinc15.docking.org/> (last accessed 25th May 2019)

Clustering Algorithms in Research		
Clustering Algorithms	Used In	Applied to Large Datasets
Ward's Hierarchical Clustering	[13, 8, 53, 27, 31, 18, 23, 61]	
Jarvis-Patrick	[31, 18, 20, 29]	
Exclusion Sphere	[27, 33, 32, 17]	
Butina Clustering	[17, 32]	
Hierarchical K-Means	[35, 62]	
Consensus Clustering	[8, 23]	
Average Link Hierarchical Clustering	[26, 60]	✓
Leader's Clustering	[57]	✓
Stochastic Clustering	[21]	
Szekely-Rizzo	[13]	
SCAP	[19]	✓
CVAA	[53]	
W-CVAA	[53]	
A-CVAA	[53]	
Guenoche Hierarchical Clustering	[31]	
Modified K-Means	[56]	
Group Average Hierarchical Clustering	[31]	
Custom Divisive Hierarchical Clustering	[54]	
CAST	[18]	
Yin Chen	[18]	
UPGMA Hierarchical Clustering	[52]	

Table 2.2: Review of clustering algorithms applied to clustering small-molecules. Applied to Large Dataset column shows whether the algorithm has been applied to datasets larger than 1 million molecules.

tively, using an incremental similarity threshold, the algorithm selects the most diverse compounds in each cluster, using a variant of MinMax algorithm. All compounds in that cluster are assigned to the most similar selected probes, with each group creating its own sub cluster. Once all the clusters are split, the threshold is increased, and each cluster is further split until each component resides in a unique cluster. This method managed to cluster the available data in 13 hours on a single machine.

[26] followed a similar approach, where components were first grouped using similarity searching, with each partition being further clustered using Average Link clustering. A parallel implementation of this approach was done to increase the performance of similarity computation, where all pairwise similarities were required.

Although Fingerprint based comparisons are faster, [19] chose to use a structure based approach to cluster the data. Considering that it is slow, a preclustering stage using set abstraction of graphs was used to enable fast partitioning of the data. Although partitions were bigger, this ensured that only structures having potential of being similar were clustered together. Slower and more accurate sub structure searching was then done on subsets of the data.

Both [19] and [26] used parallel processing approach on a server, to increase the performance of the algorithm. This enabled faster comparisons by using 32 and 64 cores concurrently for the respective researches. Table 2.2 shows the clustering algorithms applied to small-molecule clustering, highlighting the methods that were used to cluster large datasets.

[66] implemented integrated LSH approximation to accelerate similarity searching, with an additional contribution by combining it with the Jarvis Patrick algorithm. The results improved drastically, with results showing that there were improvements of 20-80 folds. The system was run on a computer cluster, further enhancing its scaling capabilities. One drawback of the LSH method is the memory required when implementing it in distributed environments. These would require data shuffling among the nodes and use of replication to handle nodes failures. This might result in large memory consumption [43, 44].

2.8 Chapter Summary

This chapter presented the current state in clustering small molecules. It analysed the necessary steps that are required to correctly represent molecules, and a classification of the algorithms that are available. However, in the presented related work there is a lack of clustering large datasets. In the current age of increasing dataset sizes and generation of data, this requirement is increasing in its importance.

Methodology

In this chapter, the design and implementation details of our approaches for distributed clustering of large small-molecules datasets are outlined. The chapter starts by first providing details about the experiment done to determine the approach which was to be distributed. This is then followed by details of the approaches implemented based on the selected algorithm. Following tests in a serial environment, Butina algorithm was found to provide good quality clusters while requiring less memory and minimal parameter settings therefore it was chosen as the method to distribute using the Spark framework.

3.1 Decision of Proposed Approach

The decision of the approach to distribute was taken following experiments to determine the best approach among the algorithms discussed in this section. The decision was based on the the scalability and result quality aspects of the algorithms. Experiments were required as nothing was identified in literature that comparatively evaluates the performance of the algorithms identified together on the same dataset and evaluation measures. Through literature analysis, four algorithms were identified having the potential of further investigation, keeping in mind the aims of the research discussed in Section 1.2. The chosen algorithms are:

1. Ward's Hierarchical Clustering - This clustering algorithm is the standard algorithm for chemical clustering, achieving the most effective clustering [53]. Therefore it merits to be further investigated in addition to the fact that it can be used as a base case for the results of other algorithms to be compared with.
2. Jarvis Patrick - This non-hierarchical clustering techniques has also been widely used in chemical clustering, providing better space complexity than Ward's clustering algorithm. Although mixed results have been achieved as seen in [30], its application has been widespread.
3. Butina Clustering - Butina clustering has been applied in a number of scenarios but is not widespread as Ward's or Jarvis Patrick as seen in Section 2.7.2. However it is based on the Exclusion Sphere approach, which has been used in a number of studies and therefore it is a technique worth visiting because of its interesting approach of selecting centroids. It was developed with the aim of creating more homogeneous clusters and achieved promising results when compared to Jarvis Patrick clustering [17]. Additionally, it requires minimal parameter setting, thus making it ideal for large datasets.
4. Exclusion Sphere Clustering - Exclusion Sphere clustering is a non deterministic single pass algorithm that has been applied in a number of studies for clustering of data and diversity selection of molecules as shown in Table 2.2. Compared to the other chosen algorithms, this method is the most scalable and therefore considering the large datasets scenario, it was added to the list of algorithms chosen.

3.1.1 Experiment Setup

Whenever possible, RDKit¹ implementations were sought since it is a standard library for chemical compounds manipulation. Both Wards and Butina clustering algorithms were available. Jarvis Patrick clustering was then implemented

¹<http://www.rdkit.org/> (last accessed 25th May 2019)

using the library `jarvispatrick`², with Exclusion Sphere clustering being implemented as per literature [32]. Extended Connectivity fingerprints, a type of Circular Fingerprint, were used based on the Morgan algorithm using RDKit library. Circular fingerprints have been applied in a wide manner in research with positive results when compared to other representations [13, 22, 67]. Tanimoto similarity was then used as a similarity metric because of its consistent use in research as [26]. Tests were run in a serial manner on a node having the following specifications:

- OS: Ubuntu 16.04 (Virtual Machine)
- CPU: Intel(R) Core(TM) i7-6700HQ 2.60GHz
- Cores Allocated: 4
- RAM Allocated: 8GB

The dataset used for this experiment was the DUD-E dataset, further explained in Section 3.4.1. Three targets from different protein categories were chosen, with their respective active and decoy molecules being obtained and combined in a single dataset. Renin (Protease), Tyrosine-protein kinase ABL1 (Kinase) and Thyroid hormone receptor beta-1 (Nuclear Receptor) were selected. The choice of different categories enables us to get a better picture of the clustering quality on different types of small molecules.

3.1.2 Scalability Results

Scalability results were performed to determine the applicability of the algorithms on large datasets. Scalability was tested by obtaining one of the datasets chosen and creating nine subsets from it, with each subset containing more data than the previous one, starting from 10% of the dataset up to 100%.

Figure 3.1 shows the results obtained from the tests performed. Exclusion Sphere clustering was run two times with different similarity threshold as it affects the number of similarity calculations required, hence the scalability of the

²<https://github.com/llazzaro/jarvispatrick> (last accessed 25th May 2019)

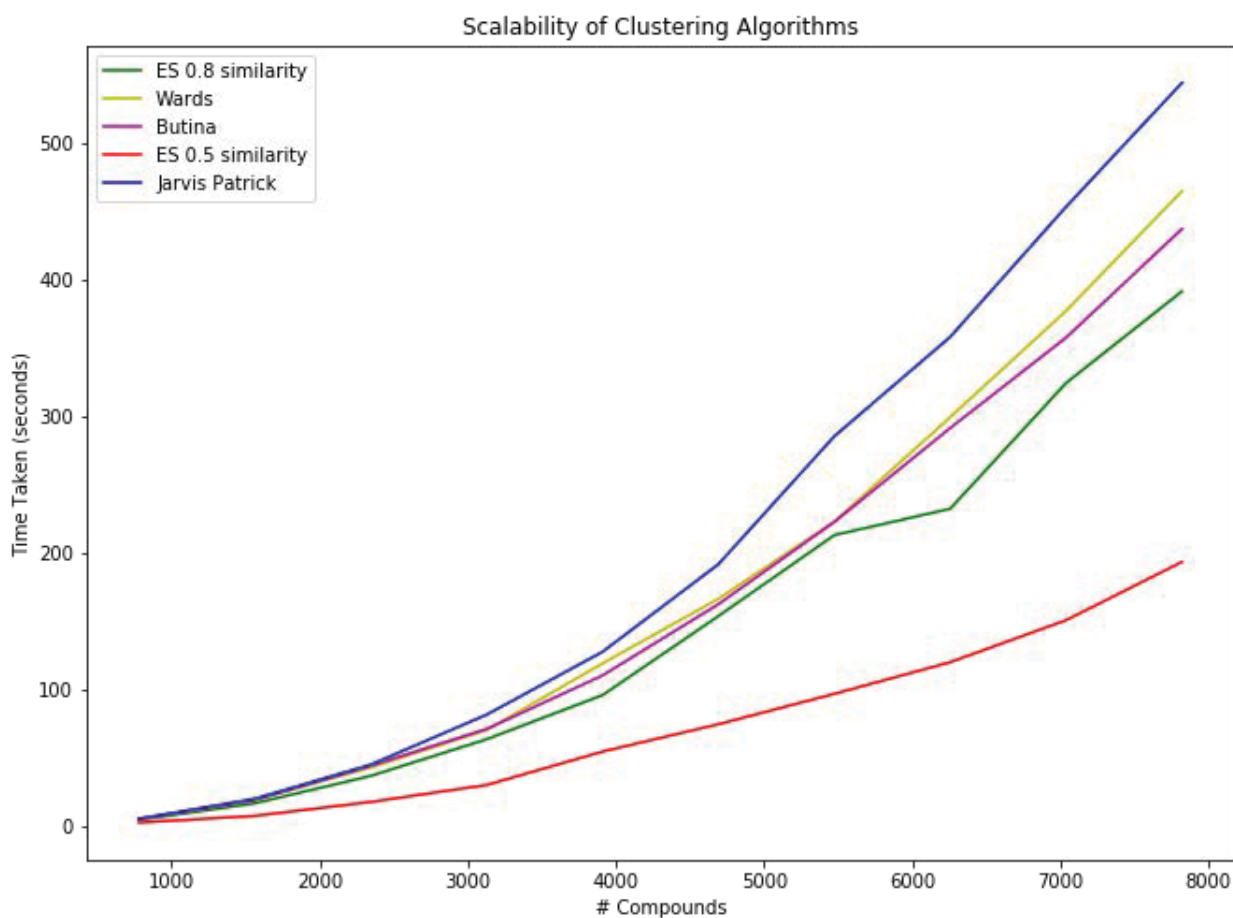


Figure 3.1: Scalability analysis for the selected clustering algorithms on increasing datasets from 761 up to 7610 molecules.

algorithm. As seen from the results, Ward's, Jarvis Patrick and Butina have the same scalability. The main contributor for this is the calculation of similarity matrix where it is required for all the three algorithms. Jarvis Patrick although having the same scalability, seems to be the slowest. This is mainly due to implementation differences, where both Wards and Butina clustering use RDKit library, which is optimised for molecular data manipulation. Exclusion Sphere algorithm scalability is dependent on the similarity threshold used, reaching $O(N^2)$ as the similarity threshold reaches to 1.

With regards to memory usage, Ward's clustering is the most demanding, having space complexity of $O(N^2)$ as it requires to store the complete similarity

matrix. Both Jarvis Patrick and Butina clustering require only the list of neighbours, a subset of the similarity matrix, therefore being more scalable for larger datasets. Exclusion Sphere does not require any storage for the similarity computations, as a molecule is assigned the moment similarity computation is done, making the similarity value useless thereafter.

3.1.3 Clustering Quality Results

The aim of the approach to be chosen is to cluster the data based on the activity classes of the molecules within the dataset such that active molecules with respect to one target protein are separate from non active ones. This helps when selecting initial molecules for diversity screening, such as not to get molecules from the same activity class and therefore reduce the amount of molecules screened. Additionally, it also enables the selection of molecules from known activity classes as molecules in the same cluster have similar activity. QPI and F-measures (explained in Section 2.6.1) were therefore used as criteria to determine quality of clusters as these were found to represent best this idea of separation.

To obtain the results, the algorithms were run with a number of parameters and the highest results achieved by each algorithm were recorded. For Ward's clustering, clusters were selected at 100 intervals and readings were noted. For Butina and Exclusion Sphere algorithms, readings were taken for similarity thresholds at increments of 0.1, starting from 0.1 up to 0.9. Finally for Jarvis Patrick, a set of k and k_{min} parameters were identified according to [12], which uses the k value of 16 as a default value as used in Daylight suite of programs. Similar parameters were also used by [29] and [20], with k_{min} values being normally set as multiples of 4. However after some unsatisfactory results, additional parameters were added. k value was halved and doubled to get a wider ranges of k and k_{min} values.

Figures 3.2 and 3.3 shows the QPI and F-measure results obtained respectively. With respect to QPI, Wards is consistently the algorithm that manages to separate actives from inactives in the most optimal manner. Butina and Exclusion Sphere achieved comparable results, with Butina being better than Exclusion Sphere method. This is understandable considering that Butina is based

on Exclusion Sphere approach but with a better selection of cluster centroids. Jarvis Patrick obtained unsatisfactory results. Research by [30] also found that Jarvis Patrick is not as effective as Ward's clustering, however considering its widespread use one would presume that better results would be obtained. One possible cause of such results may be that the parameters chosen were not optimal as each dataset requires a different set of parameters. This is a known issue of Jarvis Patrick algorithm, being also mentioned by [18].

Analysing the clusters produced, Wards clustering resulted in having less singleton molecules than the other competing methods, with only 1 molecule being clustered as singleton from the three datasets. However it tends to include more non active molecules into active clusters when compared to results obtained from Butina clustering.

F-measure values are recorded for every cluster, and the results of the highest scoring cluster were then used, as used by [53]. From the results obtained, and seen in Figure 3.3, although Ward's clustering manages to separate actives from inactive molecules in the most optimal way, it also tends to separate actives into smaller clusters. This was noted by the contrasting scores of Precision, representing the fraction of the molecules within the cluster that are active, and Recall, representing the fraction of actives clustered together out of all the actives in the dataset. High Precision rates were recorded, with all molecules in the cluster being actives, while the largest cluster of actives contained an average of 28% of active molecules. These results were obtained at a different hierarchical level than those of the QPI values. The results are consistent with those obtained by [53], where QPI values were the highest for Ward's clustering among the methods tested while F-measure results were comparatively low. Additionally, result by [23] have also shown that QPI values tend to increase when more clusters are selected from the hierarchical tree at lower levels, while F-measure is inversely proportional to the number of clusters.

Butina obtained similar results to Ward's clustering with regards to Precision, with all the molecules in the cluster being active. Additionally, the average Recall rate among the datasets was that of 51%, meaning that half of the actives were clustered together in the same cluster. The highest F-measure scores for Butina were recorded at 0.3 similarity threshold for all the three datasets tested.

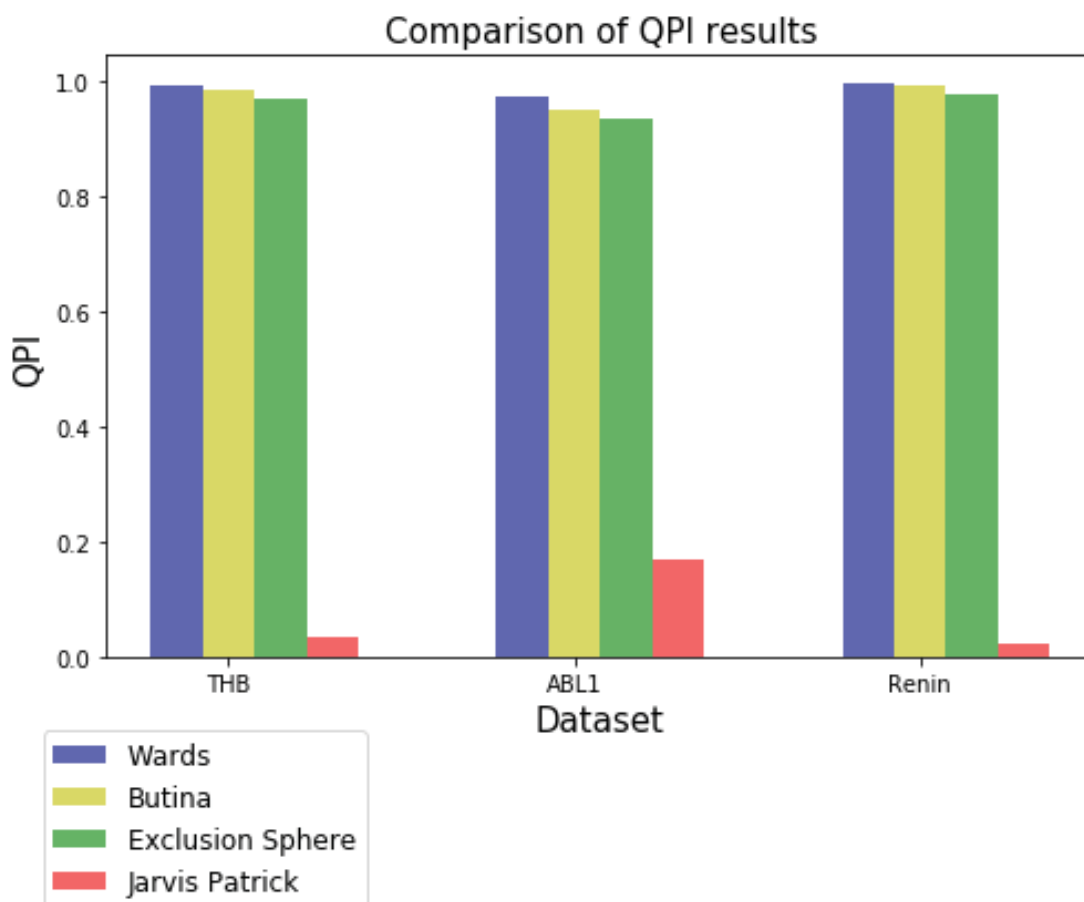


Figure 3.2: Comparison of the effectiveness of the algorithms using the QPI measure on the three selected datasets.

The results for Exclusion Sphere clustering and Jarvis Patrick follow the same results for the QPI, with Exclusion Sphere achieving similar results to Butina clustering while Jarvis Patrick obtaining low results.

Based on the results obtained, Butina clustering algorithm is an interesting approach to further investigate. It managed to obtain comparable results to Wards clustering, even though some active molecules were separated as singletons. Singletons may be a problem as these would not be encompassed by a group, therefore requiring individual testing. On the other hand, it managed to cluster half of the active molecules together, making it easier to identify additional actives when an active is already known. One drawback of such approach

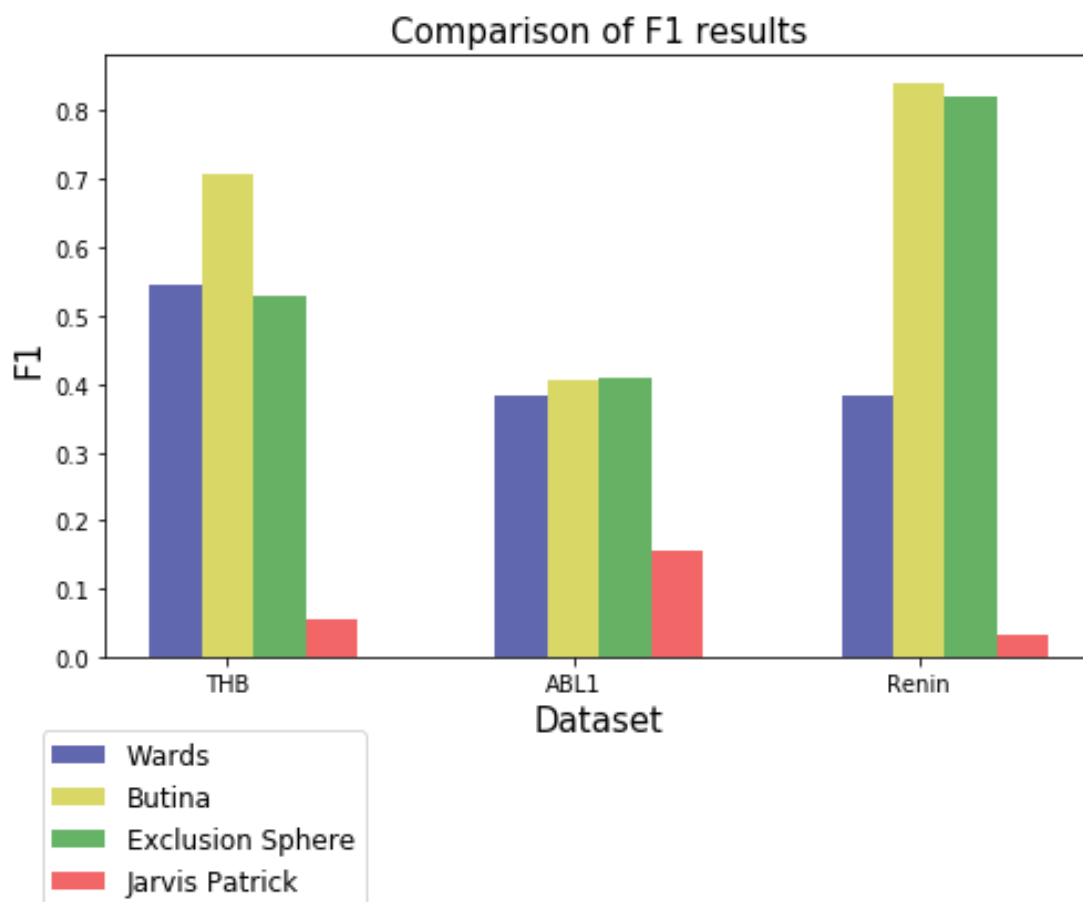


Figure 3.3: Comparison of the effectiveness of the algorithm using the F-measure on the three selected datasets.

is that the output is not in a hierarchical format. A hierarchical output enables better insight on the data as it allows drilling down to get more specific subsets or getting larger clusters.

Butina is also more scalable with regards to memory requirements. $O(N^2)$ may become unsustainable when dealing with datasets of millions of molecules, hence this should be also taken into account. In comparison to Jarvis Patrick, Butina also requires only one parameter being the similarity threshold. This makes it ideal to use by non technical users, while also requiring less investigation to identify the correct set of parameters. Exclusion Sphere clustering approach also obtained results similar to those by Butina and hence merits further

investigation. However considering that Butina algorithm is an enhancement on the Exclusion Sphere approach and it helps to obtain better results, it was decided that Butina clustering would be selected for distribution.

3.2 Approach Overview

Following the choice of the Butina clustering algorithm, two approaches were implemented based on this method. These are:

1. D-Butina: Distributed Butina (D-Butina) clustering algorithm was implemented, with the new approach producing the same results as the Butina clustering, however the algorithm was re-written to fit in a distributed environment.
2. DLSH-Butina: Distributed Locality Sensitive Hashing Butina (DLSH-Butina) was implemented by applying Locality Sensitive Hashing approximation method to the Butina clustering algorithm, substituting the calculation of the complete similarity matrix. This was done with the aim of improving the speed performance of the algorithm.

The Tanimoto similarity metric was then used for the similarity calculations among the fingerprints since it is the most widely used metric in the field as discussed in Section 2.2.5. Extended Connectivity fingerprint was then used for molecular representations as discussed in Section 3.1.1. All the processes in the flow are implemented to run in a distributed environment to enable scaling out of the system. A high level of this approach can be seen in Figure 3.4, where tasks are divided among n nodes, with each node computing local computation on a unique subset of the data. The distribution and synchronisation is managed by the master node assigning the tasks and handling data synchronisation among the workers. An overview of the distributed architecture is shown in Figure 3.5.

For the implementation, Python version 2.7.6³ was used, with Apache Spark version 2.3.0⁴ being used for the distribution. RDKit library version 2016_03_01

³<https://www.python.org/download/releases/2.7.6/> (last accessed 25th May 2019)

⁴<https://spark.apache.org/releases/spark-release-2-3-0.html> (last accessed 25th May 2019)

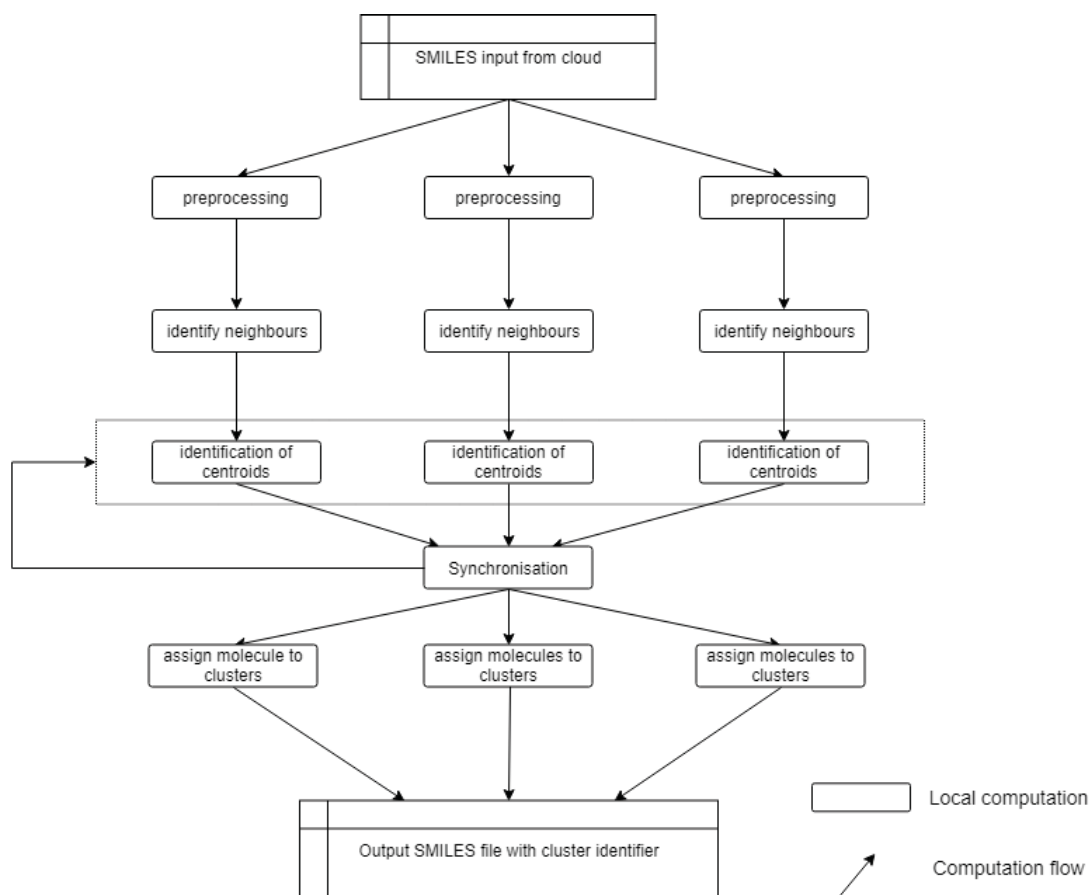


Figure 3.4: Abstract view of the approaches presented, with the tasks being split among n number of nodes, with each node performing local computation of the task at hand on a unique subset of the data. Diagram based on [61].

⁵ was used to enable handling of molecular data, conversions between the formats and similarity comparisons. Python was used as the language of choice because of its wide applicability for Machine Learning. Additionally RDKit library is natively written in Python using C compiled libraries, therefore the use of Python was optimal for this case for ease of use and integration with this library. The implementation and tests were run on Ubuntu Linux version 16.04 LTS.

⁵https://github.com/rdkit/rdkit/releases?after=Release_2016_09_1a1 (last accessed 25th May 2019)

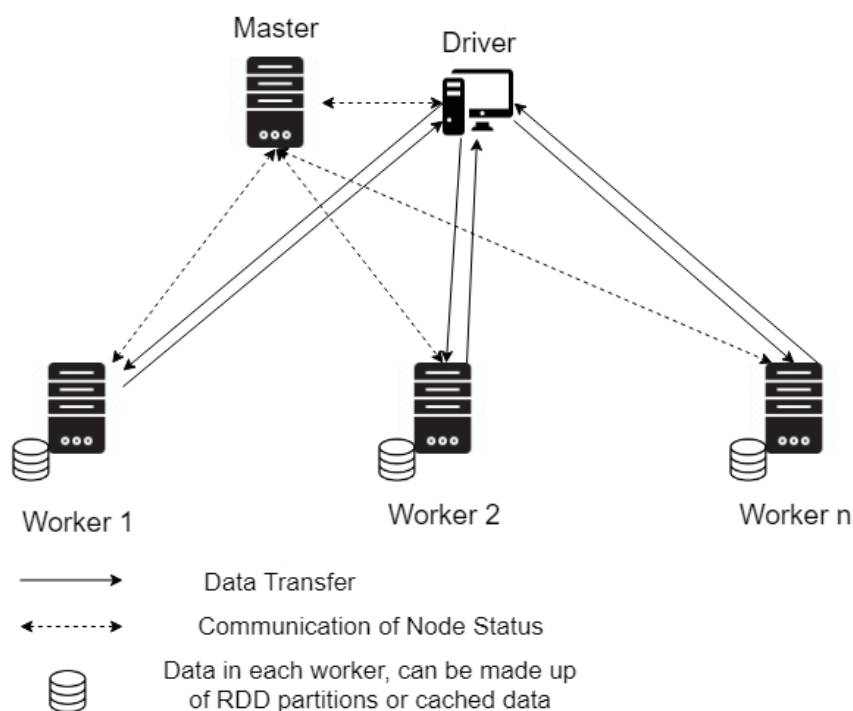


Figure 3.5: Architecture of the distributed Spark clusters, adapted from [44].

3.3 Spark Framework

The Apache Spark framework was used for the implementation of the approach presented. Spark is built on the Map Reduce Paradigm using a distributed data model to abstract how the distribution of data is performed. It uses Resilient Distributed Datasets (RDDs), a collection of data partitioned among nodes, to abstract the distribution of data. It enables in-memory data processing, providing better performance than Apache Map Reduce⁶ framework [50].

Functions provided by Spark are called on an RDD and automatically distribute or shuffle the data as necessary. It offers the functions `Map` and `ReduceByKey` that work in a similar way to the functions offered by the Map Reduce framework. However in addition to that, Spark also offer other functions to filter the data in a distributed manner and to flatten the data as necessary.

Another important functionality of Spark is the broadcast. Broadcast is the

⁶<https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html> (last accessed 25th May 2019)

process of sending the same set of data to all the worker nodes. For this to be done, it must collect all the subsets of data from each worker, combine them into a single dictionary in the form of $\langle key, value \rangle$ and replicate the result to all the workers. Although it is a useful functionality, it is a heavy operation as data needs to be transferred back and forth between the driver node and the workers. If abused it may easily become a bottle neck for the system.

3.4 Datasets

Two datasets have been used for the tests performed. These are the Directory of Useful Decoys-Enhanced (DUD-E) and the ZINC datasets. In addition to these datasets, other datasets were also used in literature, however these were not freely available.

3.4.1 DUD-E

DUD-E⁷ dataset provides active and generated decoy molecules for target proteins. This dataset is useful as actual activity of the data with respect to a target protein is known. Decoys are molecules from a larger dataset that were carefully selected and tested to be physically similar to ligands, yet they do not bind to the target protein. Additionally, eight different categories of proteins are made available, making the dataset more diverse. These are: Nuclear Receptors, GPCR, Ion Channel, Other Enzymes, Cytochrome, Protease, Kinase, and Miscellaneous Proteins. The data is available in sdf format and is subdivided by target molecule [68].

3.4.2 ZINC

ZINC⁸ dataset is a non commercial and freely available dataset. It is continuously updated with new compounds, which at the time of download amounted to 761,841,197 molecules which can be either downloaded in SMILES format or

⁷<http://dude.docking.org/> (last accessed 25th May 2019)

⁸<http://zinc15.docking.org/> (last accessed 25th May 2019)

accessed through an API. From this dataset, those relevant to this research were selected, this was done using the provided filtering to select only 'drug-like' compounds. This reduced the dataset to 749,376,606 molecules, selecting only those having Molecular Weight between 250 and 500 Daltons. Additionally these compounds also have $LogP < 5.0$ [65].

3.5 D-Butina

The Butina clustering algorithm can be conceptually divided into two steps:

1. Neighbours identification phase: The exact list of neighbours is required for each molecule in the dataset.
2. Clustering of the molecules: Given the molecules and their list of neighbours, molecules are clustered accordingly starting from the molecule having most neighbours.

The steps of the algorithm can be seen in Figure 3.6. Initially data is pre-processed to provide the necessary input for the neighbour identification phase. SMILES data is read from file storage, indexed and divided into equally sized splits $D_1, D_2 \dots D_n$. The splits are then distributed among the workers, which in turn convert the SMILES data into fingerprints.

3.5.1 Neighbours Identification Phase

Neighbours can be defined as those molecules whose similarity γ with a target molecule is equal to or greater than a defined threshold T . The identification of exact neighbours requires the calculation of all pair-wise similarities. This is a demanding task that requires $O(N^2)$ time to complete, with N being the number of molecules. Similarity calculation is the most time consuming operation in this clustering method, therefore an implementation of similarity calculation and neighbour identification in a distribution manner is done. Although time complexity of the calculation still remains the same, the time is distributed among the nodes and therefore it can handle more data in the same amount of time.

Firstly, the master node, takes the entire list of fingerprints F_{all} and broadcasts it to all the worker nodes. Secondly, the same list of indexed fingerprints is divided into splits $F_1, F_2 \dots F_n$ and distributed among the nodes in the spark cluster, with each molecule fingerprint being present in only one split. This technique of calculating the values of a matrix by replicating all the columns to all nodes, and then dividing the rows among them is a technique normally used in cases

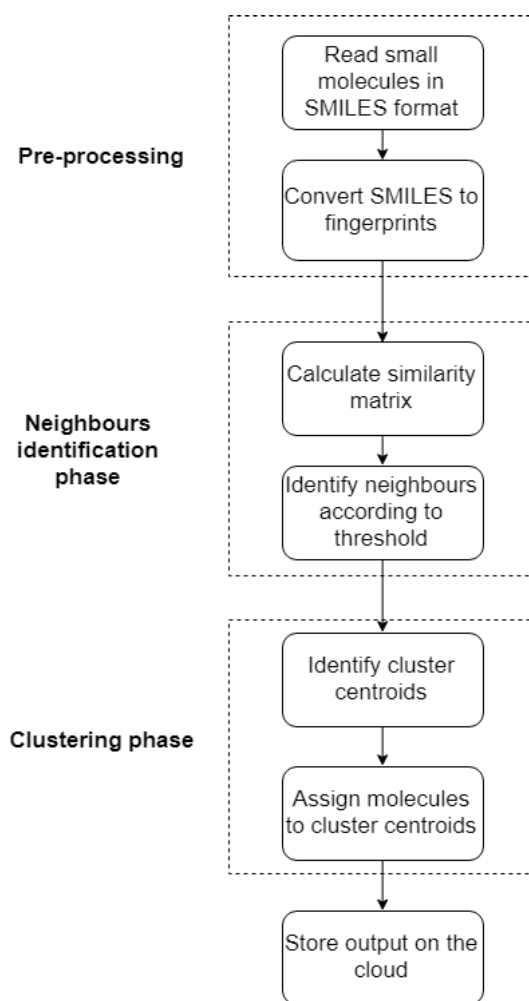


Figure 3.6: High level diagram of the D-Butina process. The pre-processing is required by all the clustering algorithms, while the neighbours identification phase is required by all the methods requiring either neighbours or similarities among the molecules.

		1	2	3	4	5	6	7	...	m
Worker 1	1									
	2									
	3									
	4									
	...									
	a									
Worker 2	a+1									
	a+2									
	...									
	b									
Worker m	f+1									
	f+2									
	...									
	m									

Figure 3.7: Calculation of the complete similarity matrix distributed among the worker nodes. The columns are replicated on all the nodes, while the rows are divided among the workers as shown. The highlighted section of the matrix is the only part that is calculated. Diagram adapted from [61]

when the amount of columns is less than that of rows [69]. This occurs because data transfer is heavy and therefore, distributing the smallest list of the two is always suggested. However in our case, the number of rows and columns are equal and therefore all the list is broadcasted.

A subset of the similarity matrix is then computed on each separate worker as per Figure 3.7. Considering the symmetry of an N by N matrix, only the lower triangle of the similarity matrix requires computation. This reduces the time required by half.

Each element from split F_i in the form $\langle id, fingerprint \rangle$ is matched with a subset of F_{all} also of the same form. The subset is chosen by selecting those fingerprints whose identifier is smaller than the matching fingerprint to obtain the lower triangle of the similarity matrix. The resultant combination is stored as an RDD as $\langle key, value \rangle$ pairs. Each element is represented by a key of type molecule identifier and fingerprint, with the value being the subset list of fingerprints.

Similarity is then computed between the key molecule and the list of values.

RDKit is used to compute the similarity mainly because of its efficiency. It provides a bulk similarity function whereby given two parameters, a fingerprint and a list of fingerprints, the similarity of parameter one against all the others is computed. This is more efficient with respect to calculating similarity of each pairwise combination separately, as it benefits from caching in CPU memory to increase its performance.

To make the algorithm efficient with regards to space consumed, comparisons are instantly filtered to maintain only those having $\gamma > T$. This means that any γ values below T are discarded. This reduces space complexity of the algorithm from $O(N^2)$ to $O(Nm)$ with m being the average number of neighbours. The value of m depends on the similarity threshold. It gets closer to N as γ gets closer to 0, while getting smaller as the γ reaches 1. Once molecules below the threshold are filtered out, the data is flattened such that a row of $\langle molecule, list_of_neighbours \rangle$ becomes multiple rows of $\langle molecule, neighbour \rangle$. Since initially, only the lower triangle of the matrix was computed, only half of the neighbours are found. Therefore each subset of the RDD containing neighbours, that is spread among the workers, is replicated, with the replicated version having its target molecule and neighbour molecule values inverted. Therefore, the complete list of neighbours is obtained.

Finally a reduce operation groups all the rows having the same key molecule in the same worker. The rows are combined by the key to convert multiple rows into a single one having the target molecule fingerprint and a list of neighbours.

3.5.2 Clustering Phase

No distributed implementation for the Butina clustering algorithm was found in literature on which improvements could be made. Therefore a number of approaches were tried before identifying a successful implementation. A distributed implementation of the serial algorithm as presented by [17] was not possible, as each potential cluster centroid required the results of all the previous generated clusters to determine whether a centroid should create its own cluster or not, making it non distributable.

A second idea was to treat the process as a set difference approach. Each row

would be considered as a set, computing a set difference with all the rows having more neighbours than itself. This would remove molecules in the set, that were owned by clusters having more neighbours. However this approach was also unsuccessful because of incorrect results obtained. The problem with this approach stems from fact that clusters generated by Butina are dependent on previous cluster results and a cluster may be invalidated by another previously created cluster owning its cluster centroid. This issue is shown in Figures 3.8a and 3.8b, where the placement of a molecule may change based on the composition of higher clusters.

Therefore, the result at any given row depends on all the previous clusters. This dependency makes it challenging to distribute the process as row y depends on the outcome of all rows from 1 to $y - 1$. To overcome this problem, the solution had to be divided into two steps.

1. Identification of cluster centroids: This step identifies the molecules that will become cluster centroids.
2. Assignment of molecules to clusters: After cluster centroids are known, clustering can be performed without the issue of clusters being removed during the process as seen in Figures 3.8a and 3.8b.

Identification of Cluster Centroids

To identify cluster centroids, an iterative algorithm was created whereby molecules are temporarily assigned to different clusters until a molecule is either designated to be a cluster centroid or removed from the list of potential cluster centroids. The temporary assignment of molecules to clusters will not be the actual cluster of a molecule, however this process is only used to identify cluster centroids. To reduce the dependency that exists in serial Butina, among potential clusters while they are being created, the commutative property of neighbourhood is used.

If molecule a is a neighbour of molecule b , then b is also a neighbour of a .

...									
54	23	25	68	89	87	75	15	21	
12	11	45	67	78	54	33	21		
63	76	55	43	23	65	88			
...									
21	54	12	...						
...									

(a)

75	54	23	17	19	27	44	42	22	90
...									
54	23	25	68	89	87	75	15	21	
12	11	45	67	78	54	33	21		
63	76	55	43	23	65	88			
...									
21	54	12	...						
...									

(b)

Figure 3.8: Assignment of molecule having identifier 21 to a cluster in two different scenarios. The reference molecules, in gray, are ordered in a descending order by the count of neighbours. (a) Molecule 21 (highlighted in red) is a neighbour of both molecules 54 and 12. However it is assigned to cluster having centroid 54 since it has more neighbours than 12, therefore taking precedence. (b) A similar scenario to (a) is shown, however 54 is assigned to cluster having centroid 75 as it is in its neighbourhood. Therefore it cannot have a cluster of its own, leading molecule 21 to be free. This can then be owned by cluster having 12 as its centroid.

From this, it can be concluded that a molecule can either be in a cluster whose centroid is itself, or any other cluster whose centroid is in its neighbourhood. This can be seen in Figure 3.8a, where molecule 21 can be assigned to clusters having centroids 54 or 12. Both 54 and 12 are within the neighbourhood of molecule 21. Molecules within the neighbourhood of a reference molecule can be:

- Molecule having more neighbours than the reference molecule
- Molecule having same amount of neighbours as the reference molecule
- Molecule having less neighbours than the reference molecule

Based on the rules of the Butina clustering, a molecule may only be assigned to a cluster having either more neighbours or the same amount of neighbours than itself. If there is no molecule satisfying this condition, then the molecule creates a cluster for itself, encompassing the remaining neighbours. Should there exist any two molecules having the same amount of neighbours, then a tie breaker is used for determinism purposes as implemented in RDKit⁹. This tie breaker determines that the molecule having the smallest identifier takes precedence.

To identify cluster centers, and to reduce the amount of data transfer, the algorithm treats each row (molecule and list of neighbours) as a separate entity, performing a task in iterations with synchronisation at every iteration. This idea is similar to the one used by [61] to perform hierarchical clustering using Spark, where the most similar compounds are found in an iterative manner and shared among all the workers. It is also equivalent to a bulk synchronous parallel (BSP) approach, where a number of steps are processed in iteration, with each iteration requiring a synchronisation step [50].

This algorithm requires the count of neighbours for every molecule in the dataset. To do this, the output of neighbour identification process explained in Section 3.5.1 was altered by adding the count of neighbours to each molecule, such that the output becomes an RDD of rows having the properties:

⁹http://www.rdkit.org/Python_Docs/rdkit.ML.Cluster.Butina-pysrc.html (last accessed 25th May 2019)

molecule id	No. Neighbours	List of Neighbours
65	15	[<neighbour id, No. Neighbours>, ...]

The neighbours are ordered in a descending order based on their count of neighbours. This enables faster performance to identify the neighbour having the most count of neighbours at each iteration.

The distributed algorithm to identify centroids accepts an RDD of molecules and their neighbours as input, with both the molecules and the neighbours being in the form of $\langle molecule_identifier, count_of_neighbours \rangle$. The RDD is distributed among the workers, with each worker performing the same operations on different subsets of the data.

Each molecule is assigned a cluster centroid. This can be, either one of its neighbours having the most count of neighbours or the molecule itself. Since neighbours are already ordered by their count of neighbours, the assigned cluster centroid can only be the first element in the neighbour list or the molecule itself, depending on which molecule has the largest count of neighbours. Therefore only a check on the first element is required.

Two lists are created, whereby molecules are assigned to either one or the other:

1. *valid_cluster_centroids* - any molecule assigned to this list during the process of the algorithm is a confirmed cluster centroid as part of the final result for D-Butina.
2. *invalid_cluster_centroids* - any molecule assigned to this list during the process of the algorithm cannot be a cluster centroid as part of the final result. Therefore it is considered invalid.

Any molecule assigned to a cluster that is not itself, may have the cluster changed as that cluster centroid may be present in another cluster as seen in Figure 3.8b. Therefore it remains pending for further computation. However, any molecule that is assigned to its own cluster will not have its cluster changed. This happens as a molecule is assigned to its own cluster in cases where no neighbours having more count of neighbours than itself exist. As per

Butina rules, a molecule may be either assigned to a cluster centroid having more neighbours than itself or it creates its own cluster. Therefore the clusters of such molecules are confirmed and as such they are obtained and stored in *valid_cluster_centroids* list. Any neighbouring molecules to cluster centroids in *valid_cluster_centroids* list will belong to those clusters and therefore cannot become cluster centroids of their own. Hence they are retrieved and assigned to *invalid_cluster_centroids* list.

At this stage, each worker in the Spark cluster will have a subset of RDDs *valid_cluster_centroids* and *invalid_cluster_centroids*. Therefore, at each iteration, the list of *invalid_cluster_centroids* is collected on the master node and combined in a single list. This is converted into a dictionary *all_invalid_cluster_centroids* with the key being the molecule identifier and an empty value. The change to a dictionary ensures that checking whether a molecule identifier exists in the dictionary takes only $O(1)$ time. The dictionary is then shared among all the workers. The list *valid_cluster_centroids* does not require collection, as the list *valid_cluster_centroids* and the molecules in that subset are both present on the same worker node.

The molecules in *all_invalid_cluster_centroids* have no chance of being a centroid and hence are removed from the data being processed. Therefore elements having their molecule identifier in *all_invalid_cluster_centroids* are removed from the RDD. Additionally, any neighbour molecule that has its molecule identifier in *all_invalid_cluster_centroids* is also removed. This process of deletion aids to increase the performance of the algorithm at each iteration by having less data to compute.

Molecules in the list *valid_cluster_centroids* are also removed to further reduce data that requires computation. However since they are recorded in the list *valid_cluster_centroids*, they are not lost.

After the molecules are deleted, the iteration starts again by assigning each molecule a new cluster centroid as per the process explained. Some molecules may have their cluster changed as their previously assigned centroid would have been removed during the process of deletion. Others may remain assigned to the same neighbour as it would have not been removed yet. As per above, any molecules assigned to their own clusters are appended to the list of

valid_cluster_centroids, while their neighbours are assigned to *invalid_cluster_centroids*. However, in the case of *invalid_cluster_centroids*, it is emptied and reassigned at each iteration as it is useless to keep removed molecules. The process of selecting *valid_cluster_centroids* and *invalid_cluster_centroids* repeats itself until the list of *all_invalid_cluster_centroids* is empty. This signifies that all centroids have been found and that no other molecule with the potential of being a centroid remains. Finally, all the subsets of *valid_cluster_centroids* are collected in a single list and becomes the final output of this algorithm, representing all the cluster centroids of the Butina algorithm. Figures 3.10a-3.10g show a working example of the algorithm to identify cluster centroids.

Pseudo Code for the algorithm is shown in Algorithms 1, with Algorithms 2 and 3 showing in detail the sections of assigning a molecule to a cluster and the removal of neighbour molecules that exist in *all_invalid_cluster_centroids*. Figure 3.9 then shows the distribution of the tasks between the workers and the communication required among the nodes.

Assignment of molecules to clusters

The final step is to assign molecules to the cluster centroids identified. The list of neighbours for each molecule is obtained for each molecule in *valid_cluster_centroids*. A set difference is required by every cluster centroid to all the other cluster centroids that have more neighbours than itself. Conceptually, the cluster centroids are ordered in a descending way by count of neighbours and each centroid does a set difference of its neighbour molecules with all the previous cluster centroids. This is done using a cartesian operation, discarding any cartesian result of type $((molecule1, listofneighbours), (molecule2, listofneighbours))$, where the count of neighbours for *molecule1* is greater than the count of neighbours for *molecule2*. This time, no clusters will be removed. The results from the set difference are intersected by key (molecule identifier) to keep only those molecules that were not assigned to larger clusters. The final result is the same as the serial Butina Clustering. This was confirmed through tests using various datasets.

The distribution of this section is done by dividing the calculation of set difference among the nodes. The result is then grouped by cluster centroid.

Algorithm 1 Identification of cluster centroids algorithm in worker.

Input: RDD of molecules M having key <molecule_id, count of neighbours> and value [<neighbour_id, count of neighbours>, ..]

Output: list of molecule cluster centroids

```

1: Divide  $M$  into  $s$  splits:  $M_1, M_2, \dots, M_s$ 
2: procedure IDENTIFYCENTROIDS( $M_i$ )
3:    $valid\_clusters \leftarrow []$ 
4:    $invalid\_clusters \leftarrow []$ 
5:   repeat
6:     Call AssignCluster() on every element in  $M_i$            ▷ Algorithm 2
7:      $valid\_cluster\_centroids \cup$  molecules that are assigned to their own
       cluster
8:      $invalid\_cluster\_centroids \leftarrow valid\_cluster\_centroids.GetAllNeighbours()$ 
9:     Send  $invalid\_cluster\_centroids$  to master
10:     $all\_invalid\_cluster\_centroids \leftarrow$  Get all invalids from master node
11:     $M_i \leftarrow$  Remove entries having molecule_id present in either
        $all\_invalid\_cluster\_centroids$  or  $valid\_cluster\_centroids$ 
12:    Call UpdateNeighbours() on every element in  $M_i$        ▷ Algorithm 3
13:  until  $len(all\_invalid\_cluster\_centroids) == 0$ 
14:  return  $valid\_cluster\_centroids$ 

```

Algorithm 2 Assignment of cluster to a molecule.

Input: Key Value pair element *mol* having having key <molecule_id, count of neighbours> and value [<neighbour_id, count of neighbours>, ..]

Output: Same as Input with assigned cluster_id

```

1: procedure ASSIGNCLUSTER(mol)
2:   molecule  $\leftarrow$  mol.key
3:   m_id  $\leftarrow$  molecule.molecule_id
4:   m_count  $\leftarrow$  molecule.neighbour_count
5:   neighbours  $\leftarrow$  mol.value
6:   nbr_id  $\leftarrow$  neighbours[0].neighbour_id
7:   nbr_count  $\leftarrow$  neighbours[0].neighbour_count

8:   if m_count < nbr_count then
9:     molecule.cluster = nbr_id
10:  else if m_count == nbr_count AND m_id < nbr_id then
11:    molecule.cluster = nbr_id
12:  else
13:    molecule.cluster = m_id

```

Algorithm 3 Update of neighbour molecules

Input: Key Value pair element *mol* having key <molecule_id, count of neighbours> and value [<neighbour_id, count of neighbours>, ..]. Dictionary *all_invalid_cluster_centroids*

Output: Same as Input with removed neighbours

```

1: procedure UPDATENEIGHBOURS(mol, all_invalid_cluster_centroids)
2:   neighbours  $\leftarrow$  mol.value
3:   for each nbr  $\in$  neighbours do
4:     if all_invalid_cluster_centroids has key nbr.neighbour_id then
5:       Remove nbr from neighbours

```

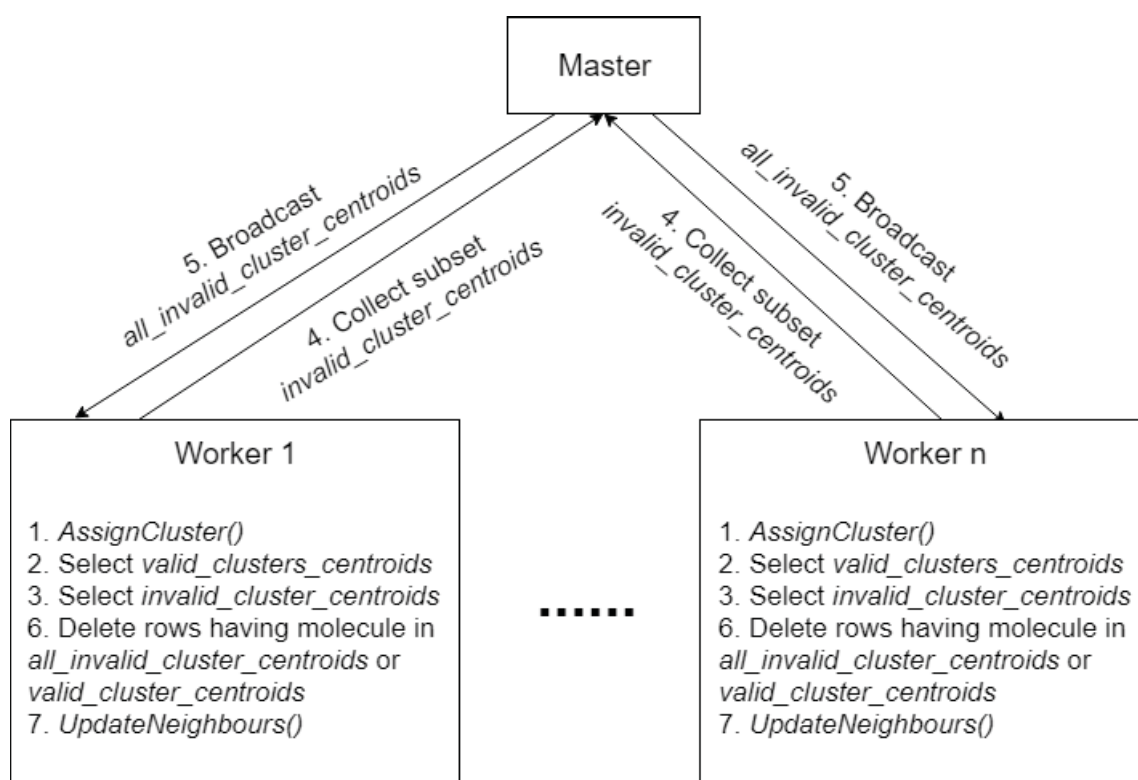


Figure 3.9: Distribution of Algorithm 1. Local computation is done on the workers from steps 1 -3. Then a synchronisation process is done in steps 4 and 5, followed by further local computation. This process is repeated until a condition is met.

Valid cluster centroids		Invalid cluster centroids							
	...								
23,14	17,10	75,9	54,8	25,6	63,6	44,5	42,4	68,4	...
75,9	23,14	17,10	54,8	19,7	27,6	44,5	42,4	22,2	90,1
54,8	23,14	75,9	25,6	68,4	89,4	87,3	15,3	21,2	
12,7	54,8	11,6	45,5	67,5	78,4	33,2	21,2		
63,6	23,14	43,5	88,4	55,2	65,2	76,1			
21,2	54,8	12,7							
	...								

(a) Molecules (represented using molecule id and count of neighbours) in gray and their list of neighbours

Valid cluster centroids		Invalid cluster centroids							
	...								
23,14	17,10	75,9	54,8	25,6	63,6	44,5	42,4	68,4	...
75,9	23,14	17,10	54,8	19,7	27,6	44,5	42,4	22,2	90,1
54,8	23,14	75,9	25,6	68,4	89,4	87,3	15,3	21,2	
12,7	64,8	11,6	45,5	67,5	78,4	33,2	21,2		
63,6	23,14	43,5	88,4	55,2	65,2	76,1			
21,2	54,8	12,7							
	...								

(b) Assign cluster centroid to each molecule

Valid cluster centroids		Invalid cluster centroids							
23		75, 54, 63, 44, 42, 17, 25, 68							
	...								
23,14	17,10	75,9	54,8	25,6	63,6	44,5	42,4	68,4	...
75,9	23,14	17,10	54,8	19,7	27,6	44,5	42,4	22,2	90,1
54,8	23,14	75,9	25,6	68,4	89,4	87,3	15,3	21,2	
12,7	64,8	11,6	45,5	67,5	78,4	33,2	21,2		
63,6	23,14	43,5	88,4	55,2	65,2	76,1			
21,2	54,8	12,7							
	...								

(c) Select all molecule which have been assigned to their own cluster

Valid cluster centroids		Invalid cluster centroids							
23		75, 54, 63, 44, 42, 17, 25, 68							
	...								
23,14	17,10	75,9	54,8	25,6	63,6	44,5	42,4	68,4	...
75,9									
54,8									
12,7		11,6	45,5	67,5	78,4	33,2	21,2		
63,6									
21,2		12,7							
	...								

(d) Remove neighbours of cluster centroids from the working set

Valid cluster centroids		Invalid cluster centroids							
23		75, 54, 63, 44, 42, 17, 25, 68							
	...								
23,14	17,10	75,9	54,8	25,6	63,6	44,5	42,4	68,4	...
75,9									
54,8									
12,7		11,6	45,5	67,5	78,4	33,2	21,2		
63,6									
21,2		12,7							
	...								

(e) Assign cluster centroid to remaining molecules

Valid cluster centroids		Invalid cluster centroids							
23, 12		75, 54, 63, 44, 42, 17, 25, 68, 11, 45, 67, 78, 33, 21							
	...								
23,14	17,10	75,9	54,8	25,6	63,6	44,5	42,4	68,4	...
75,9									
54,8									
12,7		11,6	45,5	67,5	78,4	33,2	21,2		
63,6									
21,2		12,7							
	...								

(f) Select all molecule which have been assigned to their own cluster

Valid cluster centroids		Invalid cluster centroids							
23, 12		75, 54, 63, 44, 42, 17, 25, 68, 11, 45, 67, 78, 33, 21							
	...								
23,14	17,10	75,9	54,8	25,6	63,6	44,5	42,4	68,4	...
75,9									
54,8									
12,7		11,6	45,5	67,5	78,4	33,2	21,2		
63,6									
21,2									
	...								

(g) Remove neighbours of cluster centroids from the working set

Figure 3.10: Identification of cluster centroids

3.6 DLSH-Butina

The second approach implemented was that of calculating neighbours using an approximation technique. Neighbours identification is the most time consuming process in the Butina clustering algorithm, hence approximation was implemented as a way of increasing the speed of this process. LSH was used as an approximation method as it was already applied to the process of molecules similarity searching by [66] and [45] with successful results, providing better speed performance. However it does not provide exact neighbours. A high level view of the process can be seen in Figure 3.11. The implementation of this method is also done in a distributed way with the aim of handling even larger datasets.

The preprocessing stage for DLSH-Butina also involves reading SMILES data, indexing it and dividing it into equally sized splits $D_1, D_2 \dots D_n$. The splits are then distributed among the workers, which in turn convert the SMILES data into fingerprints.

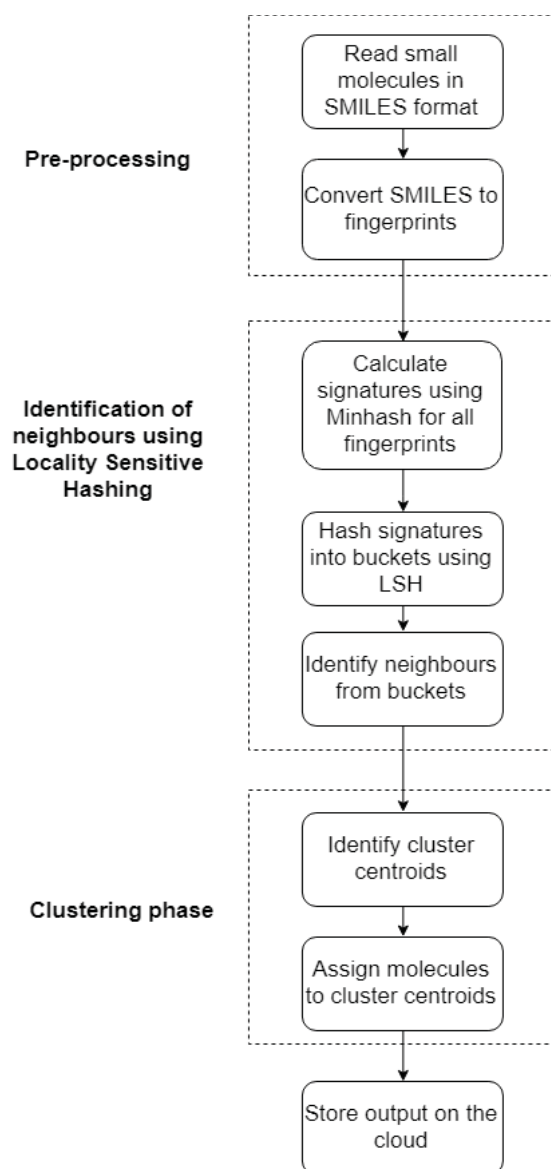


Figure 3.11: High level diagram of the DLSH-Butina process. The pre-processing is required by all the clustering algorithms, while the identification of neighbours phase is the implemented section introduced in this algorithm.

3.6.1 Locality Sensitive Hashing

LSH implementation was based on the approach by [45], where it was adapted to be run in a distributed manner using Apache Spark. The implementation is divided into three parts:

1. Creation of Minhashes: Minhash is a process where high dimensional data is hashed to reduce its dimensionality, enabling faster estimation of Jaccard similarity when compared to using the complete bit fingerprint.
2. LSH: Minhash output are hashed and placed in buckets with collisions in buckets being considered as neighbours.
3. Identification of Neighbours: Potential neighbours of a query molecule are obtained by retrieving molecules hashed to the same buckets as the query molecule.

To perform Minhash, a list is created enumerating all the numbers starting from 0 up to fingerprint length - 1, representing all the bit positions of a fingerprint. A list P_x having x permutations of the bit positions of a fingerprint is created. Each entry represents one permutation of bit positions. List P_x is shared among all the worker nodes in the Spark cluster such that it is common to all. Fingerprints are then divided into approximately equal-sized splits F_1, F_2, \dots, F_n and distributed among the workers, with each worker getting a different and unique split. Hashing is then performed on each fingerprint.

Hashing of a fingerprint is performed by ordering the fingerprint bits according to each entry in P_x . Once a bits in the fingerprint are ordered, the position of the first '1' bit is stored in a list of signatures S_i . This is done for every fingerprint, where each fingerprint is ordered once for every entry in P_x . The output is a list of signatures S for every fingerprint. The hashing process for one fingerprint is shown in Figure 3.12.

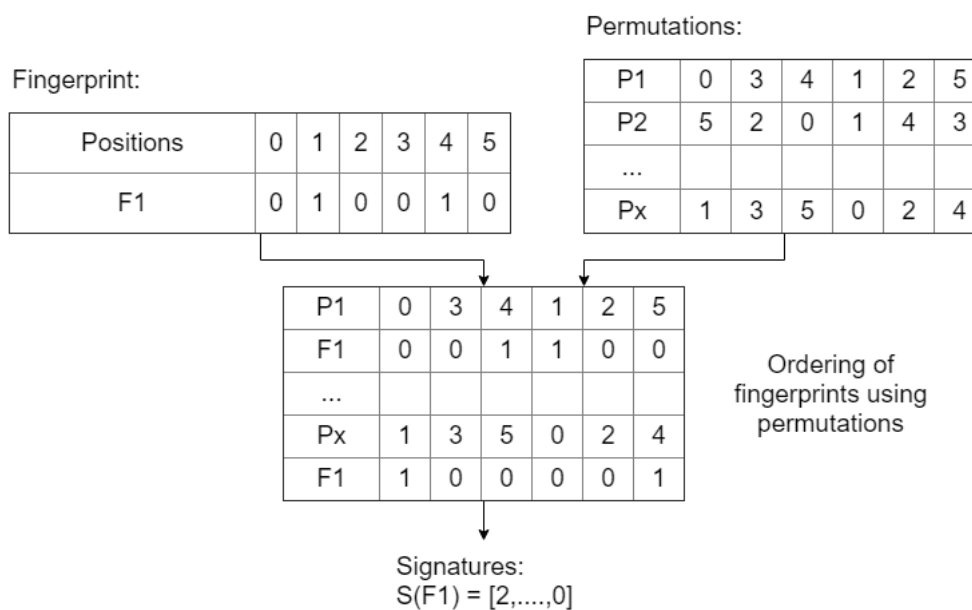


Figure 3.12: The Minhash process is shown. In this example, one fingerprint and a list of permutations P are provided as input. The fingerprint bits are ordered according to P . For each permutation, the position of the first 1 bit is recorded and stored in a list of signatures S . As seen in the example, ordering fingerprint 1 according to permutation 1, results in position 2 being the first 1 bit. Image adapted from [45].

Locality Sensitive Hashing hashes the signatures into buckets such that molecules hashed in the same dictionary bucket would be considered as neighbours. This creates indexes based on hashes such that fast retrieval of neighbour molecules can be done. Each n number of elements in a signatures S_i are grouped together into b bins, such that bn is equal to the length of S_i . b is a user defined threshold and affects the the similarity of objects in the buckets. A large value of b results in a small value of n , allowing low similarity molecules to be hashed into the same bin. A larger n would require two molecules to have multiple signatures that are similar to each other. Therefore non similar molecules has less probability of having a large number of similar signatures and hence hashing into the same bucket.

The bin elements are then hashed into a single value, as can be seen in Figure 3.13. The hash function used is one that does not create a unique value for

every different input, but it hashes similar inputs into the same value. The operation left shift binary conversion was used, which combines n values into a single number [45]. The hash results are then used to create two dictionaries. The first dictionary is *buckets_dictionary*. It uses the hash result as a key and the value being a list of molecule identifiers that hash to that value. A second dictionary, called *molecule_hashes*, is created using the molecule identifier as a key and the list of hash results as a value, as seen below:

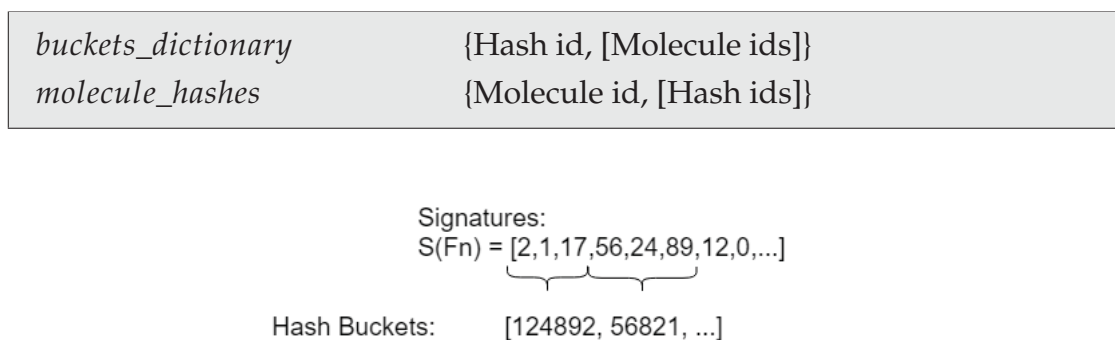


Figure 3.13: n number of signature elements are grouped and hashed into a single value.

The *molecule_hashes* dictionary is used to get the list of buckets that a molecule hashes into, in a single operation. The values are then used in the *buckets_dictionary*, where each key is used to access the list of molecules that hash into the same buckets.

In a distributed environment, fingerprint F are already divided among the workers for the minhash process. Therefore the resultant signatures S belonging to those fingerprints are also divided. LSH is then performed on each separate worker, with separate dictionaries being created. The dictionary *buckets_dictionary*, results in being distributed among all the workers, with values for the same key being also distributed. Therefore a merging process is required, combining all the values for the same key. *buckets_dictionary* is collected and the combined result is then broadcasted to all the workers.

The final step is to identify the neighbours of a query molecule m . This is done in a distributed manner by using the two dictionaries created. First, the dictionary *molecule_hashes* is used to obtain the buckets that m is in. Then

buckets_dictionary is used to obtain the list of molecules that hash to each of those bucket positions. This list represents molecules that hash to the same bucket and therefore are potential neighbours. Finally, similarity calculation using threshold T is done for m against each molecule obtained from the buckets. This removes any non similar molecules that were incorrectly hashed to the same bucket. For our case, quality of clusters is important therefore the step of calculating similarity was included to remove any incorrect results. Although similarity calculation is still required, each molecule is only compared to those molecules having a potential of being neighbours, rather than all the other molecules, thus reducing $O(N^2)$ complexity. The complete flow of the LSH process can be seen in Figure 3.14, showing the computation done on each worker and the communication required with the master node. The format of the final output is identical to the one produced from actual neighbour identification in Section 3.5.1.

3.6.2 Clustering Phase

The final step is to perform the clustering process. The process remains unchanged from that explained in Section 3.5.2. Since the output of the neighbours is identical in both actual neighbour identification and LSH, no difference in implementation is required. Cluster centroids are first identified, followed by assignment of molecules in clusters.

3.7 Chapter Summary

This chapter presented the implementation details of approaches designed to target the problem at hand. It highlighted the problems and issues that were targeted with the solutions implemented. The environment and datasets used were also discussed to provide a holistic view of the solution.

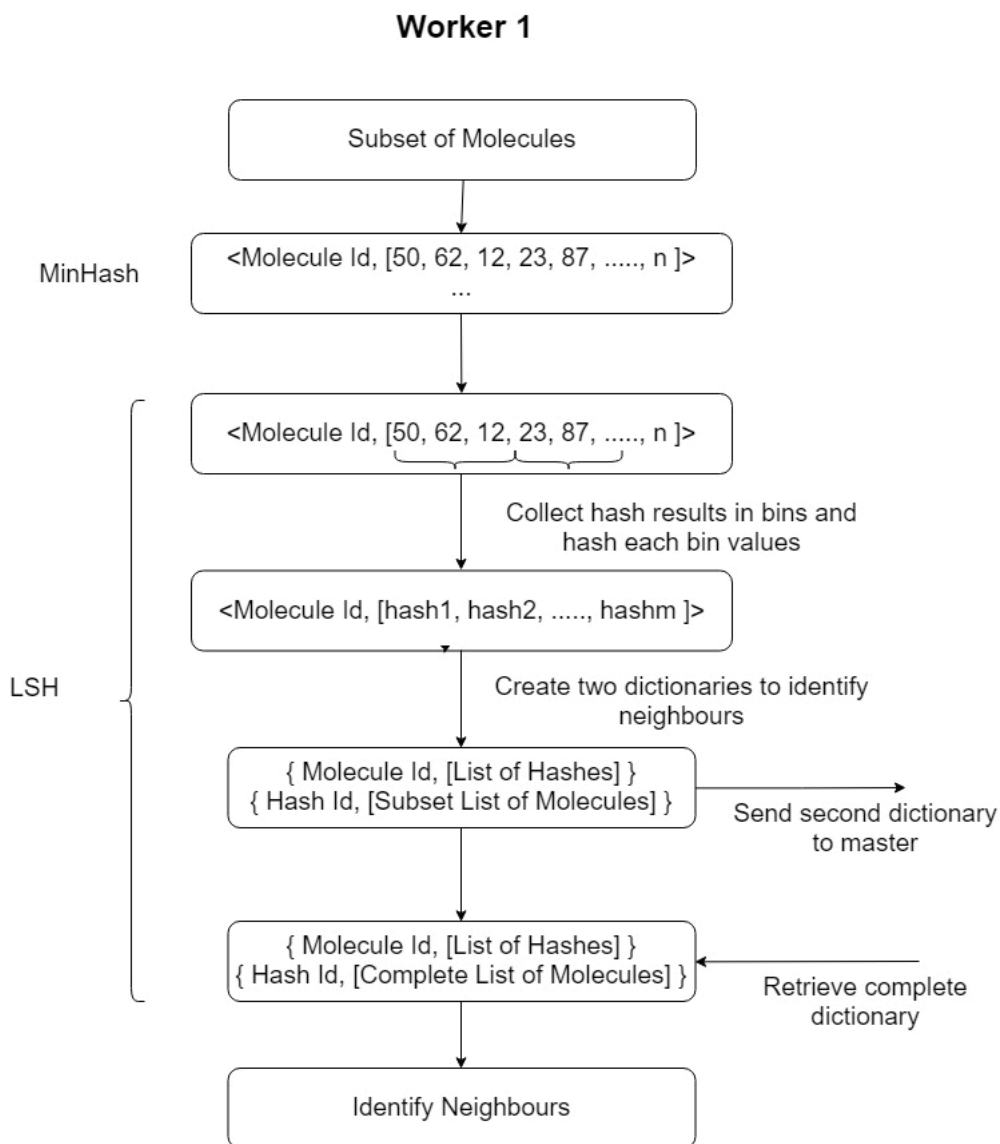


Figure 3.14: Process of identifying neighbours using LSH method, showing the execution of one worker and its communication requirements with the master node.

Results & Evaluation

This chapter first details the results of the experiments performed to obtain the optimal parameters for our approaches (implemented in Section 3) to run the evaluation. This is followed by an evaluation of the approaches and a discussion of the results obtained.

4.1 Experiments Design

A number of experiments were designed throughout this study to enable informed decisions. The experiments were focussed around the aim proposed in Chapter 1, that of implementing a distributed clustering algorithm to increase the speed performance of clustering while maintaining good quality of clusters produced by having active molecules grouped together.

The algorithms evaluated are D-Butina and DLSH-Butina. The decision was taken after a number of serial approaches were compared in Section 3.1, with the selected algorithm being distributed and approximation applied to the process. Initial experiments were performed to determine the optimal parameters for the D-Butina and DLSH-Butina approaches.

Tests were then performed on a distributed environment to evaluate the speed, scalability and performance accuracy of the approaches presented. The distributed approaches implemented in Chapter 3 were compared to the serial version to determine the speedup gain while also analysing the scalability of the algorithms. The scalability and performance accuracy results produced

were then analysed and compared with bisecting k-means, an algorithm that has good scalability and therefore can handle large datasets [38].

[38] found that clusters generated by Ward's clustering are better than bisecting k-means in terms of Sum of Squared Error. However results by bisecting k-means have similar standard deviation to results produced by Ward's clustering. Additionally, the algorithm has the advantage of better scalability especially for large datasets. Therefore it was used to compare the speed and results with the approaches implemented.

Throughout all the experiments, extended connectivity fingerprint (see Section 2.2.3) from the RDKit library was used as molecular representation. These are a type of circular fingerprints, with the implementation being based on Morgan's algorithm. Circular fingerprints were chosen since these have been widely applied in research with positive results to identify molecular activity [13, 22, 67]. Similarity between fingerprints was handled by RDKit using the Tanimoto similarity metric, with this metric being a standard in comparing descriptors [26].

4.2 Cloud Infrastructure

Experiments performed on the distributed versions of D-Butina and DLSH-Butina were done using a Spark cluster on Azure Batch¹. Batch was selected as it is cost efficient with respect to other providers, allowing the use of low priority VMs to save further costs. It further provides the functionality of ready made Spark enabled Docker² containers, which can be extended to install the required libraries.

To perform the tests, six and eleven nodes were used, having one master node and the remaining as workers. E2_v3 memory intensive instances were employed, with the following specifications for each node:

- 2.3 GHz Intel XEON E5-2673 v4 (Broadwell) processor
- 2 Cores

¹<https://azure.microsoft.com/en-us/services/batch/> (last accessed 25th May 2019)

²<https://www.docker.com/> (last accessed 25th May 2019)

- 16Gb RAM
- 50Gb Temporary Storage
- Ubuntu 16.04 LTS

A Docker container was created with the required installed libraries. It was based on the image available by aztk, aztk/spark:v0.1.0-spark2.3.0-base, having all the necessary Spark libraries installed. Additional libraries were then installed and current ones updated to the necessary versions. Anaconda version 5.2.0 was installed to handle modules installed. Python version 2.7.6³ was used, with Apache Spark version 2.3.0⁴ being selected. Hadoop library 2.7.0 was installed, and two required jar files azure-storage-2.0.0.jar and hadoop-azure-2.7.0.jar being copied to SPARK_HOME. RDKit library version 2016.03.04⁵ was finally installed to enable handling of molecular data, conversions between the formats and similarity comparisons.

Interaction with the Azure Spark cluster was done using the Azure Distributed Data Engineering Toolkit⁶. This allows to easily interface with Batch to setup Spark clusters, deploy jobs and monitor the activity through Spark UI.

4.3 Evaluation Dataset

Due to time and resource limitations, subsets from the ZINC (Section 3.4) dataset were used to create the evaluation datasets shown in Table 4.1. The subsets were then combined with actives from DUD-E (Section 3.4) dataset to determine the ability of the implemented approaches to cluster active molecules together. Four datasets were created to evaluate the speed of the approaches presented.

Considering that the research focuses on large datasets, a decision was taken to keep file sizes as small as possible by opting to represent the data as SMILES

³<https://www.python.org/download/releases/2.7.6/> (last accessed 25th May 2019)

⁴<https://spark.apache.org/releases/spark-release-2-3-0.html> (last accessed 25th May 2019)

⁵https://github.com/rdkit/rdkit/releases?after=Release_2016_03_04 (last accessed 25th May 2019)

⁶<https://azure.microsoft.com/en-us/blog/on-demand-spark-clusters-on-docker/> (last accessed 25th May 2019)

Dataset Name	No. Molecules	Size (Megabytes)
<i>Dataset1</i>	50,000	2.5
<i>Dataset2</i>	100,000	5.27
<i>Dataset3</i>	200,000	11.63
<i>Dataset4</i>	500,000	32.13

Table 4.1: Characteristics of evaluation datasets

string rather than SDF. This increased the efficiency in storage since each molecule is only represented by a short string as described in Section 2.1. In comparison, storing 2D and 3D encoded information of 500,000 molecules as SDF format would have resulted in approximately 2.4Gb.

For quality evaluation, *Dataset4* was combined with ABL1 active molecules obtained from DUD-E dataset, removing any duplicate actives from the ZINC subset in the process. *Dataset4* was clustered to determine how the different approaches selected above group the active molecules together.

4.4 Results

Parameters for D-Butina and DLSH-Butina were required to evaluate the performance of the two approaches. D-Butina requires the similarity threshold as explained in Section 2.3.1, while DLSH-Butina requires the amount of signatures to hash together when creating buckets.

4.4.1 Selecting the Butina Similarity Threshold

The parameter required for the D-Butina clustering algorithm is the similarity threshold γ . It represents the neighbourhood boundary of the cluster centroids. An experiment was done to determine the optimal value of γ to separate activity classes among the dataset. To determine γ , results obtained from serial Butina clustering were analysed. Serial Butina was used since the results of the distributed approach D-Butina are identical to the serial version.

Serial Butina clustering was performed on three datasets, ABL1, THB and Renin inhibitors as explained in Section 3.1.1. For each dataset, clustering was

run for $\gamma = 0.1$ up to 0.9 at 0.1 increments. Increments of 0.1 were chosen as a research by [70] found that 0.3 and 0.8 were optimal parameters for two different fingerprints. Hence increments of 0.1 are a good fit to most molecular descriptors. Quality Partition Index (QPI) and F-measure results were computed, plotted in Figures 4.1 and 4.2, and analysed.

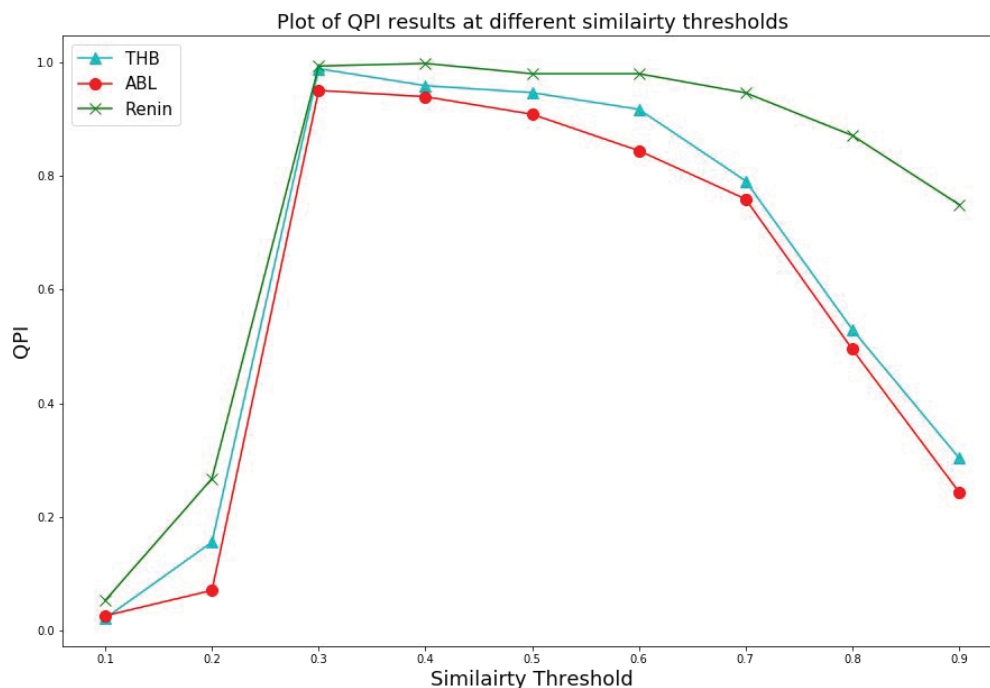


Figure 4.1: Analysis of the QPI results for parameter estimation of the Butina clustering approach.

Figure 4.1 shows the QPI results obtained at increasing similarity thresholds. The highest results are obtained between the range of 0.3-0.4 similarity. QPI results then tend to decrease gradually as similarity is increased. The results are similar to those obtained by [70], where it was found that the optimum value of similarity using ECFP4 fingerprints was at 0.3. They concluded that the likelihood for two same activity molecules to have at least a similarity of 0.3 was hundreds of times higher than the similarity of an active and a random compound.

F-measure results were plotted in Figure 4.2 to analyse the generated clusters. Precision, Recall and F-measure were calculated for each generated cluster

and we used the highest scores in accordance with [53]. The results are shown in Tables 4.2- 4.4. As can be seen in Figure 4.2, the highest F-measure values were also obtained at a threshold of 0.3 similarity for the three datasets selected, similar to the QPI. It is worth noting that the precision value is 1.0 for $\gamma \geq 0.3$, showing that the clusters contain only active molecules. However, the recall values differ among the datasets. At a threshold of $\gamma=0.3$, the recall value for Renin inhibitors was 0.739, while that for THB and ABL1 were 0.548 and 0.254 respectively.

Possible reasons for the large difference in Recall values among the datasets maybe be related to the distribution of actives for a target protein and the distribution of the known actives. For ABL1, the actives may be more spread out. Therefore smaller clusters of actives are created. Additionally the actives that have been identified for Renin and THB may be more similar to each other compared to ABL1 which are more widespread. Another possibility could be that DUD-E dataset may have only a subset of all the actives for ABL1. Therefore other molecules within the range of active centroids, that are not yet identified, may be missing altogether from the dataset. In fact the number of actives for Renin is larger than that for ABL1, having 387 actives when compared to 272 actives for ABL1. However this does not justify the higher result achieved by THB, where only 168 actives are available.

In comparison to the QPI results, F-measure results decrease at a faster rate when the similarity threshold is increased. This is affected by the size of the created clusters. In fact, the recall rates at $\gamma=0.4$ drop to 0.54, 0.33, 0.15 for Renin inhibitors, THB and ABL1 respectively.

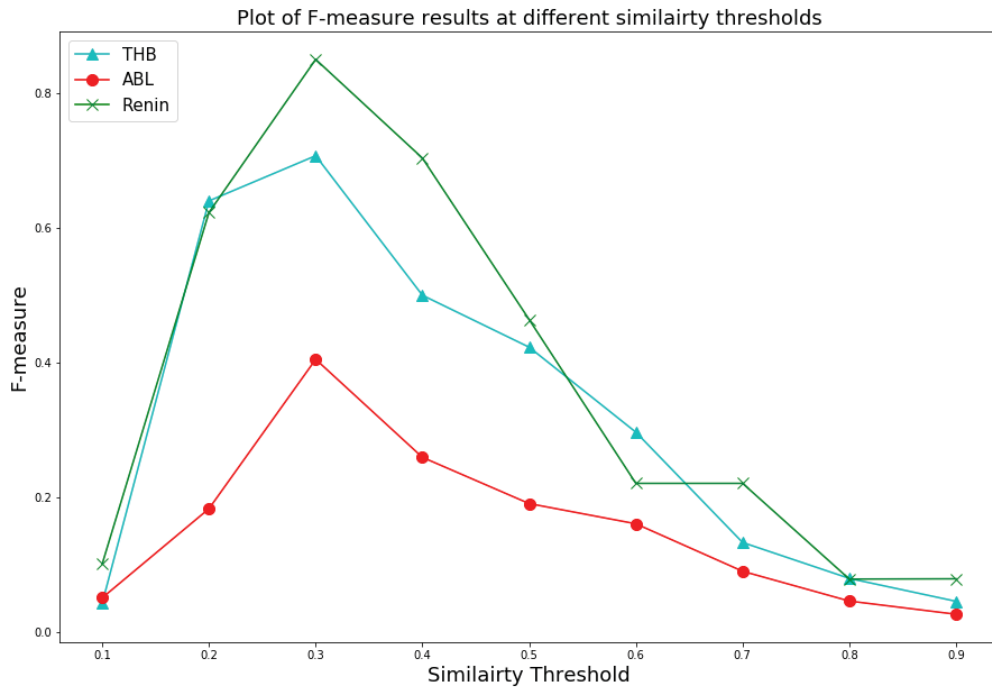


Figure 4.2: Analysis of the F-measure results for parameter estimation of the Butina clustering approach.

Dataset	Threshold	Precision	Recall	F-measure
Renin	0.1	0.05	1	0.10
	0.2	0.94	0.47	0.62
	0.3	1	0.74	0.85
	0.4	1	0.54	0.70
	0.5	1	0.30	0.46
	0.6	1	0.12	0.22
	0.7	1	0.12	0.22
	0.8	1	0.04	0.08
	0.9	1	0.04	0.08

Table 4.2: Precision, Recall and F-measure values for different thresholds of Butina algorithm on the Renin dataset.

Dataset	Threshold	Precision	Recall	F-measure
THB	0.1	0.02	1	0.04
	0.2	0.81	0.53	0.64
	0.3	1	0.55	0.71
	0.4	1	0.33	0.50
	0.5	1	0.27	0.42
	0.6	1	0.16	0.29
	0.7	1	0.07	0.13
	0.8	1	0.04	0.08
	0.9	1	0.02	0.05

Table 4.3: Precision, Recall and F-measure values for different thresholds of Butina algorithm on the THB dataset.

Dataset	Threshold	Precision	Recall	F-measure
ABL1	0.1	0.03	0.98	0.05
	0.2	0.94	0.10	0.18
	0.3	1	0.25	0.41
	0.4	1	0.15	0.26
	0.5	1	0.11	0.19
	0.6	1	0.08	0.16
	0.7	1	0.05	0.09
	0.8	1	0.24	0.05
	0.9	1	0.01	0.02

Table 4.4: Precision, Recall and F-measure values for different thresholds of Butina algorithm on the ABL1 dataset.

4.4.2 Locality Sensitive Hashing (LSH) Parameters

As described in Section 2.3.3, LSH requires signatures S to be grouped into b bins such that each bin contains h signatures that are hashed into a value to represent a bucket. The value h affects the similarity values that are hashed to the same buckets.

The number of signatures S was selected to be a value around 100. This value determines the dimensionality of S , with larger values requiring longer computation times. However, a value around 100 is representative enough for most fingerprints when considering the redundancy and correlation in the bits of fingerprints [66].

The aim of using LSH is to correctly place molecules in buckets that have pairwise similarity $\gamma \geq 0.3$. In an ideal scenario, no two molecules having $\gamma < 0.3$ are placed in the same bucket. The probability of two objects being hashed into the same bucket is determined by the value h . By hashing multiple signatures together, the chance of two molecules having a low similarity, getting hashed together is reduced. Therefore as h increases, the molecules placed in the same bucket have higher similarity values.

To identify the optimal parameter, three values of h were compared. The chosen values for comparison are h_2 , h_3 and h_4 , with the percentage of correctly identified molecules at every similarity level being shown in Table 4.5. Initially, the complete similarity matrix of the test dataset was computed, and each result being grouped at 0.1 intervals such that a count of all the similarity values was obtained.

LSH was then applied on the same dataset using different h values. For each value, the similarity of every molecule with its potential neighbours was calculated. In LSH, potential neighbours are those molecules that collide with a molecule in at least one bucket.

Once the similarities were calculated, these were compared with the actual count of similarities obtained to determine the effect LSH has in reducing low similarity collisions.

		h_2	h_3	h_4
Similarity Range	# Actual Similarities	% identified	% identified	% identified
0.0 - 0.1	26,967,186	0.85	0.09	0.00
0.1 - 0.2	92,422,870	0.92	0.21	0.01
0.2 - 0.3	4,924,396	0.97	0.44	0.03
0.3 - 0.4	441,460	0.99	0.79	0.13
0.4 - 0.5	154,148	0.99	0.95	0.33
0.5 - 0.6	43,928	1.00	0.99	0.67
0.6 - 0.7	17,600	1.00	1.00	0.88
0.7 - 0.8	6,302	1.00	1.00	0.98
0.8 - 0.9	1,876	1.00	1.00	0.99
0.9 - 1.0	1,454	1.00	1.00	1.00

Table 4.5: Percentage of the similarity matrix, grouped at intervals, that is identified for each value of h signatures hashed together.

From the results shown in Table 4.5, $h = 3$ provided the most optimal balance of reducing computation and memory costs. This was achieved by eliminating low similarity collisions while also keeping an acceptable percentage of molecule similarities having $\gamma \geq 0.3$. At this value, only 9% of the 26,967,186 similarities in the similarity matrix having a threshold between 0 and 0.1 were incorrectly hashed in the same buckets. However when $h = 3$ it managed to identify 79% of the total similarities for $0.3 < \gamma \leq 0.4$, and 95% for $0.4 < \gamma \leq 0.5$. $h = 4$ filters out a large number of valid similarities at $0.3 < \gamma \leq 0.6$. When $h = 2$ the similarities are reduced by a minimal amount, thus having little effect on the computation time of neighbour calculation.

Therefore $h = 3$ was selected for LSH. The signatures S need to be grouped into groups of 3. However, since $S = 100$ is not divisible by 3, $S = 102$ was chosen as it is the nearest divisible value larger than 100. This means that an additional 2 hash value need to be added to the signatures during the minhash process.

4.5 Evaluation

We first evaluated the efficiency of the approaches by considering speed and scalability. This was followed by the quality of the clusters produced by evaluating the ability of the clustering algorithms to separate active molecules from non active ones.

4.5.1 Clustering Efficiency

D-Butina and DLSH-Butina were compared with the serial version of the Butina clustering algorithm, using the RDKit implementation, to determine the effect of applying distribution and approximation on the clustering efficiency. Additionally, the approaches are also compared to the results produced by bisecting k-means. Although bisecting k-means was not applied on a wide range of studies in clustering small-molecules datasets, it can be applied to large datasets because of its scalability [38]. The mllib⁷ library of Apache Spark was used for the bisecting k-means algorithm. The implementation is based on the approach proposed by [37]. The value of the parameters were kept as default apart from the value of k . It is a user defined value used to determine the number of clusters required as output from the bisecting k-means approach and therefore serves as a stopping criteria. The value of k was chosen by selecting the same number of clusters generated by Butina algorithm at the threshold selected.

Considering differences in the experimental setup, the experiments performed cannot be directly compared to results from literature with respect to quality. This is mainly due to variables in the research, being either molecular representations, dataset used and the hardware used to run the systems. Additionally, some algorithms of interest did not have their implementation available, hence could not be used as part of the evaluation.

To record the performance of the algorithms, time of the computation was calculated from the point of reading the molecules data from the cloud storage up to assigning a cluster to each molecule. The sections where the data is con-

⁷<https://spark.apache.org/docs/latest/mllib-clustering.html#bisecting-k-means> (last accessed 25th May 2019)

verted into specific formats, prepared to be written to persistent storage and the actual process of writing the data was not considered for speed purposes. Additionally for similarity calculation, all the Butina based approaches were implemented to make use of optimised bulk similarity calculation as implemented by RDKit. Bisecting K-means uses an internal implementation for similarity calculations using a custom parametrised similarity function.

To get a complete picture of speed and scalability of the presented approaches, tests were run on the four datasets explained in Section 4.3 using two different Azure Batch Spark cluster configurations. The approaches were first run on six nodes, with five workers and one master node. Then the same datasets were again run on eleven nodes, with ten workers and one master node. This provides a clearer picture how the approaches scale with respect to data and nodes, enabling a comparison among the approaches and individually as the size of the dataset is increased. Tables 4.6 - 4.8 show the running time results of the implemented approaches D-Butina and DLSH-Butina compared to Serial Butina algorithm, and the time taken for bisecting k-means. The data is then shown graphically in Figures 4.3 and 4.4.

		Serial Butina	D-Butina	
	Dataset	Execution Time (minutes)	Execution Time (minutes) 5 workers	Execution Time (minutes) 10 workers
1	50,000	12	9	5
2	100,000	58	25	15
3	200,000	277	102	62
4	500,000	1650	682	414

Table 4.6: Showing the time taken in minutes by D-Butina algorithm to cluster the datasets compared to the Serial Butina approach.

Compared to the serial implementation, D-Butina has a speedup of 2.40 on *Dataset4* when using five worker nodes. The speedup is increased to 3.98 when using ten worker nodes. Having less speedup than the number of nodes was expected considering that distributed systems have overhead costs, requiring

		Serial Butina	DLSH-Butina	
	Dataset	Execution Time (minutes)	Execution Time (minutes) 5 workers	Execution Time (minutes) 10 workers
1	50,000	12	4	3
2	100,000	58	8	5
3	200,000	277	49	23
4	500,000	1650	401	197

Table 4.7: Showing the time taken in minutes by DLSH-Butina algorithm to cluster the datasets compared to the Serial Butina approach.

		Serial Bisecting k-means	Bisecting k-means	
	Dataset	Execution Time (minutes)	Execution Time (minutes) 5 workers	Execution Time (minutes) 10 workers
1	50,000	18	13	8
2	100,000	33	20	18
3	200,000	66	40	38
4	500,000	156	90	96

Table 4.8: Showing the time taken in minutes by Bisecting k-means algorithm to cluster the datasets.

distribution of the data that negatively effect performance. This is shown in Amdahl's law that states that parallel speedup using P processors cannot reduce the time taken for a serial fraction f of the algorithm, therefore the expected speedup is given by $Speedup = \frac{1}{f + \frac{1-f}{P}}$ [71]. Therefore in this case, it is expected to not achieve a speedup of 5.00 when using five workers. The difference between nodes and speedup is also present in similar results found in literature, where a distributed Ward's clustering by [61] using Map Reduce achieved a speedup of 1.19 when using three mapper tasks and speedup of 2.40 when using six mapper tasks compared to a serial implementation. Similarly a Spark implementation of a hierarchical clustering algorithm by [51] also achieved speedup that is not of



Figure 4.3: Comparison of the speed performance in logarithmic scale of the algorithms on the four datasets when using 5 worker nodes.

the same magnitude of the number of cores used, achieving a speedup between 200 and 310 on 392 cores depending on the dataset and vector dimensions representing the data. DLSH-Butina achieved better speedup, with a speedup of 4.11 when using five workers and 8.37 when using ten workers. This results are promising as speedup increased when more nodes were added to the cluster. Large performance improvements were also noted by [66] when implementing LSH for Jarvis-Patrick algorithm. A speedup of approximately 76 was obtained when LSH was used in conjunction with a Jaccard coefficient > 0.98 . Overall, bisecting k-means obtained the most optimal speedup on *Dataset4*, achieving approximately a speedup of 18 using both five and ten workers.

In terms of scalability, bisecting k-means is the most optimal algorithm as it scales approximately in a linear way. However it was noticed that increasing the workers did not increase the performance of the algorithm. This merits further investigation to determine the reason of not having performance gains

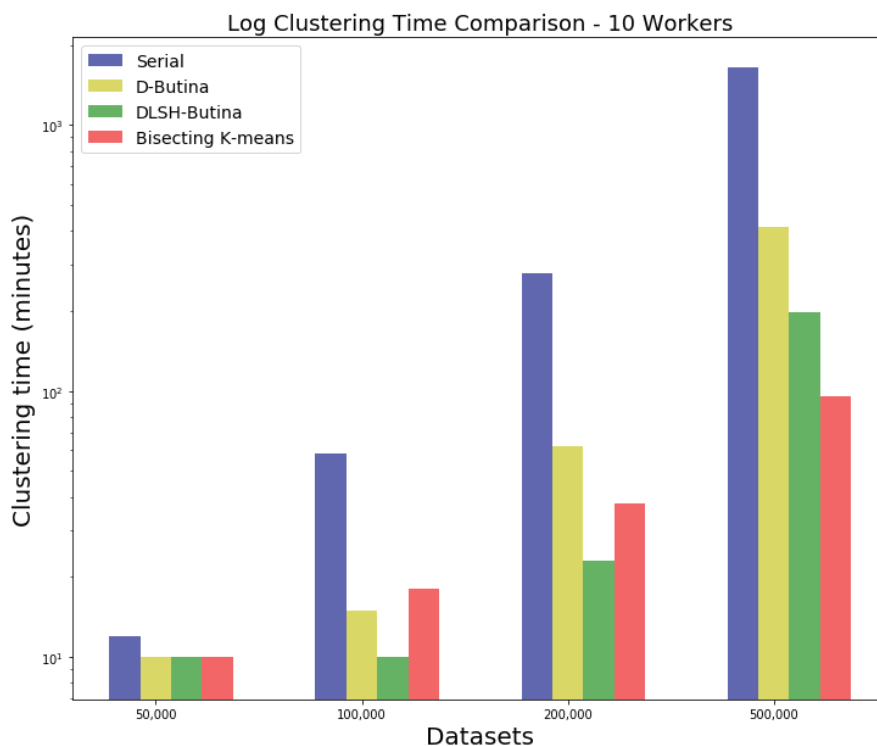


Figure 4.4: Comparison of the speed performance in logarithmic scale of the algorithms on the four datasets when using ten worker nodes. Time taken is shown on a logarithmic scale.

when adding more workers. Additionally, bisecting k-means has a performance gain of 87% and 78% when compared with the results obtained by D-Butina and DLSH-Butina on *Dataset4* using five workers. This is substantial considering that bisecting k-means was performing worse than both D-Butina and DLSH-Butina on smaller dataset sizes.

Runtime scalability for D-Butina and DLSH-Butina starts at less than $O(N^2)$. One reason for this may be the data transfer overhead across the nodes, hiding the quadratic complexity. However as can be seen between the readings for 200,000 and 500,000 molecules, the scalability was higher than the expected magnitude of 6.25 increase. One of the reasons for this is the data shuffling among the nodes. This is an expensive process that negatively affects the performance as more data is added. Additionally, the low threshold of neighbourhood further reduces the performance of the algorithm as each molecule has more neigh-

bours, hence requiring handling of more data during computation and transfer of data.

As expected, DLSH-Butina has a better scalability and speedup than D-Butina since it requires less similarity comparisons, computing only those that collide in the same bucket as results have shown in Table 4.5. Additionally, some actual neighbours are not hashed into the same buckets, hence molecules would have less neighbours. This reduces the amount of data to distribute albeit lower clustering quality.

Automatically Spark shuffles/transfers the data among the workers in processes of broadcast (see Section 3.3), when data is aggregated among the nodes and when a request for some data cannot be satisfied with the data present on that particular worker. With regards to data shuffling among nodes in the process of clustering the data, bisecting K-means is the most efficient with 400 Mb and 500 Mb data transferred to cluster 200,000 and 500,000 molecules respectively. On the other hand, DLSH-Butina transfers 900Mb and 7 Gb of data while D-Butina transfers 1 Gb and 9 Gb of data respectively for 200,000 and 500,000 molecules. Although this data does not reside in memory all at once, the transfer of this data negatively affects the performance of the algorithm. This shows that scalability of the transfers of the data is not efficient for D-Butina and DLSH-Butina.

4.5.2 Clustering Quality Results

The quality of the cluster result produced is analysed to determine the ability of the algorithms in separating active molecules from putative inactive ones. Output files from the dataset, mentioned in Section 4.3, were obtained for D-Butina, DLSH-Butina and bisecting k-means.

Unfortunately the activity data of the whole dataset is not known, therefore molecules whose activity is not known cannot be considered as inactives. To determine molecular activity against a target, a number of lab experiments are required to manually test all the molecules with respect to a target protein. Nonetheless, clustering could be performed to analyse the clusters created and investigate the grouping of known active molecules [54, 35].

For the purpose of analysing the results obtained, an active cluster is considered as one having the percentage of active molecules in the cluster greater than the percentage of actives in the whole dataset [13]. As discussed in Section 2.6.1, actives which are incorrectly grouped with non active clusters or clustered as singletons are considered as *FalseNegatives*, or the combination of r and s in the calculation of the QPI value. Inactive molecules that are clustered in active clusters are *FalsePositives*, however since no dataset contain the actual activity data of a large dataset, this cannot be known. Therefore *FalsePositives* will be used in the context of unknown activity molecules being clustered within active clusters as done by [54]. These will then be analysed, as they may provide an insight on additional small molecules that may be active with respect to the target protein selected.

Given enough time and resources, the molecules whose activity is not known in active clusters may be tested experimentally for their actual activity. This would show whether unknown activity molecules clustered with active molecules would be active with respect to a target protein or not.

Table 4.9 shows the results obtained for the two approaches implemented, D-Butina and DLSH-Butina with results for bisecting k-means. The results for serial Butina were not included since D-Butina and Butina produce the same output.

Approach	D-Butina	DLSH-Butina	Bisecting k-means
No. clustered ABL1 inhibitors	257	242	272
False Negatives	15	30	0
False Positives	315	98	5838
% ABL1 molecules included in active clusters	94.4	88.9	100
No. Active Clusters	36	44	76

Table 4.9: Comparison of clustering results obtained using the two approaches proposed compared to Bisecting k-means.

D-Butina managed to cluster more active molecules compared to DLSH-

Butina with 257 molecules being successfully clustered in active clusters. 14 active molecules were clustered as singletons, with 1 active molecules being clustered in an unknown activity cluster. In comparison to this, DLSH-Butina, had more singleton clusters. However D-Butina clustered 315 unknown activity molecules within active clusters, while DLSH-Butina had only 98 unknown activity molecules clustered within active clusters. This merits further investigation to determine the similarity of unknown activity molecules to the active molecules in the same cluster, as potentially some of these molecules could also be actives. Bisecting k-means did not create any singletons. Although this seems a good result as it managed to cluster all the molecules into active clusters, 5,838 unknown activity molecules were clustered with the 272 active molecules. This is a large percentage amount when compared to only 272 active molecules, and therefore requires to be investigated. Out of 76 clusters, 37 active clusters contain only 1 active molecule together with large number of unknown activity ones. Therefore these 37 clusters would provide minimal to no added information to a random dataset. Having one active molecule that is inside one of the 37 clusters, would not provide further actives when searching in its cluster. Additionally most other clusters also contain a large proportion of unknown activity molecules, hence reducing their potential of having only actives in them. In fact, out of the 76 active clusters, only 7 clusters contain only active molecules.

With regards to the number of active clusters created, D-Butina obtained a positive result when compared to both DLSH-Butina and bisecting k-means, since it has the lowest number of clusters. This is ideal as having all the actives clustered together makes it easier to identify additional active compounds. This was also noted by [19] during the comparison of the presented approaches. In fact this is contrary to what was achieved by bisecting k-means where active molecules were spread out on 76 clusters with most clusters having minimal number of actives.

The distribution of unknown activity molecules within active clusters provides an insight of the unknown activity molecules to determine whether these have the potential to also be active. For D-Butina, the 315 unknown activity ZINC molecules are grouped in five active clusters. DLSH-Butina has 98 unknown activity molecules spread among seven active clusters. For Bisecting

k-means, unknown activity molecules are spread among 69 active clusters with most clusters containing a few number of actives. Although tests are required to determine the actual activity of the molecules, the distribution obtained by D-Butina seems to be of more interest than the other approaches. Considering that the unknown activity molecules are contained within a few clusters, this might show that there are structures within molecules that may exhibit activity. This can be useful for Structure Activity Relationship (SAR), to identify substructures within molecules for activity [35].

The cluster results obtained by D-Butina were selected to be investigated as they are the most promising among the three sets of results obtained. The similarity of each unknown activity molecule was calculated against the actives in the same cluster. [70] found that the similarity $\gamma > 0.3$ of two same activity molecules is hundreds of times greater compared to two random molecules, when the molecules are being represented by ECFP4 fingerprints. Therefore the similarity of unknown activity molecules was calculated against actives, and those achieving a similarity greater than 0.3 with any active molecule within the same cluster are listed in Appendix A as being potentially actives themselves. Out of 315 unknown activity molecules, nine molecules were identified as being potential active when considering their similarity with respect to active molecules. Some molecules had a similarity $\gamma > 0.3$ when compared to all the actives within the cluster, however the highest similarity pair was selected to be shown in the table.

This is a real world application where clustering can be used to identify new active compounds through the use of already known active molecules. The advantage offered is that molecules require a similarity comparison against other members within the cluster and not with every element in the dataset, hence this increases efficiency in the process.

4.6 Chapter Summary

This chapter started by performing experiments to find parameters for the two approaches presented, D-Butina and DLSH-Butina. Evaluation was then done based on the speed performance and quality of clusters produced when compared to the serial Butina approach and bisecting k-means. Finally, the results obtained were analysed and discussed in comparison to literature found.

Conclusion

This chapter presents a summary of the dissertation. It is then followed by a discussion on the aims and objectives achieved, while also discussing limitations of the research and what could be improved or extended. Finally future work is discussed providing areas within the research that merit further investigation.

Clustering is an important step in the drug discovery process such that structured selection of molecules for activity testing can be done [7]. This helps to reduce the time and costs required to identify active molecules with respect to a target protein. One challenge is the size of the datasets available. The datasets available contain millions of molecules and their size is constantly increasing, therefore widely used methods are not able to cluster these large datasets in reasonable time.

From the literature review in Chapter 2, it was found that Ward's clustering is the standard algorithm for small-molecule clustering [8]. However, considering its time and space complexity, it is unable to handle large datasets. Four clustering algorithms were selected and experiments were performed to determine which algorithm has the potential to be implemented in a distributed approach. The Butina clustering algorithm was chosen considering its good performance in separating actives from inactive molecules, better memory scalability than Ward's clustering and minimal parameter setting.

In Chapter 3, two distributed approaches were implemented based on Butina clustering algorithm, D-Butina and DLSH-Butina. D-Butina is a distributed clustering approach using Spark framework, with the clusters created being identi-

cal to the serial Butina algorithm. DLSH-Butina improves on the speed aspect of D-Butina by using approximation technique LSH in the process of neighbour identification.

In Chapter 4, the two approaches implemented were evaluated in comparison to the serial Butina clustering algorithm and bisecting k-means algorithm. Evaluation was done with the aim of comparing the speedup with respect to the serial approach and the scalability of the methods implemented in comparison to serial Butina and bisecting k-means. Finally evaluation was done to determine the ability of the algorithms to separate active molecules from putative inactive ones, followed by an analysis of grouping active molecules together.

5.1 Contributions

The aim of the dissertation was to create a distributed small-molecule clustering algorithm using a Big Data paradigm to cluster large small-molecule datasets. Since the clustering process of large datasets is a time consuming one, applying big data technologies to this process would aid to increase the performance, enabling to scale out the system. The approaches presented manage to achieve this aim of distributing the Butina clustering algorithm, achieving an increasing speedup as more nodes are added to the distributed cluster. Hence the primary aim was achieved.

The first objective was to identify a clustering algorithm that has been applied to small-molecule clustering and has the potential to handle larger datasets. An experiment was performed to compare algorithms found in literature highlighting the scalability both in terms of speed and memory and the quality of the clustering produced. Finally, Butina clustering algorithm was selected after taking into consideration its relative efficiency in terms of memory used, its minimal parameter setting and the good quality results achieved.

Through analysis of literature, the Apache Spark big data framework was selected as the most appropriate for the work required. The distributed implementation of the Butina clustering algorithm was then performed by creating a new approach to enable the distribution of the algorithm, D-Butina. No litera-

ture was found which attempted the distribution of this algorithm before. The implemented approach can scale as more worker nodes are added to the Spark cluster, achieving a speedup of 2.40 when using five workers and 3.89 when using ten workers on a dataset of 500,000 molecules, compared to the serial implementation. This was achieved while still obtaining the same clustering results of the serial approach. Although the scalability of the approach suffers from the distribution of data as the size of the dataset increases, our approach can still offer a benefit in clustering large datasets by adding more workers. This achieved the aim of distributing the Butina technique to enable large scale clustering.

Considering the $O(N^2)$ runtime scalability required by D-Butina to compute similarity matrix during the neighbour identification phase, LSH was used as an approximation technique. This was done to try and improve the performance of the algorithm in terms of speed. The change resulted in increasing the speedup to 4.11 for five workers and 8.37 when using ten workers compared to the serial Butina algorithm. On the other hand, the results obtained are of a lower quality with clusters being more fragmented into smaller ones and more active compounds being clustered into putative non-active clusters, making it more difficult to identify new actives.

Although both approaches suffer from the distribution of data among the worker nodes, they offer an approach to cluster the data in a distributed way using a technique that obtains good quality clusters, comparable to the standard algorithm Ward's clustering as seen in the comparison of serial approaches in Section 3.1. Hence the objectives of distributing the Butina clustering algorithm and applying an approximation method to improve the performance have been achieved.

5.2 Critique and Limitations

The dissertation is focussed on distributing the Butina clustering algorithm using a big data framework. This entailed distributing an algorithm that has never been distributed before. However limitations exist in the current approach. One such drawback is the high dependency on the shuffling of the data as seen in

Section 4.5.1. Compared to the bisecting k-means method the data shuffled by D-Butina and DLS-Butina is magnitudes higher, hence being more limited in scaling up. Unfortunately due to lack of resources, the algorithm could not be tested on a larger scale to identify the limits of its scalability. By testing the approaches on larger datasets and incrementing the number of workers, a better picture of the scalability of the implemented approaches could be identified. Additionally the limit at which scaling continues to grow at satisfactory rates, giving a performance benefit, would be determined.

With regards to clustering quality, there are no available activity-labeled large datasets of small molecules. Considering this limitation, given enough resources, experimental work could be done to determine the activity of unknown activity molecules that were clustered in active clusters. This would further validate the results produced. Some researchers evaluate clustering quality on small datasets, however this may hide some aspects on the quality of the results obtained. Some scenarios with regards to the data distribution might not be encountered. Therefore it would be ideal to have larger datasets of labeled data available, while also having a number of datasets that are standard to use for activity classification and clustering. This would enable better comparison among different research.

5.3 Future Work

Although the primary aim of this research has been achieved, the approaches presented can still be improved to increase their usability in the real world scenario. Butina algorithm and the presented approach D-Butina create good quality clusters, separating active molecule from inactive ones even when the dataset is large, with only five clusters having mixed activity. This algorithm merits further research considering the positive results obtained by having unknown activity molecules being present in only a small number of active clusters. Additionally 272 active molecules were contained within 36 clusters, making it easier to identify new active compounds.

One possible improvement is to improve D-Butina approach to reduce the

amount of data shuffling. Considering the large difference in the size of the shuffled data between D-Butina and bisecting k-means, there may be room for improvement to make the algorithm more efficient. This would aid the performance of the algorithm as the overhead of data distribution is reduced.

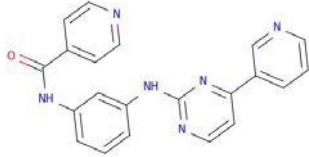
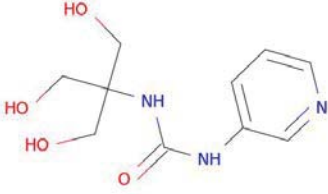
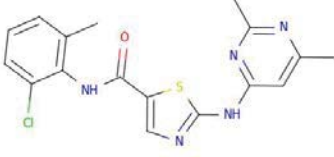

With regards to DLSH-Butina approach, it also achieved satisfactory results considering that approximation is involved. Although they are of less quality, the results obtained take approximately half the time D-Butina takes. This may be applied in cases where a reduced quality may be accepted in return for better speed performance. Notwithstanding its results, further optimisations with regards to the performance of the algorithm can be done. Methods implemented by [44] can be used to reduce the amount of data shuffled among the workers when using LSH. This approach entails partitioning the data in a way such that after hashing the data, all the elements hashed to the same key are already in the same partition, hence not requiring a shuffle to combine the data with the same key that is otherwise distributed among the workers. Additionally, when querying for a value, the query is not sent in broadcast to all the worker nodes, but a calculation is done to identify the partition that contains the required subset of data. The improvements presented in the research by [44] offer 87% reduction in the amount of data shuffled, while also 60% time reduction in the indexing time. This can benefit DLSH-Butina approach to further increase its performance and handle larger datasets.

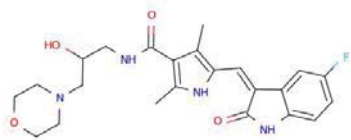
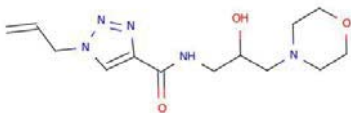
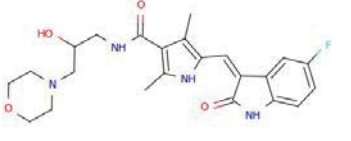
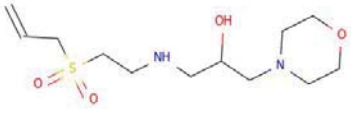
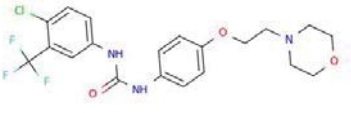
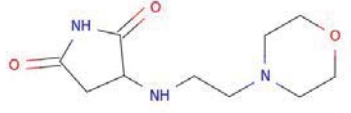
From evaluation aspect of the research, the large scale evaluation was performed on one type of protein target. Ideally this process is repeated for a number of targets from different categories discussed in Section 3.1.1. This would provide a better overview of the clustering results that Butina achieves. Additionally, considering that only research by [54] was identified of having performed a similar quality evaluation on such a large dataset, it would be beneficial to perform a similar experiment using a number of standard algorithms used in this area of small-molecule clustering. This would provide information on how the algorithms deal with large datasets.

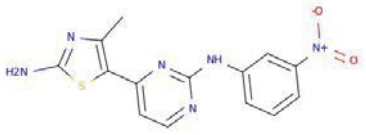
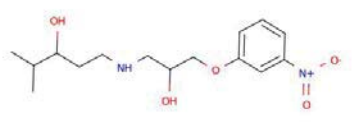
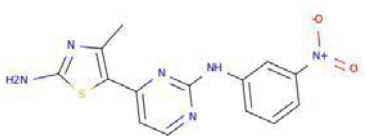
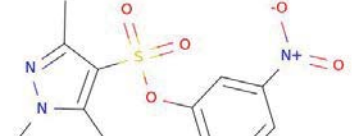
5.4 Final Remarks

The dissertation presents a big data approach to cluster large small-molecules datasets. A distributed approach based on Butina clustering algorithm was created while also being improved using approximation to increase the speed performance of the algorithm. These approaches performed well and we are now able to cluster millions of molecules by distributing the computation on a computer cluster. This research offers a contribution to the scientific community by extending an existing approach to handle larger datasets while evaluating its performance on a larger scale than ever performed before.

Unknown Activity molecules

Molecules		
active	unknown activity	similarity
 <chem>O=C(Nc1cccc(Nc2nccc(-c3cccn3)n2)c1)c1ccncc1</chem>	 <chem>O=C(Nc1ccnc1)NC(CO)(CO)CO</chem>	0.310345
 <chem>Cc1cc(Nc2ncc(C(=O)Nc3c(C)cc3)c1)s2)nc(C)n1</chem>	 <chem>Cc1cccc(Cl)c1NC(=O)Cn1ccc([N+](=O)[O-])c1N</chem>	0.366197

 <p>Cc1[nH]c(C=C2C(=O)Nc3ccc(F)cc32)c(C)c1C(=O)NCC(O)CN1CCOCC1</p>	 <p>0.365854</p> <p>C=CCn1cc(C(=O)NCC(O)CN2CCOC2)nn1</p>
 <p>Cc1[nH]c(C=C2C(=O)Nc3ccc(F)cc32)c(C)c1C(=O)NCC(O)CN1CCOCC1</p>	 <p>0.312500</p> <p>C=CCS(=O)(=O)CCNCC(O)CN1CCOCC1</p>
 <p>O=C(Nc1ccc(OCCN2CCOCC2)cc1)Nc1ccc(Cl)c(C(F)(F)F)c1</p>	 <p>0.307692</p> <p>O=C1CC(NCCN2CCOCC2)C(=O)N1</p>

 <p>Chemical structure of molecule 0.305556: A thiazole ring substituted with an amino group and a methyl group, linked via a pyridine ring to a para-nitrophenyl group.</p>	 <p>Chemical structure of molecule 0.305556: A complex molecule featuring a central amine group connected to a chain with two hydroxyl groups and a para-nitrophenyl group.</p>	0.305556
<chem>Cc1nc(N)sc1-c1ccnc(Nc2cccc([N+](=O)[O-])c2)n1</chem>	<chem>CC(C)C(O)CCNCC(O)COc1cccc([N+](=O)[O-])c1</chem>	0.323944
 <p>Chemical structure of molecule 0.323529: A thiazole ring substituted with an amino group and a methyl group, linked via a pyridine ring to a para-nitrophenyl group.</p>	 <p>Chemical structure of molecule 0.323529: A thiazole ring substituted with a methyl group and a sulfonamide group, linked to a para-nitrophenyl group.</p>	0.323529
<chem>Cc1nc(N)sc1-c1ccnc(Nc2cccc([N+](=O)[O-])c2)n1</chem>	<chem>Cc1nn(C)c(C)c1S(=O)(=O)Oc1cccc([N+](=O)[O-])c1</chem>	

		0.306667
<chem>Cc1nc(N)sc1-c1ccnc(Nc2cccc([N+](=O)[O-])c2)n1</chem>	<chem>CC(=Cc1cccc([N+](=O)[O-])c1)CNCCc1ncn(C)n1</chem>	

Table A.1: Showing unknown activity molecules compared to active molecules clustered together, having a similarity of $\gamma > 0.3$.

CD Contents

The attached CD contains the following contents:

- Soft copy of this report in pdf format
- Source code for D-Butina in `D_Butina`
- Source code for DLSH-Butina in `DLSH_Butina`
- Source code for quality analysis of the clusters output in folder `Output_Analysis`
- Data files used to perform evaluation in folder `Data`
- Docker script and necessary files to create docker image used in folder `Docker`

Installation Instructions

This appendix provides a guide to run D-Butina and DLSH-Butina on aztk Spark cluster.

C.1 Environment Setup

A number of steps are required such that the environment can be setup to create Spark clusters using aztk and enabling access to Azure Storage.

1. Install and setup aztk library
2. Create a Storage Account on Microsoft Azure. Upload data to cluster in repository file as a blob in the Storage account.
3. Update configuration in hidden file `secrets.yaml`, which can be found in the location where aztk was installed.
 - Setup azure batch account on Azure portal by running the script found at <https://github.com/Azure/aztk/blob/master/docs/00-getting-started.md> on the Azure Cloud Shell.
 - Complete in the installation
 - Copy the `service_principal` output in your `.aztk/secrets.yaml`.
4. Set the Azure Storage path containing the data in the file `Distributed_Approaches/dataPath.txt`

C.2 Running D-Butina

After having the environment set up, the necessary files can be copied, which allow D-Butina to run on Spark clusters.

1. Copy folder `Distributed_Approaches` to the same directory that `aztk` was installed
2. Open terminal in the containing folder and create cluster using the command `aztk spark cluster -id <cluster_name> -size <number of nodes> -vm-size <type of node> -docker-repo cassar1/rdkkitimage`
3. Check cluster status using the command `aztk spark cluster get -id <cluster_name>` until all the nodes have status `idle`
4. Submit task using `aztk spark cluster submit -id <cluster_name> -executor-memory <memory of each node in the form '2G'> -name <name of task> Distributed_Approaches/clustering.py <number of partitions> -py-files Distributed_Approaches/helpers/py`
5. Check the status by connecting to the master node through `ssh` using the command `aztk spark cluster ssh -id <cluster_name> -username <username>`

C.3 Running DLSH-Butina

After having the environment set up, the necessary files can be copied, which allow DLSH-Butina to run on Spark clusters.

1. Copy folder `Distributed_Approaches` to the same directory that `aztk` was installed
2. Open terminal in the containing folder and create cluster using the command `aztk spark cluster -id <cluster_name> -size <number of nodes> -vm-size <type of node> -docker-repo cassar1/rdkkitimage`
3. Check cluster status using the command `aztk spark cluster get -id <cluster_name>` until all the nodes have status `idle`

4. Submit task using `aztk spark cluster submit -id <cluster_name>`
`-executor-memory <memory of each node in the form '2G'>`
`-name <name of task> Distributed_Approaches/clustering.py`
`<number of partitions> -py-files Distributed_Approaches/helpers/py`
5. Check the status by connecting to the master node through ssh using the
command `aztk spark cluster ssh -id <cluster_name> -username <username>`

C.4 Running Output Analysis

Having the output file from either D-Butina or DLSH-Butina:

1. Copy output file to `Output_Analysis/results`
2. Run script using `python evluation_main.py`

Bibliography

- [1] K. C. Nicolaou, "Advancing the drug discovery and development process", *Angewandte Chemie - International Edition*, vol. 53, no. 35, pp. 9128–9140, 2014, ISSN: 15213773. DOI: 10.1002/anie.201404761.
- [2] B. E. Blass, *Case Studies in Drug Discovery*. 2015, pp. 499–529, ISBN: 9780124115088. DOI: 10.1016/B978-0-12-411508-8.00013-X.
- [3] J. D. Maccuish and N. E. Maccuish, "Chemoinformatics applications of cluster analysis", vol. 4, no. February, pp. 34–48, 2014. DOI: 10.1002/wcms.1152.
- [4] M. M. Kemp, M. Weïwer, and A. N. Koehler, "Unbiased binding assays for discovering small-molecule probes and drugs", *Bioorganic and Medicinal Chemistry*, vol. 20, no. 6, pp. 1979–1989, 2012, ISSN: 09680896. DOI: 10.1016/j.bmc.2011.11.071.
- [5] A. R. Leach and V. J. Gillet, *An Introduction to Chemoinformatics*. Springer International Publishing, 2007, ISBN: 978-1-4020-6290-2.
- [6] B. E. Blass, *Drug Discovery and Development*. 2015, pp. 1–34, ISBN: 9780124115088. DOI: 10.1016/B978-0-12-411508-8.00001-3.
- [7] M. J. Valler and D. Green, "Diversity screening versus focused screening in drug discovery", *Drug Discoveries and Therapeutics*, vol. 5, no. 7, p. 286, 2000.
- [8] F. Saeed, N. Salim, A. Abdo, and H. Hentabli, "Graph-Based Consensus Clustering for Combining Multiple Clusterings of Chemical Structures", *Molecular Informatics*, vol. 32, no. 2, pp. 165–178, 2013, ISSN: 18681743. DOI: 10.1002/minf.201200110.

- [9] M. A. Liebert and P. Willett, "Dissimilarity-Based Algorithms for Selecting Structurally Diverse Sets of Compounds", vol. 6, pp. 447–457, 1999.
- [10] A. Koutsoukas, S. Paricharak, W. R. J. D. Galloway, D. R. Spring, A. P. Ijzerman, R. C. Glen, D. Marcus, and A. Bender, "How Diverse Are Diversity Assessment Methods? A Comparative Analysis and Benchmarking of Molecular Descriptor Space", 2014.
- [11] G. M. Downs and J. M. Barnard, "Clustering Methods and Their Uses in Computational Chemistry", in *Reviews in Computational Chemistry*, vol. 18, 2003, ch. 1, pp. 1–40.
- [12] R. D. Brown and Y. C. Martin, "An Evaluation of Structural Descriptors and Clustering Methods for Use in Diversity Selection", *SAR and QSAR in Environmental Research*, no. September, 1998. DOI: 10.1080/10629369808033260.
- [13] T. Varin, R. Bureau, C. Mueller, and P. Willett, "Clustering files of chemical structures using the Székely-Rizzo generalization of Ward's method", *Journal of Molecular Graphics and Modelling*, vol. 28, no. 2, pp. 187–195, 2009, ISSN: 10933263. DOI: 10.1016/j.jm gm.2009.06.006.
- [14] P. J. Hansen and P. C. Jurs, "Chemical applications of graph theory. Part I. Fundamentals and topological indices", *Journal of Chemical Education*, vol. 65, no. 7, p. 574, 1988, ISSN: 0021-9584.
- [15] A. Dalby and J. Nourse, "Mdl Computer Chemical Structure File Formats", *Journal of Chemical Information and Computer Science*, vol. 32, pp. 244–255, 1992, ISSN: 1549-9596. DOI: 10.1021/ci00007a012.
- [16] D. Weininger, "SMILES, a Chemical Language and Information System: 1: Introduction to Methodology and Encoding Rules", *Journal of Chemical Information and Computer Sciences*, vol. 28, no. 1, pp. 31–36, 1988, ISSN: 00952338. DOI: 10.1021/ci00057a005.
- [17] D. Butina, "Unsupervised data base clustering based on daylight's fingerprint and tanimoto similarity: A fast and automated way to cluster small and large data sets", *Journal of Chemical Information and Computer Sciences*, vol. 39, no. 4, pp. 747–750, 1999, ISSN: 00952338. DOI: 10.1021/ci9803381.
- [18] J. W. Raymond, C. Blankley, and P. Willett, "Comparison of chemical clustering methods using graph- and fingerprint-based similarity measures", *Journal of Molecular Graphics and*

- Modelling*, vol. 21, no. 5, pp. 421–433, 2003, ISSN: 10933263. DOI: 10.1016/S1093-3263(02)00188-2.
- [19] M. Seeland, A. K. Johannes, and S. Kramer, “Structural clustering of millions of molecular graphs”, *Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14*, pp. 121–128, 2014. DOI: 10.1145/2554850.2555063.
- [20] C. Humblet, “Enhancing the diversity of a corporate database using chemical database clustering and analysis”, vol. 9, pp. 407–416, 1995.
- [21] C. H. Reynolds, R. Druker, and L. B. Pfahler, “Lead Discovery Using Stochastic Cluster Analysis (SCA): A New Method for Clustering Structurally Similar Compounds”, *Journal of Chemical Information and Computer Sciences*, vol. 38, no. 2, pp. 305–312, 1998, ISSN: 1549-9596. DOI: 10.1021/ci9700561.
- [22] D. Rogers and M. Hahn, “Extended-connectivity fingerprints”, *Journal of Chemical Information and Modeling*, vol. 50, no. 5, pp. 742–754, 2010, PMID: 20426451. DOI: 10.1021/ci100050t. eprint: <https://doi.org/10.1021/ci100050t>.
- [23] C. W. Chu, J. D. Holliday, and P. Willett, “Combining multiple classifications of chemical structures using consensus clustering”, *Bioorganic and Medicinal Chemistry*, vol. 20, no. 18, pp. 5366–5371, 2012, ISSN: 09680896. DOI: 10.1016/j.bmc.2012.03.010.
- [24] A. Kumar and K. Y. J. Zhang, “Hierarchical virtual screening approaches in small molecule drug discovery”, *Methods*, vol. 71, no. C, pp. 26–37, 2015, ISSN: 10959130. DOI: 10.1016/j.ymeth.2014.07.007.
- [25] M. A. Johnson and G. M. Maggiora, *Concepts and Applications of Molecular Similarity*, M. A. Johnson and G. M. Maggiora, Eds. New York, New York, USA: Wiley, 1990, p. 393, ISBN: 0471621757 9780471621751.
- [26] P. Thiel, L. Sach-Peltason, C. Ottmann, and O. Kohlbacher, “Blocked inverted indices for exact clustering of large chemical spaces”, *Journal of Chemical Information and Modeling*, vol. 54, no. 9, pp. 2395–2401, 2014, ISSN: 15205142. DOI: 10.1021/ci500150t.
- [27] W. Liu and D. E. Johnson, “Clustering and its application in multi-target prediction.”, *Current opinion in drug discovery & development*, vol. 12, no. 1, pp. 98–107, 2009, ISSN: 2040-3437.

- [28] J. Cordeiro, S. Hammoudi, L. Maciaszek, O. Camp, and J. Filipe, "Towards an Efficient and Distributed DBSCAN Algorithm Using MapReduce", *Lecture Notes in Business Information Processing*, vol. 227, pp. 75–90, 2015, ISSN: 18651348. DOI: 10.1007/978-3-319-22348-3.
- [29] M. G. Malhat, H. M. Mousa, and A. B. El-Sisi, "Improving Jarvis-Patrick algorithm for drug discovery", *2014 9th International Conference on Informatics and Systems, INFOS 2014*, DEKM61–DEKM66, 2015. DOI: 10.1109/INFOS.2014.7036710.
- [30] G. M. Downs, P. Willett, and W. Fisanick, "Similarity Searching and Clustering of Chemical-Structure Databases Using Molecular Property Data", *Journal of Chemical Information and Computer Sciences*, vol. 34, no. 5, pp. 1094–1102, 1994, ISSN: 00952338. DOI: 10.1021/ci00021a011.
- [31] R. D. Brown, Y. C. Martin, R. D. Brown, Y. C. Martin, P. P. Division, A. Laboratories, D. E. Ap, A. P. Road, and A. Park, "Use of Structure Activity Data To Compare Structure-Based Clustering Methods and Descriptors for Use in Compound Selection Use of Structure - Activity Data To Compare Structure-Based Clustering Methods and Descriptors for Use in Compound Selection", *Journal of Chemical Information and Computer Sciences*, vol. 36, no. May, pp. 572–584, 1996. DOI: 10.1021/ci9501047.
- [32] M. Stahl and H. Mauser, "Database clustering with a combination of fingerprint and maximum common substructure methods", *Journal of Chemical Information and Modeling*, vol. 45, no. 3, pp. 542–548, 2005, ISSN: 15499596. DOI: 10.1021/ci050011h.
- [33] D. J. Huggins, A. R. Venkitaraman, and D. R. Spring, "Rational Methods for the Selection of Diverse Screening Compounds", *ACS Chemical Biology*, pp. 208–217, 2011.
- [34] A. Böcker, G. Schneider, and A. Teckentrup, "NIPALSTREE: A new hierarchical clustering approach for large compound libraries and its application to virtual screening", *Journal of Chemical Information and Modeling*, vol. 46, no. 6, pp. 2220–2229, 2006, ISSN: 15499596. DOI: 10.1021/ci050541d.
- [35] A. Bocker, S. Derksen, E. Schmidt, A. Teckentrup, and G. Schneider, "A hierarchical clustering approach for large compound libraries", *Journal of Chemical Information and Modeling*, vol. 45, no. 4, pp. 807–815, 2005.
- [36] J. H. Ward, "Hierarchical Grouping to Optimize an Objective Function", *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963.

- [37] M. Steinbach, G. Karypis, and V. Kumar, "A Comparison of Document Clustering Techniques", *KDD workshop on text mining*, vol. 400, pp. 1–2, 2000, ISSN: 978-1-4244-2874-8. DOI: 10.1109/ICCCYB.2008.4721382.
- [38] M. G. Malhat, H. M. Mousa, and A. B. El-Sisi, "Clustering of chemical data sets for drug discovery", *2014 9th International Conference on Informatics and Systems*, DEKM-11-DEKM-18, 2014.
- [39] O. Kurasova, V. Marcinkevicius, V. Medvedev, A. Rapecka, and P. Stefanovic, "Strategies for Big Data Clustering", *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, pp. 740–747, 2014, ISSN: 1082-3409. DOI: 10.1109/ICTAI.2014.115.
- [40] M. Jain and C. Verma, "Adapting k-means for Clustering in Big Data", *International Journal of Computer Applications*, vol. 101, no. 1, pp. 19–24, 2014.
- [41] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A. Y. Zomaya, S. Foufou, and A. Bouras, "A survey of clustering algorithms for big data: Taxonomy and empirical analysis", *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 3, pp. 267–279, 2014, ISSN: 21686750. DOI: 10.1109/TETC.2014.2330519.
- [42] C. Radhika and D. Parameswari, "Distributed Clustering for Big Data with MapReduce", *IOSR Journal of Computer Engineering*, vol. 19, no. 03, pp. 25–28, 2017, ISSN: 22788727. DOI: 10.9790/0661-1903032528.
- [43] W. Zhang, D. Li, Y. Xu, and Y. Zhang, "Shuffle-efficient distributed Locality Sensitive Hashing on spark", *Proceedings - IEEE INFOCOM*, vol. 2016-Septe, pp. 766–767, 2016, ISSN: 0743166X. DOI: 10.1109/INFCOMW.2016.7562179.
- [44] D. Li, W. Zhang, S. Shen, and Y. Zhang, "SES-LSH: Shuffle-Efficient Locality Sensitive Hashing for Distributed Similarity Search", *Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017*, pp. 822–827, 2017. DOI: 10.1109/ICWS.2017.99.
- [45] M. Nowotka and A. Hersey, *Locality sensitive hashing for compound similarity search*. Poster, Seventh Joint Sheffield Conference on Chemoinformatics, University of Sheffield, Sheffield, University of Sheffield, Jul. 2016.
- [46] A. Gionis, P. Indyk, and R. Motwani, "Similarity Search in High Dimensions via Hashing", in *Proceedings of the 25th International Conference on Very Large Data Bases*, ser. VLDB '99, 1999, pp. 518–529, ISBN: 1-55860-615-7.

- [47] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters", in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, 2004, pp. 137–150.
- [48] N. Hans, S. Mahajan, and S. N. Omkar, "Big Data Clustering Using Genetic Algorithm On Hadoop Mapreduce", vol. 4, no. 04, pp. 58–62, 2015.
- [49] S. Landset, T. M. Khoshgoftaar, A. N. Richter, and T. Hasanin, "A survey of open source tools for machine learning with big data in the Hadoop ecosystem", *Journal of Big Data*, vol. 2, no. 1, p. 24, 2015, ISSN: 2196-1115. DOI: 10.1186/s40537-015-0032-1.
- [50] M. Zaharia, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, and S. Venkataraman, "Apache Spark: a unified engine for big data processing", *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016, ISSN: 00010782. DOI: 10.1145/2934664.
- [51] C. Jin, R. Liu, Z. Chen, W. Hendrix, A. Agrawal, and A. Choudhary, "A Scalable Hierarchical Clustering Algorithm Using Spark", *Big Data Computing Service and Applications (BigDataService), 2015 IEEE First International Conference on*, pp. 418–426, 2015. DOI: 10.1109/BigDataService.2015.67.
- [52] M. Stahl, H. Mauser, M. Tsui, and N. R. Taylor, "A robust clustering method for chemical structures", *J Med Chem*, vol. 48, no. 13, pp. 4358–4366, 2005. DOI: 10.1021/jm040213p.
- [53] F. Saeed, A. Ahmed, M. S. Shamsir, and N. Salim, "Weighted voting-based consensus clustering for chemical structure databases", *Journal of Computer-Aided Molecular Design*, vol. 28, no. 6, pp. 675–684, 2014, ISSN: 15734951. DOI: 10.1007/s10822-014-9750-2.
- [54] S. V. Trepalin and A. V. Yarkov, "Hierarchical clustering of large databases and classification of antibiotics at high noise levels", *Algorithms*, vol. 1, no. 2, pp. 183–200, 2008, ISSN: 19994893. DOI: 10.3390/a1020183.
- [55] M. Seeland, S. A. Berger, A. Stamatakis, and S. Kramer, "Parallel structural graph clustering", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6913 LNAI, no. PART 3, pp. 256–272, 2011, ISSN: 03029743. DOI: 10.1007/978-3-642-23808-6_17.

- [56] R. E. Higgs, K. G. Bemis, I. A. Watson, and J. H. Wikel, "Experimental Designs for Selecting Molecules from Large Chemical Databases", *Journal of Chemical Information and Computer Sciences*, vol. 2338, no. 97, pp. 861–870, 1997. DOI: 10.1021/ci9702858.
- [57] W. Li, "A fast clustering algorithm for analyzing highly similar compounds of very large libraries", *Journal of Chemical Information and Modeling*, vol. 46, no. 5, pp. 1919–1923, 2006, ISSN: 15499596. DOI: 10.1021/ci0600859.
- [58] Z. Feng, B. Zhou, and J. Shen, "A parallel hierarchical clustering algorithm for PCs cluster system", *Neurocomputing*, vol. 70, no. 4-6, pp. 809–818, 2007, ISSN: 09252312. DOI: 10.1016/j.neucom.2006.10.034.
- [59] T. Sun, C. Shu, F. Li, H. Yu, L. Ma, and Y. Fang, "An Efficient Hierarchical Clustering Method for Large Datasets with Map-Reduce", *2009 International Conference on Parallel and Distributed Computing, Applications and Technologies*, pp. 494–499, 2009, ISSN: 2379-5352. DOI: 10.1109/PDCAT.2009.46.
- [60] M. Dehmer, F. Emmert-streib, and S. Tripathi, "Large-Scale Evaluation of Molecular Descriptors by Means of Clustering", *PloS one*, vol. 8, no. 12, 2013. DOI: 10.1371/journal.pone.0083956.
- [61] M. G. Malhat and A. B. El-Sisi, "Parallel ward clustering for chemical compounds using MapReduce", *Proceedings - 2015 10th International Conference on Computer Engineering and Systems, ICCES 2015*, pp. 23–27, 2016, ISSN: 18650929. DOI: 10.1109/ICCES.2015.7393011.
- [62] A. Bo, "Toward an Improved Clustering of Large Data Sets Using Maximum Common Substructures and Topological Fingerprints Toward an Improved Clustering of Large Data Sets Using Maximum Common", *Journal of chemical information and modeling*, pp. 2097–2107, 2008.
- [63] J. Chen, S. J. Swamidass, Y. Dou, J. Bruand, and P. Baldi, "ChemDB: A public database of small molecules and related chemoinformatics resources", *Bioinformatics*, vol. 21, no. 22, pp. 4133–4139, 2005, ISSN: 13674803. DOI: 10.1093/bioinformatics/bti683.
- [64] "PubChem substance and compound databases", *Nucleic Acids Research*, vol. 44, no. D1, pp. D1202–D1213, 2016, ISSN: 13624962. DOI: 10.1093/nar/gkv951.

- [65] T. Sterling and J. J. Irwin, "ZINC 15 Ligand Discovery for Everyone", *Journal of Chemical Information and Modeling*, vol. 55, no. 11, pp. 2324–2337, 2015. DOI: 10.1021/acs.jcim.5b00559.
- [66] Y. Cao, T. Jiang, and T. Girke, "Accelerated similarity searching and clustering of large compound sets by geometric embedding and locality sensitive hashing", *Bioinformatics*, vol. 26, no. 7, pp. 953–959, 2010, ISSN: 13674803. DOI: 10.1093/bioinformatics/btq067.
- [67] G. Hu, G. Kuang, W. Xiao, W. Li, G. Liu, and Y. Tang, "Performance Evaluation of 2D Fingerprint and 3D Shape Similarity Methods in Virtual Screening", *Journal of Chemical Information and Modeling*, vol. 52, no. 5, pp. 1103–1113, 2012.
- [68] M. M. Mysinger, M. Carchia, J. J. Irwin, and B. K. Shoichet, "Directory of useful decoys, enhanced (DUD-E): Better ligands and decoys for better benchmarking", *Journal of Medicinal Chemistry*, vol. 55, no. 14, pp. 6582–6594, 2012, ISSN: 00222623. DOI: 10.1021/jm300687e.
- [69] R. B. Zadeh, X. Meng, A. Staple, B. Yavuz, L. Pu, S. Venkataraman, E. Sparks, A. Ulanov, and M. Zaharia, "Matrix Computations and Optimization in Apache Spark", 2015. DOI: 10.1145/2939672.2939675. arXiv: 1509.02256.
- [70] S. Jasial, Y. Hu, M. Vogt, and J. Bajorath, "Activity-relevant similarity values for fingerprints and implications for similarity searching", *F1000Research*, vol. 5, p. 591, Apr. 2016.
- [71] J. L. Gustafson, "Amdahl's law", in *Encyclopedia of Parallel Computing*, D. Padua, Ed. Boston, MA: Springer US, 2011, pp. 53–60, ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_77.