Virtual Screening Algorithms
Thomas G. Kristensen

PhD Dissertation

Department of Computer Science
Bioinformatics Research Centre
Aarhus University
Denmark
Virtual Screening Algorithms

A Dissertation
Presented to the Faculty of Science
of Aarhus University
in Partial Fulfilment of the Requirements for the
PhD Degree

by
Thomas G. Kristensen
January, 2011
Abstract

When developing new drugs, one key aspect is identifying novel chemical compounds for inhibiting or promoting chemical reactions in living organisms. The space of synthesizable molecules is rapidly expanding, and computational methods are needed for screening through large databases of these to identify new potential drugs. These methods need to be both effective and efficient to aid in the drug development process.

A multitude of methods based on different representations exist for performing virtual screenings. My research contributions span many different representations: binary vectors, numeric vectors, strings, graphs and 3D structures. When dealing with simple representations, such as vectors, my focus has been on increasing the efficiency of existing methods. When working with more complex models, such as graphs, my focus has been on increasing the effectiveness. During my PhD I have worked with the projects presented in the following paragraphs.

Molecules can be represented as binary vectors which can be compared using the Tanimoto similarity measure. This measure is cheap to calculate, and it is therefore often used for very large databases. I have developed novel storage and retrieval methods for looking up similar binary vectors efficiently. The new data structures have been compared to existing methods and have been observed to perform superior to all other methods they have been tested against.

I have also worked on general vectors where entries are allowed to take on arbitrary values. I have devised and tested a limit which can be used for accelerating database look-ups, using existing metric data structures.

String representations are also common in computational chemistry, as they can be used to canonically describe a molecule. The LINGO similarity measure is one measure which is used to rapidly compare molecules based on some string representation. I have developed and tested an algorithm for rapidly calculating the LINGO similarity matrix for a set of molecules, which outperforms the existing methods.

Another very common representation is based on graphs and the comparison of these, typically using some variation of the size of the largest common sub graph. In my work I have developed a tree representation of small molecules, which has been used to compare molecules and measure their similarity. Both the effectiveness and the efficiency of this method has been compared to that of existing methods. Together with PREP student Anders Johnsen I have developed a novel method for comparing graphs that is based on a more refined similarity measure than the size of the largest common sub graph. The project
was done as a Practical Research Project under my supervision.

Lastly, I have worked on aligning molecules in 3D space. This part of my work spawned different projects, including investigations into ways of handling angles and orientations in evolutionary computation. It also covered comparisons between different numerical optimisers on problems from computational chemistry, and I have developed methods for effectively aligning molecules to recreate their binding mode. The methods were tested against existing methods, both on their ability to recreate the native binding mode and on their effectiveness as a screening protocol.
Acknowledgements

I own a debt of gratitude to all the people who have helped me complete my PhD.

First of all I would like to thank my supervisor Christian N. S. Pedersen for all our valuable discussions and his feedback to my work. Also, I would like to thank him for leading the way through the process of performing research, publishing papers and writing a dissertation.

René Thomsen and Michael H. Christensen of Molegro ApS, our local virtual screening experts, have been an invaluable source of information and suggestions to my work. Without them I would not know where to even begin investigating the virtual screening field.

Pierre F. Baldi was extremely helpful during my stay abroad at the Institute of Genomics and Bioinformatics at the University of California, Irvine, and even helped me find a place to live, which I am immensely grateful for. Katarina Fletcher also guided me through all the hoops that are involved in gaining entry to the US and UCI – without her, my first couple of weeks would have been much harder. I would like to thank the IGB group (along with their CS friends) for a nice environment, both academically and socially. Especially Ramzi J. Nasr and Michael D. Zeller helped me integrate into the wonderful environment that is the UCI campus.

Lastly, I would like to thank all the wonderful people here at the Bioinformatics Research Centre at Aarhus University for a good research environment. It is nice to see that experts from very diverse fields can get along at the same centre, and even have fruitful discussions. I think a PhD student needs an international environment to learn from other research institutes and cultures, and BiRC has definitely provided this in both its faculty and students. I hope my fellow PhD students will have as good an experience here as I have had.

Thomas G. Kristensen,
Aarhus, January 2011.
Contents

Abstract v
Acknowledgements vii

I Overview 1

1 Introduction 3
  1.1 Virtual screening . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.2 Retrospective screening studies . . . . . . . . . . . . . . . . . 4
  1.3 Thesis outline . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2 Vector similarity 11
  2.1 Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
    2.1.1 Binary vectors . . . . . . . . . . . . . . . . . . . . . . . . 12
    2.1.2 Counting vectors . . . . . . . . . . . . . . . . . . . . . . . 13
    2.1.3 Real valued vectors . . . . . . . . . . . . . . . . . . . . . . 13
    2.1.4 Distance and similarity measures . . . . . . . . . . . . . . 13
    2.1.5 Queries . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
  2.2 Related work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
  2.3 Research contributions . . . . . . . . . . . . . . . . . . . . . . . 17
    2.3.1 kD grid . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
    2.3.2 Multibit tree . . . . . . . . . . . . . . . . . . . . . . . . . . 18
    2.3.3 Other binary vector datastructures . . . . . . . . . . . . . . 19
    2.3.4 Converting Tanimoto query to distance query . . . . . . . 20
  2.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
  2.5 Future work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

3 String similarity 23
  3.1 Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
  3.2 Related work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
  3.3 Research contributions . . . . . . . . . . . . . . . . . . . . . . . . 27
    3.3.1 SMILES string edit distance . . . . . . . . . . . . . . . . . 27
    3.3.2 Rapid LINGOsim calculation . . . . . . . . . . . . . . . . . 28
  3.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
  3.5 Future work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Graph comparison</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>Background</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>Related work</td>
<td>32</td>
</tr>
<tr>
<td>4.3</td>
<td>Research contributions</td>
<td>34</td>
</tr>
<tr>
<td>4.3.1</td>
<td>LarCCS</td>
<td>34</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Clique distribution</td>
<td>35</td>
</tr>
<tr>
<td>4.4</td>
<td>Summary</td>
<td>36</td>
</tr>
<tr>
<td>4.5</td>
<td>Future work</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>Ligand alignment</td>
<td>39</td>
</tr>
<tr>
<td>5.1</td>
<td>Background</td>
<td>40</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Overlap measures</td>
<td>40</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Types of alignments</td>
<td>42</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Algorithms</td>
<td>43</td>
</tr>
<tr>
<td>5.2</td>
<td>Related work</td>
<td>45</td>
</tr>
<tr>
<td>5.3</td>
<td>Research contributions</td>
<td>47</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Optimal overlay using Differential Evolution</td>
<td>47</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Optimal angles using Differential Evolution</td>
<td>48</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Aligning pharmacophores using DE and CMA-ES</td>
<td>50</td>
</tr>
<tr>
<td>5.4</td>
<td>Summary</td>
<td>51</td>
</tr>
<tr>
<td>5.5</td>
<td>Future work</td>
<td>51</td>
</tr>
<tr>
<td>6</td>
<td>Conclusions and summary</td>
<td>53</td>
</tr>
<tr>
<td>II</td>
<td>Papers</td>
<td>55</td>
</tr>
<tr>
<td>7</td>
<td>A tree-based method for screening chemical fingerprints</td>
<td>57</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>59</td>
</tr>
<tr>
<td>7.2</td>
<td>Methods</td>
<td>60</td>
</tr>
<tr>
<td>7.2.1</td>
<td>kD grid</td>
<td>61</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Singlebit tree</td>
<td>63</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Multibit tree</td>
<td>65</td>
</tr>
<tr>
<td>7.3</td>
<td>Experiments</td>
<td>66</td>
</tr>
<tr>
<td>7.4</td>
<td>Results</td>
<td>68</td>
</tr>
<tr>
<td>7.5</td>
<td>Conclusion</td>
<td>71</td>
</tr>
<tr>
<td>8</td>
<td>Transforming Tanimoto queries to range queries</td>
<td>75</td>
</tr>
<tr>
<td>9</td>
<td>Data structures for accelerating queries on real valued vectors</td>
<td>81</td>
</tr>
<tr>
<td>9.1</td>
<td>Introduction</td>
<td>83</td>
</tr>
<tr>
<td>9.2</td>
<td>Data structures</td>
<td>84</td>
</tr>
<tr>
<td>9.3</td>
<td>Experimental setup</td>
<td>86</td>
</tr>
<tr>
<td>9.4</td>
<td>Results</td>
<td>87</td>
</tr>
<tr>
<td>9.5</td>
<td>Conclusion</td>
<td>90</td>
</tr>
</tbody>
</table>
Part I

Overview
Chapter 1

Introduction

The drug discovery process is long and expensive. It is estimated that the development cost of a new drug is between $500 million and $2 billion [1], and the time from a potential drug candidate is discovered to it reaches the pharmacy shelves can be more than 10 years [99]. Strict regulations, ensuring our safety as consumers, demand rigorous tests to identify side effects and long term medical consequence. When a drug candidate is discovered, or an existing drug is sought improved, it is necessary to identify other drug candidates to find compounds with a better effect. This is done in the first process of the drug development process and can be accelerated and made less expensive by employing computational methods. These methods can also aid researchers in investigating the growing chemical structure databases. An example is the ZINC database which contains more than eight million compounds [48]. Rigourously testing all of these is unfeasible when using traditional wet-lab methods and calls for computational aid. The process of identifying drug candidates using computational methods is called virtual screening, or sometimes high throughput virtual screening (HTVS).

1.1 Virtual screening

A small molecule binding to a larger molecule is called a ligand. In virtual screening we are targeting proteins in the human body to find novel ligands that will bind to them. Virtual screening can be divided into two classes: target based and ligand based. In target based virtual screening, a 3D structure of the target protein is known, and the goal is to identify ligands, from a database of candidates, that will bind to this 3D structure. One way of doing this is by using molecular docking, a computational process where ligands are moved in 3D space in an attempt to find a configuration of target and ligand which maximises some scoring function. The ligands in the database are ranked according to their maximum score, and the best of these can be investigated further, e.g. by examining how they bind to the protein and what interactions take place. Many different programs for performing these dockings exist, but the target based methods will not be the focus of this study.

Finding the 3D structure of a protein is expensive and sometimes not feasible
as proteins, e.g. in the membrane, are difficult to crystallise. An alternative to target based virtual screening when the structure of the target is unknown is ligand based screening, in which known active binding ligands are used for screening the database. These methods have the advantage that they are faster; several ligands can be taken into account; and some studies suggest that they outperform target based methods, at least on some proteins [43]. In ligand based virtual screening, one or more known active binders are used to screen a database of ligands, by calculating a similarity score between the known active(s) and each database entry. The database can be re-ranked according to this similarity, and the top of the returned ligands can be investigated further, e.g. by using a computational more expensive method such as docking.

There is a wide variety of ligand representations which can be used when quantifying the similarity between ligands. For example, if the ligands are represented as graphs, an obvious similarity measure could be the size of the largest common sub graph. These similarity measures are often normalised to prevent large molecules with many features from dominating the results. In my work I have used two such normalisation measures; the first of these I will refer to as Tanimoto normalisation. If $S(A,B)$ denotes some similarity between ligand $A$ and $B$, the Tanimoto normalised similarity is defined as

$$\frac{S(A,B)}{S(A,A) + S(B,B) - S(A,B)}.$$ 

This normalisation has the property that, if $0 \leq S(A,B) \leq S(A,A)$ for any $A$ and $B$, the normalisation will yield a measure between zero and one, where one is optimal. The measure will penalise molecules where $S(A,A)$ and $S(B,B)$ are very different. The other normalisation method, which I will refer to as product normalisation, has the form

$$\frac{S(A,B)^2}{S(A,A)S(B,B)}$$

and lies between zero and one, as the Tanimoto normalised similarity. No studies comparing the two methods in a chemoinformatics framework seem to exist.

Different ligand representations and similarity measure give rise to different rankings and have varying performance on different targets. The ranking produced by different method can be combined using data fusion techniques, such as using the sum of ranks for each ligand. This is a well studied subject [36,45,126] and will not be the focus of this thesis.

1.2 Retrospective screening studies

As previously mentioned, the drug discovery process is very long, and it is therefore difficult to evaluate new screening methods. A common strategy for performing an assessment of the quality of a screening method is to perform a retrospective screening study. In such a study, several known active binders must be known beforehand. Some of the known binders are shuffled with a
1.2. Retrospective screening studies

set of decoy ligands to create a hypothetical compound database, while the remaining binders are used for screening the database. Typically, only one ligand is retained for re-ranking the database. Figure 1.1 illustrates the process of re-ranking a set of known binders and decoys according to one selected binder.

The Directory of Useful Decoys (DUD) data set [47] is designed for performing retrospective screening studies on target based methods, but it has also been used on ligand based methods. The data set is divided into 40 subsets, one for each of 40 targets. For each target, a varying number of active binders is known, and for each active, a set of 33 decoys (on average) with similar weight and other properties, is added to the database. A drawback of this data set is, that the active binders for each target are structurally very similar, which makes it a bit easier to discriminate using simple ligand based similarity measures.

The Maximum Unbiased Validation (MUV) data set [97] seeks to correct this by selecting decoys that are much more difficult to distinguish using ligand

<table>
<thead>
<tr>
<th>selected binder</th>
<th>enrichment</th>
<th>enrichment factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>50%</td>
<td>1.25</td>
<td></td>
</tr>
<tr>
<td>50%</td>
<td>0.83</td>
<td></td>
</tr>
<tr>
<td>100%</td>
<td>1.25</td>
<td></td>
</tr>
<tr>
<td>100%</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1.1: Illustration of a retrospective screening study.
Chapter 1. Introduction

The classification result \(p\) is better than \(q\) at the specificity level as it has a better sensitivity. The ROC curve for a random classifier is presented as a dotted line. An optimal classifier has a false positive rate of zero and a true positive rate of one, as indicated by the point in the upper left corner.

Figure 1.2: Illustration of ROC points (\(p\) and \(q\)) and a ROC curve (solid curve). The classification result \(p\) is better than \(q\) at the specificity level as it has a better sensitivity. The ROC curve for a random classifier is presented as a dotted line. An optimal classifier has a false positive rate of zero and a true positive rate of one, as indicated by the point in the upper left corner.

Information. In the MUV, the ratio between known binders and decoys is 500 and there are 17 different targets, each with 30 known binders. The binders are selected such that they cover as many scaffolds (classes of binding modes) as possible. Studies confirm that this data set is much harder to identify binders in. Both of these data sets are well known in the field and will be referred to in the coming chapters.

The ranking produced in a retrospective screening study can be evaluated in several ways. One of the most intuitive is to use the enrichment at different percent of the ordered lists of ligands. For example, the enrichment at one percent would be the percentage of known active binders in the top one percent of the ordered list of binders and decoys. The enrichment at different points of a ranked list can be seen in Figure 1.1. A derived measure is the enrichment factor which measures how much better than randomly ordered lists a method is, by measuring the fraction of found known actives in the top \(x\)% percent of the ranked list, compared to the ratio between binders and decoys in the entire database. For example, if the enrichment at some point is 30% and 10% of the database is actives, this is three times better than expected. The enrichment factors are also presented in Figure 1.1.
1.2. Retrospective screening studies

The enrichment and the enrichment factor has several shortcomings. Firstly, they only describe a property of a fraction of the ordered list, and secondly, they require the (arbitrary) specification of some threshold. Receiver operating characteristic (ROC) curves solve these two problems. In binary classification the ROC curve is defined as a plot between the false positive rate (FPR) and the true positive rate (TPR), corresponding to the specificity and sensitivity. The FPR is calculated as the number of negatives (decoys) incorrectly categorised as being positive (hits) divided by the total number of negatives. Similarly, the TPR is the number of correctly categorised positives over the total number of positives. If a binary classification strategy has a free parameter, such as a classification threshold, adjustments to the parameter will give rise to different FPR and TPR points, yielding a ROC curve illustrated in Figure 1.2. In virtual screening, the free parameter is the similarity threshold for which ligands are classified as hits or decoys.

The area under curve (AUC) value gives a summary of the curve by calculating the area under it. A random classifier is expected to yield ROC points where FPR = TPR which forms a straight line from (0, 0) to (1, 1), and therefore an AUC value of 0.5. Any method which yields an AUC value above 0.5 performs better than a random ranking; and methods worse than random can be turned into one better than random simply by reversing the ranking. An optimal ranking will yield an AUC value of 1.0. The AUC value has the disadvantage that the two ROC curves in Figure 1.3 will yield the same AUC value, even though the full line is clearly superior to the dashed. Several correction measures have been proposed, e.g. BEDROC [117] and CROC [109], but some studies show that they do not add any new information. For example, the BEDROC has been shown to be very highly correlated to the enrichment factor [101].

Figure 1.3: An illustration of two ROC curves which yield the same AUC value. However, the full line can be said to be superior to the dashed as it has much better early recall.
Chapter 1. Introduction

1.3 Thesis outline

The remainder of this dissertation is divided into two parts. The first part is an overview of the molecule representations I have worked with for performing ligand based virtual screening, including their background, related work from the field and my research contributions.

Chapter 2 covers vector based methods with a short presentation of how they are created and used. The chapter will mainly focus on accelerating queries into databases of vectors and will cover both real valued, as well as binary vectors.

Chapter 3 covers string methods, all based on canonical SMILES strings. The LINGO method is described, and existing methods for accelerating the calculation of this is presented, as well as a method developed as part of my studies. Furthermore, a new method using the Levenshtein distance from bioinformatics is introduced.

Chapter 4 covers graph representations, both for trees and general graphs. Several existing representations and similarity measures are discussed, and two novel methods are presented.

Chapter 5 covers methods based on aligning 3D structures, both for extracting pharmacophores but also for directly comparing molecular structures. Several existing methods are presented and categorised using a framework introduced in the chapter. Two new methods are presented and compared to the existing literature using the framework. The chapter will also cover work done in the field of evolutionary computation.

Chapter 6 contains a small summary of the research contributions.

The second part of the thesis contains the four conference papers, which have all been presented at international conferences, and the two journal articles, which have been published in international peer-reviewed journals. The papers and articles are


1.3. Thesis outline


Apart from these articles I have recently had the article *Using inverted indices for accelerating LINGO calculations* accepted in *Journal of Chemical Information and Modelling* on the condition that the reviewers revisions are followed. A revised manuscript has been submitted to the journal.

I have also presented my work as posters on conferences [68, 69, 72]. These are

- T. G. Kristensen and C. N. S. Pedersen. All-to-all RF Distance Methods. At *Bioinformatics 2008, Warsaw, Poland, 2008*.


- T. G. Kristensen, C. N. S. Pedersen, R. Thomsen and M. Christensen. Largest Common Chemical Feature Subtree as a Virtual Screening Method. At *17th Annual International Conference on Intelligent Systems for Molecular Biology (ISMB) and 8th European Conference on Computational Biology (ECCB), Stockholm, Sweden, 2009*. 
Chapter 2

Vector similarity

If molecules are represented as points in space it is possible to easily compare them using standard distance and similarity measures. This chapter will present different vector representations of molecules, comparison methods for these vectors and algorithmic methods to handle them.

The simplest representation is vectors with binary entries, each entry corresponding to the presence or absence of some property, e.g. aromatic rings. Such vectors are called *binary vectors* or fingerprints, and lie in some space \( \{0, 1\}^n \). If the entries are allowed to take on positive integers, instead of only zeroes and ones, the vectors can be used for counting the number of occurrences of properties in the molecule, and the vector is called a *counting vector*. These lie in \( \mathbb{N}_+^n \) and can for example be used to count e.g. the number of aromatic rings in a molecule. In the most general case the vectors are allowed to lie anywhere in space (\( \mathbb{R}^n \)), in which case they are called *real valued vectors* or descriptors. These can be used for describing more fine grained properties of molecules, such as their molecular weight or log \( p \) value. Vectors of a given kind can be compared using similarity or distance metrics. Distance in Euclidian space is probably the best known metric, but the Tanimoto similarity seems to yield better results in retrospective screening studies [128]. All entries of vectors in this chapter will belong to one of the above mentioned vector groups. The vectors of the different groups are not mixed in the field, and combined vectors e.g. in \( \{0, 1\} \times \mathbb{R} \times \mathbb{R} \times \mathbb{N}_+ \), have not been studied.

Vectors are widely used due to the fact that similarity is often very fast to calculate, and it is therefore very fast to screen a very large database. They are also very space efficient, easy to generate and they are (often) invariant under rotations of bonds. One drawback is the fact that algorithms handling them can be hard to analyse, as there is no framework for modelling vectors generated from molecular data. However, there has recently been some novel work done in the field of modelling binary vectors using different distributions [8]. It remains to be seen, if this will make it possible to better understand how different methods perform on real data.

The material in this chapter covers my conference papers *A tree based method for the rapid screening of chemical fingerprints* [64] (Chapter 7) and *Data structures for accelerating Tanimoto queries on real valued vectors* [71] (Chapter 8); and my journal articles *A tree based method for the rapid screening*
of chemical fingerprints [65] (Chapter 7) and Transforming Tanimoto queries on real valued vectors to range queries in Euclidian space [63] (Chapter 8).

This chapter will first define binary, counting and real valued vectors. It will cover how these can be used for performing a virtual screening, using vector similarity measures. Queries will be defined, and existing studies accelerating these will be presented. Finally, my research contributions covering accelerating binary and real valued vector queries will be presented.

2.1 Background

There are three major types of vectors, namely binary, count and real valued, all covered in this section. There are several applications of these vectors, but as the focus in this dissertation will be on virtual screening, this section will cover distance and similarity measures, along with different types of queries that can be performed on these.

2.1.1 Binary vectors

Structural keys and molecular fingerprints are the two major types of binary vectors. Structural keys are generated by fixing a dictionary of \( n \) fragments, and using this for generating the binary vectors called structural keys. An obvious drawback of this method is, that the dictionary has to be generated by hand before it is possible to extract the structural keys, and in generating this dictionary, important fragments might be excluded. Therefore, structural keys have largely been abandoned, and molecular fingerprints are much more often used for generating binary vector representation.

Molecular fingerprints are generated by traversing the molecular structures in either breadth first or depth first searches, starting in all atoms. The searches are terminated at a maximum recursion depth, and the canonical order in which the atoms were visited corresponds to a key, which can be used to generate a key-unique index that can then be set. Using this strategy, different sets of molecules will give rise to fingerprints of different dimensionality and compression is therefore often used to truncate the fingerprints to a restricted length. One way of doing this is to use a binary operator to fold a fingerprint to a given length \( n \), generating a shorter summary fingerprint but losing information in the process. If the exclusive or (XOR) operator is used the \( i \)th entry in the compressed fingerprint is set, if and only if there is an uneven number of key-unique indexes \( j_k \) for which \( j_k \mod n = i \). Similarly, if the OR operator is used the \( i \)th entry in the compressed fingerprint is set if there is a least one key-unique index \( j \) for which \( j \mod n = i \). A study by Swamidass et al. [111] examined the effect of the OR strategy and found that the effect on some widely used similarity coefficient can be modelled as a systematic error for which it is possible to correct. Another study by Baldi et al. [5] describes a loss-less compression scheme where Golomb and Elias entropy codes are used to compress fingerprints, greatly reducing their size, but having the drawback that fingerprints have to be uncompressed to measure similarity.
2.1.2 Counting vectors

Counting vectors, also called non-binary vectors, can be generated in a manner similar to structural keys and molecular fingerprints, simply by counting occurrences of fragments or atom paths in the molecules. If binary vectors are interpreted as being sets of features, counting vectors are multisets in which an element (e.g. a molecule path) appears several times. There does not seem to be any research in compressing counting vectors, which is understandable as they are seldom used directly but often converted to binary vectors by setting non-zero entries to one. Another conversion strategy relating to similarity measures is presented in the next section. Counting vectors are not widely covered in the literature, but they have an important relationship to some of the string methods covered in Chapter 3.

2.1.3 Real valued vectors

Real valued vectors are often called descriptors in the literature as they describe measurable properties of the molecule. An example of such properties could be the molecules log \( p \) value or weight. Many of the descriptors are of a type that can be measured experimentally without the aid of a computer, but for large databases of molecules computational methods are preferred for generating molecular descriptors.

The numerical values used for real valued vectors are often on very different scales, as they measure varying properties. Therefore, the real valued vectors are often normalised by mean centering and giving unit variance to each entry of the vectors. This is done by calculating the mean \( \mu_i \) and standard deviation \( \sigma_i \) for each entry \( i \) in the vectors \( v_1, \ldots, v_m \), and setting the \( j \)th vectors \( i \)th entry \( v_{j,i} \) to

\[
\frac{v_{j,i} - \mu_i}{\sigma_i}.
\]

2.1.4 Distance and similarity measures

There are two main type of numerical measures that can be used to compare vectors: distance and similarity coefficients. As the name suggests, a distance coefficient \( D \) measures the distance between objects; the more the two objects resemble each other, the closer the coefficient is to zero, only taking on the value zero for identical objects. A class of well known distances are the Minkowski distances which are given by the formula

\[
D(A, B) = \left( \sum_{i=1}^{n} |A_i - B_i|^t \right)^{\frac{1}{t}}
\]

which for a \( t \) of one yields the Manhattan (Hamming) distance, and for a \( t \) of two yields the well known Euclidian distance. Minkowski distances are also metrics, but this is not the case for all distance coefficients. To be a metric a distance coefficient \( D \) must satisfy the following four rules for all \( A \) and \( B \):
Chapter 2. Vector similarity

<table>
<thead>
<tr>
<th>Name</th>
<th>Set definition</th>
<th>Numerical definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manhattan</td>
<td>$</td>
<td>A \cup B</td>
</tr>
<tr>
<td>Euclidian</td>
<td>$\left(</td>
<td>A \cup B</td>
</tr>
<tr>
<td>Soergel</td>
<td>$\left(</td>
<td>A \cup B</td>
</tr>
<tr>
<td>Tanimoto</td>
<td>$</td>
<td>A \cup B</td>
</tr>
<tr>
<td>Dice</td>
<td>$2</td>
<td>A \cup B</td>
</tr>
<tr>
<td>Cosine</td>
<td>$</td>
<td>A \cup B</td>
</tr>
</tbody>
</table>

Table 2.1: The table lists six of the most common coefficients used for comparing vectors. The set definitions cover both sets and multisets. For a more thorough discussion including ranges and other theoretical results see Willett et al. [129].

Figure 2.1: An illustration of one way of turning a counting vector into a binary vector.

1. Non-negative: $D(A, B) \geq 0$.

2. Identity of indiscernibles: $D(A, B) = 0$, if and only if $A = B$.


4. Triangle inequality: $D(A, C) \leq D(A, B) + D(B, C)$.

If two non-identical objects $A$ and $B$ can give rise to zero distance, the coefficient is called a pseudo metric.

A similarity coefficient $S$ measures similarity, and take on its maximum value for identical objects; the more the two objects resemble each other, the closer the coefficient is to its maximum value. The similarity coefficient often lies in the interval $[0, 1]$, if not, it can often be mapped to this using normalisation (Section 1.1). If a similarity coefficient maps to this interval, it is simple to convert the measure to a distance by simply extracting it from one.

The binary and counting vectors are often interpreted as sets and multisets respectively. Therefore, each measure presented in Table 2.1 has a definition for the set/multiset and for the real value based vectors. In the set definition column of the table, $|A|$ denotes the number of on bits in $A$, and the intersection $|A \cap B|$ of two multisets is the minimum number of elements present in both multisets. Any multiset can be transformed into a set, by generating new elements corresponding to number of occurrences of original elements, as illustrated in Figure 2.1. This means that algorithms for handling binary vectors are directly applicable on counting vectors with a exponential growth penalty. It is not well defined how to use the set definitions on the real valued vectors, but the numerical definitions can always be used on the binary and counting.
2.2. Related work

vectors; in some cases it even yields the same coefficient, as is the case with the Tanimoto similarity.

One of the most commonly used measures is the Tanimoto coefficient for binary vectors, as it has been shown to perform very well in retrospective screening studies [129]. The Tversky similarity is a generalisation of the Tanimoto coefficient on sets, given by the formula:

\[ S(A, B) = \frac{|A \cup B|}{\alpha|A| + \beta|B| - (1 - \alpha - \beta)|A \cup B|} \]

where \( \alpha \) and \( \beta \) are constants. When \( \alpha \) and \( \beta \) are one, the coefficient is the Tanimoto coefficient, and when \( \alpha \) and \( \beta \) are \( \frac{1}{2} \), the coefficient is the Dice coefficient. In general, if \( \alpha \) and \( \beta \) are identical, the coefficient is symmetric; if they are not, the coefficient is skewed towards either \( A \) or \( B \).

Two coefficients are called monotonic if they produce the same similarity ranking. An example of this is the Tanimoto and Dice coefficients.

2.1.5 Queries

Given a database \( DB \) of vectors, a very common task in chemoinformatics is to extract the vectors in the database with shortest distance or largest similarity to a given query vector \( Q \). There are two main types of queries, namely threshold queries and top-\( x \) queries. Threshold queries return all vectors that have a distance smaller than, or similarity larger than, some threshold \( t \) to \( Q \) under some distance or similarity measure. Top-\( x \) queries return the \( x \) vectors that are closest to or most similar to \( Q \) under some distance or similarity measure. Threshold queries can be used if an estimate of how many similar molecules exist is sought, or if all similar molecules are to be passed on in the next phase of the screening process. Top-\( x \) queries are more interesting if limited resources are available in the next step of the screening procedure and only \( x \) molecules can be tested in the next phase.

It is possible to adapt methods for performing top-\( x \) queries to thresholds queries, by repeatedly adjusting the \( x \) until the least suited returned element is on the other side of the \( t \). Similarly, it is possible to use a threshold based method to answer top-\( x \) queries by repeatedly adjusting \( t \). However, most methods for performing queries can be adapted by maintaining a heap of \( x \) elements, using the distance of the worst element in the heap as a running threshold.

2.2 Related work

Early studies focused on performing retrospective virtual screening studies to verify the merit of vector based representations for virtual screening, while more recent work in the field seems to have been focused on accelerating queries on large databases. Particularly threshold queries using the Tanimoto coefficient have been well studied, as retrospective studies have shown it to be superior to other distance and similarity measures [128]. Therefore, in the following,
the word “query” will always refer to a threshold query using the Tanimoto coefficient.

Several query accelerating data structures for large fingerprint databases have been suggested in recent years. Swamidass et al. proposed organising the fingerprints into subsets according to how many bits are set to one in them [110]. The strategy is based on the observation that, given two fingerprints, the maximum Tanimoto coefficient is the minimum number of bits set to one in either over the maximum number of bits set to one in either. For example, if two fingerprints \( A \) and \( B \) have the number of bits \( |A| = 3 \) \( |B| = 5 \) set to one, the maximum Tanimoto coefficient \( T(A, B) \) between the two is \( \frac{3}{5} \). This bound can be calculated for every subset, and those for which the bound is below the sought threshold can be discarded. The study is mainly focused on deriving theoretical models for predicting how large fractions of the Tanimoto calculations that can be discarded, and little information is given about actual runtimes. However, a later study [7] from the same group suggests that the speedup was on the scale of 2.9 on a data set of 100,000 fingerprints. The study also presents bounds for Tversky and other similarity measures, but most of the analysis and all experiments are on the Tanimoto coefficient.

Another study proposed an extension of the bit bound approach, in which fingerprint summaries were saved along with the original fingerprints [7]. A fingerprint summary consists of a 128 bit XOR compressed version of the fingerprint, and is devised such that it can be used to quickly derive a bound on the coefficient between two given fingerprints, before calculating the coefficient explicitly. Another bound which can be used for fingerprints grouped by the number of on bits in their summaries is also presented. The article is mainly concerned with the theoretical framework but states that, used in combination with the bit bound, the summary strategy yields a speedup of 5.5 when compared to the naive approach on 100,000 fingerprints and a threshold of 0.8.

Smellie proposed something called the compressed binary bit tree [103] which is (although not named so in the study) a trie, a well known data structures from bioinformatics, typically used for storing strings and allowing linear time exact string searches. Storing a large database of fingerprints in a trie allows queries to prune away subtrees, for which it can be proven that no fingerprint in the subtree can have a Tanimoto coefficient above the query threshold. This is done by storing the number of bits matched with the query when traversing the trie, and using this information to calculate an upper bound on the coefficient between the query fingerprint and every fingerprint in the subtree. The study showed that the tries take up a vast amount of memory, rendering them unusable for large data sets. A compression scheme in which sequences of zeroes are replaced with an integer encoding this number was therefore proposed, implemented and tested. On 100,000 fingerprints with a Tanimoto threshold of 0.8 the compressed trie yielded a speedup of 5.7 compared to the naive approach which explicitly calculates all the coefficients, on the tested data. More impressive speedups are archived with higher coefficients. Even though the study is dated later than the two previous studies, the trie is not compared to either the bit bound or the XOR strategy, neither theoretical or practical. Also, the compressed trie only works on sparse data, that is, uncompressed fingerprints:
on sparse fingerprints it is impossible to gain any compression, as there are few
long sequences of zeroes.

As mentioned in Subsection 2.1.4, a similarity can be converted to a distance
by subtracting it from one. The Tanimoto distance on binary vectors is identical
to the Soergel distance which has been proved to be a metric. When taken
on binary vectors, it is trivial to see that the Tanimoto distance obeys the
three first properties to be a metric (non-negative, identity of indiscernibles
and symmetry). The fourth, the triangle inequality, is a bit harder to prove,
and has been the subject of two different proofs [77, 104]. As the Tanimoto
distance is a metric, it is possible to use standard metric data structures such as
vantage point trees (vp-trees) or geometric near-neighbour access trees (GNATs)
for accelerating Tanimoto distance (and thereby Tanimoto similarity) queries.
However, so far no studies have been published in this area.

Baldi and Hirschberg proposed an inequality even tighter than the triangle
inequality [6]. The inequality is based on intersections between fingerprints
and is proven to be strict, in all but the degenerate cases. Although some
suggestions as to how to utilise the intersection bound is presented in the study,
no experiments are performed to verify the quality of the method. A later study
by Nasr et al. presented a method based on small signature vectors on which
the intersection inequality can be used to achieve an efficient bound on the
Tanimoto coefficient [86].

2.3 Research contributions

The majority of my work in screening using vectors has been on accelerating
queries. More specifically, I have focused on Tanimoto queries on both binary
and real valued vectors.

2.3.1 kD grid

The kD grid is a data structure with generalises both the bit bound and the
trie data structure. It was developed in co-operation with Jesper Nielsen of
Aarhus University, and works by storing fingerprints in k dimensional grid cells,
based on how many bits are on in portions of the fingerprint, as illustrated
in Figure 2.2. The data structure in itself is stored as a tree, which in the
case of one dimension (k = 1) degenerates to the bit bound data structure (a
simple list), and in the case of dimensionality equal to the fingerprint length
degenerates to the trie data structure. The kD grid uses an extension of the bit
bound to prune the tree during search; this bound also degenerates to the trie
bound when the data structure degenerates to a trie.

Experiments performed on two million fingerprints generated by the chemistry
development kit (CDK) on the ZINC database indicates a speedup of 2.3
over the naive approach. Although this is not as impressive as the factor of
2.9 reported on the bit bound in previous studies, it should be mentioned that
we were not able to replicate the bit bound results in our experiments. This
is most likely due to the fact that we used another implementation platform
and, more importantly, another data set all together. We also implemented the
Chapter 2. Vector similarity

Figure 2.2: Illustration of the $k$D grid for a $k$ of three.

XOR approach, both the XOR based data structure and the pre-check on pairs of fingerprints. Experiments confirm that it is indeed faster than the 1D grid, but not as fast as the 4D grid. The experiments are described in more detail in the papers referenced in the next section.

I have also programmed a separate implementation in the ChemDB framework at the University of California, Irvine. A set of experiments have been performed on this implementation and the results are promising but a theoretical analysis and conclusion is still pending.

2.3.2 Multibit tree

Each grid cell of the $k$D grid stores a list of fingerprints and runs through these lists in linear time. The knowledge of how many bits are on in all the fingerprints is therefore discarded when running through each list. Jesper Nielsen and I therefore devised the Multibit tree for organising fingerprints in the grid cells. In a Multibit tree, fingerprints are stored in leaves, and every inner node knows which bits are on or off for all fingerprints of the subtree rooted in it. When performing a query in the tree, information about which bits are matched for all fingerprints in the subtree is stored along with information about how many bits are on in all fingerprints in the tree and in the query. This information is used to calculate a tight bound on the Tanimoto coefficient of all fingerprints.
in the subtree.

An implementation was used in the same experiments as for the $kD$ grid, and a speedup of 6.7 was achieved when it was used in combination with the 1D grid, again on two million fingerprints and a threshold of 0.8. It also clearly outperformed the XOR strategy in the same framework. The results of both experiments have been published in the proceedings of *Workshop on Algorithms in Bioinformatics 2009* [64], and an extended version of the article has been published in the journal *Algorithms for Molecular Biology* [65]. The extended article can be found in Chapter 7.

An implementation of the Multibit tree has also been programmed in the Chem-DB framework and experiments have been performed. Initial results indicate that the method is superior to the existing methods on the Chem-DB data as well, but it was difficult to conclude why this is, as no rigorous framework for formally modelling fingerprints existed. A recent paper [8] which presents several formal models for fingerprints might change this and make it possible to perform a formal analysis of the performance of both the $kD$ grid and the Multibit tree.

### 2.3.3 Other binary vector datastructures

As mentioned in Section 2.2, the Tanimoto distance on binary vectors is a metric. However, no experiments have been performed on chemical fingerprint data to assert whether or not standard metric data structures can fruitfully be used to accelerate queries. I have therefore implemented and experimented with the metric data structures vp-trees, $\mu$-trees and GNATs [15, 46, 132]. All three data structures are based on using the triangle inequality to cluster points according to some reference point, and using this reference point as a lower bound on the Tanimoto distance to a query. Furthermore, as the Tanimoto distance only takes on a discrete set of values for a fixed set of fingerprints it is possible to use a metric data structure based on discrete values and I have therefore also been able to implement Burkhard-Keller trees [18]. These are similar to the non-discrete trees, in that they store a reference fingerprint in every node of the tree, but differ in that they store a child for every (discrete) distance between the reference and all fingerprints in its subtree. This often leads to trees with more children, but with a tighter bound for each subtree.

I have also developed a tree data structure (called Intersection tree) based on a similar construction as that of the Bukhard-Keller trees for using the intersection inequality by Baldi and Hirschberg [6]. Fingerprints are stored in the leaves of the tree, and every inner node contains a reference fingerprints which is used to cluster fingerprints according to the size of their intersection with this reference. This intersection size can then be used to effectively prune subtrees when querying by calculating the intersection between the query fingerprint and the reference fingerprint in each node and use the bound presented in the original paper [6].

Finally, I have implemented strategies based on inverted indices, a technique frequently used for solving text matching problems. In inverted indices algorithms, fingerprints are stored in multiple lists according to their on
bits. Given a query, the algorithms select the lists for which the query fingerprint has its bits on and uses $T$-occurrence algorithms (such as divideskip or mergeskip [76]) to extract the fingerprints that occur more than $T$ times in the lists. The algorithms can be used by converting the Tanimoto threshold in a similarity query to a $T$ value using a formula presented in Li et al. [76].

All of these data structures have been implemented and tested in the ChemDB framework. Each of these data structures can of course (and have been) combined with the $kD$ grid in all experiments. All experiments performed in the ChemDB framework have been on uncompressed fingerprints which are sparse, but this does not seem to alter the fact that the $1D$ grid with the Multibit tree seems to outperform the other methods. Results have not yet been published, but current work in the IGB group is working on a C implementation of the Multibit tree for ChemDB.

### 2.3.4 Converting Tanimoto query to distance query

It can be proven that the Tanimoto distance is not a metric when applied to real valued vectors, and it is therefore not possible to use standard metric data structures to accelerate queries directly. However, I have presented a reduction from Tanimoto similarity queries to distance queries in Euclidian space for which there exists a multitude of data structures for accelerating queries, e.g. from computer graphics and animations. Given a query fingerprint $Q$ and threshold $t$, it is possible to construct a new point, $C(Q,t)$, and a sphere radius, $r(Q,t)$, such that for every point $A$, $\|C(Q,t) - A\| \leq r(Q,t)$ holds if and only if $T(Q,A) \geq t$, using the formula

$$C(Q,t) = \frac{t + 1}{2t}Q$$

$$r(Q,t) = \frac{\sqrt{-4t^2 + (t + 1)^2}}{2t} ||Q||$$

as illustrated in Figure 2.3. If the threshold $t$ is less than zero, a similar result holds. This result along with the proof has been published in Journal of Mathematical Chemistry [63] and can be found in Chapter 8.

I have also performed experiments on real valued vectors generated by CDK and Molegros Virtual Docker [83] over a subset of the ZINC database. The vectors were stored in vp-trees and GNATs, which are usable for general metrics, and in $kD$ trees [12, 15, 132], which are tailored for Euclidian space. The vp-trees and GNATs were altered slightly, to take advantage of the fact that it was known they operated in Euclidian space. Experiments indicated that vp-trees and GNATs are vastly superior to a simple linear scan and to $kd$ trees. A more detailed account of the experiments and their results have been published in the Workshop on Algorithms in Bioinformatics 2010 [71] proceedings and can be found in Chapter 9. As the limit is applicable to real valued vectors, it is of course also applicable to binary vectors and experiments on fingerprints were therefore also performed. However, they did not yield any speedup and they are therefore not included in the article.
2.4 Summary

Vector based screening methods are highly efficient screening procedures, which can be used to rapidly identify similar molecules. This chapter presented three main vector types, namely binary, counting and real valued vectors. The relationship between these was discussed, and different distance and similarity measures were presented. Threshold and top-$x$ queries were presented and conversion between the two discussed. Studies concerning the acceleration of threshold queries by using the bit bound, the XOR bound and tries were discussed and briefly compared. It was also briefly mentioned that the Tanimoto distance on binary vectors is a metric, as it obeys the triangle inequality, and an intersection bound sharper than the triangle inequality was also presented. The main research contribution were novel data structures for accelerating Tanimoto queries on binary vectors, namely the $k$D grid and the Multibit tree. Experiments have also been conducted on vp-trees, $\mu$-trees, GNATs, Burkhard-Keller trees, Intersection trees and inverted indices lists, all yielding lower speedups than the Multibit tree on 1D grids. Finally, a conversion between Tanimoto queries and distance queries was presented, along with a brief presentation of experiments and results with these.

2.5 Future work

Most of the work I have performed in accelerating Tanimoto queries have been focused on measuring performance and behaviour of data structures on real data. Little work has gone into formally analysing results using theoretical
models, and this is definitely an area where there is room for further research. However, for these methods to be applicable for drug discovery companies what matters is not theoretical lookup time, but that the techniques are available and fast in practice.

It is possible to show that both the kD grid and the Multibit tree can be used to perform Tversky queries, but no experiments concerning this have been performed and the proof has not been documented. Whether or not they can be used to accelerate other similarity measures have not been examined. Both of these areas are interesting, as they will allow the data structures to be more generally applicable. Also, the Multibit tree could be used for directly calculating the Tanimoto coefficient while searching which might reduce the running time even further.

The experiments I performed were on two million fingerprints, but there are currently more than eight million commercially available molecules on the market. The experiments were kept at two million because of hardware limitations: it was not possible to keep a tree of more than two million fingerprints in main memory. Moving parts of the trees to disk would remove this problem, but it would also require carefully constructed IO efficient strategies for ordering and accessing the data. Another approach might be to distribute the searches and calculations using a standard cluster or a map-reduce approach such as the Hadoop system [125].

Finally, some of the implemented data structures are currently being adapted to the ChemDB website to aid researches in computable medicine. Finishing this implementation project would move the data structures from the more theoretical part of chemoinformatics to a more applied subbranch of the field.
Chapter 3

String similarity

The vector representations deliver a summary of a given molecule, but they might not be unique (at least not without getting excessively large), and they are therefore not suited for being used as unique keys into a molecule database. The simplified molecular input line entry specification (SMILES) developed by Arthur and David Weininger in the 1980s has a canonical form and can therefore be used in a database setting [124]. The specification has since undergone some minor alternations, most noticeably by Daylight and the Blue Obelisk, the latter resulting in the OpenSMILES specification. Furthermore, it is easily human readable unlike the InChI format devised by IUPAC, and easy to write by hand (when non-canonical). A SMILES string is generated by writing a sequence of letters, one for each atom type, marking branches with parentheses and rings with numerical indexes. As an example, consider the visualisation of 3-cyanoanisole in Figure 3.1, which can be represented by the SMILES string COc(c1)cccc1C# N. The main path of the molecule is the string COccccccC\# N, the hash-mark symbolising a triple-bond. (c1) marks the branch containing just one carbon atom, and the number “1” here and later in the path defines the bond between the two carbon atoms. There is of course more constructs and the canonical forms have different rule sets for generating unique strings for every molecule.

Aside of being used for unique identifiers and convenient molecule generators SMILES strings have also been used for generating Lingos: a multiset over a fragmentation of the SMILES string of a molecule. These Lingos can be used for predicting biochemical features and (just as for the vectors) rapidly calculating

![Figure 3.1: Illustration of a possible SMILES string for 3-cyanoanisole. The primary backbone is highlighted with thick lines. c1 indicates the two points where the ring is merged.](image-url)
Chapter 3. String similarity

The material in this chapter covers my work with the Levenshtein distance on SMILES strings, and my work with using inverted indices for calculating the LINGOsim similarity. The results from my Levenshtein experiments are unpublished, but the developed LINGOsim calculation methods have been accepted in *Journal of Chemical Information and Modeling* [66], on the condition of some minor modifications. A revision of the first submitted manuscript can be found in Chapter 10.

This chapter will first cover SMILES strings and Lingos, along with some applications of these. Existing methods for efficiently calculating similarity of Lingos will be presented; and finally, a novel method for using SMILES strings for screening (not using Lingos) will be introduced, along with a Lingo similarity calculation acceleration study.

3.1 Background

SMILES strings are obtained by writing up the atoms (nodes) in a depth first traversal of a molecule (graph). The molecule is prepared by removing hydrogen atoms and, during the traversal, marking back-edges with running indexes, rendering it possible to reconstruct rings (cycles). Atoms are represented by their periodic table abbreviation enclosed in brackets, e.g. [Au] for gold. These can be omitted for B, C, N, O, P, S, F, Cl, Br and I, in which case implicit hydrogen is assumed. Explicit hydrogens and charges are written inside the brackets and aromatic atoms are written in lower-case. Adjacent atoms in the line specification are assumed to be single-bonded, whereas double and triple bonds are annotated by placing a = or a # between the bonded atoms. Numbers are used for joining rings that have been split, using the %-symbol to distinguish multi-digit indexes from multiple single indexes in molecules with many interconnected rings. Table 3.1 illustrates the 2D structure and SMILES string for three well known small molecules.

A multitude of programs for generating 3D structures from SMILES strings exists. All of these rely on energy minimisation procedures for generating realistic 3D positions, rendering the methods using them slow. An attempt to overcome this problem has been made in creating Lingos for predicting ADME (how medication is absorbed, distributed, metabolised and excreted) properties, and for measuring similarity between molecules [119]. Given a SMILES string, its multiset of Lingos is generated by first replacing all occurrences of “Cl” and “Br” with “L” and “R”, and all numerical literals with zero. Next, the string is split into \( n - q - 1 \) consecutive substrings, each of length \( q \) (four in the original study) as illustrated in Figure 3.2. This can be done in linear time in the length \( n \) of the original SMILES string, rendering it a very fast procedure if SMILES strings are already known. The resulting multiset, called the Lingos of the SMILES string, will vary according to which SMILES generator has been used. However, they are comparable if the same canonical method is used for generating the SMILES strings for a set of molecules. One interesting interpretation of Lingos is as counting vectors where each entry corresponds to...
3.1. Background

Table 3.1: 2D structures and SMILES strings for the three molecules vanilin, nicotine and thiamin. SMILES strings are taken from Wikipedia’s SMILES entry.

![SMILES strings for vanilin, nicotine, and thiamin](image)

<table>
<thead>
<tr>
<th>SMILES</th>
<th>LINGO freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{S} = \text{c1cccccc1Cl} )</td>
<td>( \text{c0cc} ) 1</td>
</tr>
<tr>
<td>(a) Example SMILES</td>
<td>( \text{0ccc} ) 1</td>
</tr>
<tr>
<td>( \text{S}' = \text{c0cccc0L} )</td>
<td>( \text{ccc0} ) 1</td>
</tr>
<tr>
<td>(b) Simplified string</td>
<td>( \text{cc0L} ) 1</td>
</tr>
<tr>
<td>(c) Lingos</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.2: A multiset of Lingos is generated from a canonical SMILES string. The multiset contains two occurrences of the string cccc.

Tanimoto similarity between Lingos (also called LINGOsim) can be used for performing a screening, simply by performing a query using the LINGOsim similarity for finding molecules similar to a known active. Another interesting problem related to Lingos, is the effective calculation of the Tanimoto similarity both between single pairs of Lingos, but also for all pairwise similarities in a set of Lingos. Both of these problems will be discussed further in the next section.
3.2 Related work

It is not immediately obvious how to efficiently calculate the LINGOsim between two Lingos. One strategy proposed by Grant et al. builds a pseudo finite state machine from one of the Lingos, and runs the other Lingos through this [38]. The finite state machine (FSM) is constructed by building a trie over all the Lingos, such that the Lingo generated from the root to a leaf at depth \( q \) stores the number of occurrences in the SMILES string. The article does not make any claims as to the time complexity, but it seems that given the FSM the procedure runs in time \( O(nq) \) where \( n \) is the length of the original SMILES string and \( q \) is the Lingo fragment size. The pre-processing can also be done in time \( O(nq) \), rendering this an optimal algorithm for pairwise LINGOsim calculations when the input size is measured as \( nq \). However, it can be argued that the Lingos need not take up more space than \( O(n + q) \), as they need not be generated explicitly in which case the algorithm is suboptimal. The trie can be both generated and used to perform the LINGOsim calculation in \( O(n + q) \) time, which is \( O(n) \) assuming \( q \leq n \), rendering the algorithm optimal.

More interestingly, the study also examined how suitable the LINGOsim measure is for performing a screening study, by examining how well it identifies known actives compared to Daylight fingerprints. The study showed very good performance, the LINGOsim measure often performing superior to Daylight fingerprints. There are very few cases in which Daylight fingerprints are superior, namely when molecules are very similar, but the canonical rules for generating the SMILES strings results in very dissimilar Lingos. The study also confirms that \( q = 4 \) is well suited for performing screenings.

Another screening study combined the rapid calculation of LINGOsim between potential binders with Autodocks slower receptor docking procedure [120]. The presented method samples a subset of a ligand database and docks all of these towards a target receptor, yielding a measure of binding affinity. The best ligands were used for identifying a new population of potential binders by a pseudo-evolutionary algorithm, in which the ligands “adopt” the new population by selecting the ligands most closely related to the current population, measured using LINGOsim. Experiments demonstrate that the procedure is able to identify 62% of the potential hits, after docking only 6.5% of the entire ligand database without any prior information about known binders. An obvious generalisation of the procedure is to allow other pairwise similarity measures and docking procedures, but this has not been covered in the Autodock study. In light of the previously described results, in which very similar molecules have a very different LINGOsim coefficient, it would have been very nice to see a study using the same framework, but a fingerprint based method for adopting the next generation of the algorithm, thus potentially avoiding getting stuck when SMILES strings for similar molecules are dissimilar.

A recent study introduces a novel algorithm for rapidly calculating the LINGOsim by representing each Lingo fragment as a 32 bit integer, and running through sorted lists of these integers, along with lists of their occurrence count [41]. One 32 bit integer can represent four bytes, one for each ASCII character in a fragment of size four, which restricts the algorithm to be only
linear in $n$ in the cases for which $q$ matches the word size of the architecture. The algorithm is implemented on both CPU and GPU, yielding a speedup of two (on a single kernel) for the CPU, and a speedup of 30 on a Tesla T10 GPU when calculating the matrix of all against all LINGOsims.

3.3 Research contributions

There are two primary string similarity methods I have worked with. I have developed a method for calculating the similarity between molecules by using the edit distance of their SMILES strings, and I have worked with accelerating the LINGOsim similarity matrix for a set of molecules efficiently. Both topics will be covered in this section.

3.3.1 SMILES string edit distance

The edit (or Levenshtein) distance [75] is a distance measure between strings which measures the minimum number of insertions, deletions and substitutions required to transform one string $a$ of length $n$ to another string $b$ of length $m$. An optimal way of transforming the string Saturday into Sunday is by removing at and replacing r by n, yielding an edit distance of three.

\begin{align*}
\text{Saturday} \\
\text{S--unday}
\end{align*}

The edit distance has been used in gene comparison, where the strings are over the alphabet of the four letters A, C, G and T, representing the four nucleotides in DNA. Pairs of DNA segments that have a small edit distance are most likely more closely related, than pairs with a large edit distance, and the edit distance can therefore be used to infer the ancestral relationship between a set of species. The edit distance between two strings $a$ and $b$, can be calculated using a recursive formula [51]

\[ d_{i,j} = \min \begin{cases} 
0, & i = 0 \land j = 0 \\
 d_{i-1,j} + 1, & i > 0 \land j \geq 0 \\
 d_{i,j-1} + 1, & i \geq 0 \land j > 0 \\
 d_{i-1,j-1}, & i > 0 \land j > 0, a_i = b_j \\
 d_{i-1,j-1} + 1, & i > 0 \land j > 0, a_i \neq b_j 
\end{cases} \]

where $i$ is an index in $a$ and $j$ is an index in $b$. The table $D$ of size $n \times m$ can be filled with the values of $d$ in time $O(nm)$, as each entry in the table can be calculated in constant time using the field immediately above, immediately to the left, and left above (see Figure 3.3).

As observed by the Lingo studies, similar molecules yields similar SMILES strings, but the LINGOsim measure can scramble up the main path of a SMILES string. This might have the effect that information about the overall structure of the molecule is discarded. I have examined whether the edit distance can be used to compare SMILES strings and if the resulting distance measure is suitable for performing a screening. Experiments were performed on the Directory
Chapter 3. String similarity

of Useful Decoys (DUD) data set with SMILES strings extracted using CDKs canonical SMILES generator. As in Lingo construction all numerical literals are replaced by zero. The rankings were compared to those of a descriptor method with the Tanimoto coefficient, and a largest common subtree. It was observed that the edit distance method performed comparable to these two methods, outperforming them on some of the targets, when measured using the one percent enrichment. More detailed results can be found in my progress report [61], no further attempt at publishing the results have been performed as the results did not outperform existing methods greatly.

3.3.2 Rapid LINGOsim calculation

As briefly mentioned in Section 3.1, Lingos can be regarded as counting vectors where each entry corresponds to a SMILES fragment. For a set of SMILES strings, these counting vectors can be generated by maintaining a trie of observed $q$ length fragments with a unique identifier in every leaf. The counting vector solves the word size restriction from the previously mentioned similarity matrix calculation study, and the same algorithm can be used for arbitrary $qs$, as long as the number of unique $q$-length fragments in the data set does not exceed the word size, in which case more than one register is needed to store a fragment. With these vectors, the Tanimoto similarity can be calculated efficiently using the $O(n + m)$ approach described in Haque et al. [41], which is optimal for a pair of Lingo multisets. Furthermore, the counting vector approach makes it possible to convert the Lingos into binary vectors using the technique discussed in Section 2.1 which has been done in this study. The extra storage needed is fairly little as there are few identical fragments observed in every single SMILES string. The resulting binary vectors are called the verbose representation.

Three methods for calculating the similarity coefficients have been tested:

- The verbose representation can be interpreted as a binary vector, and standard methods from Tanimoto similarity calculation can be used. This is done in time $O(s^2n)$ where $s$ is the number of SMILES strings and $n$ is the average SMILES string length.

- Another strategy is to store the binary vector representations of the Lingos in a Multibit tree and traverse this tree for every row in the similarity
3.3. Research contributions

\[ x_1 : 0, 1, 2, 3 \quad x_2 : 0, 2, 3, 4 \]
\[ x_3 : 2, 4, 5 \quad x_4 : 1, 3, 4 \]
(a) Input verbose representation.

\[
\begin{array}{cccccc}
I_0 & I_1 & I_2 & I_3 & I_4 & I_5 \\
\hline
1 & 1 & 1 & x_2 & x_3 \\
x_2 & x_4 & x_2 & x_3 & x_3 \\
x_3 & x_4 & x_4 & & & \\
\end{array}
\]
(b) Inverted indices datastructure.

\[ x : 2, 4, 5 \]
(c) Query verbose representation.

\[
\begin{array}{cccc}
1 & 2 & 3 & 1 \\
\end{array}
\]
(d) The \( C \) counting vector.

Figure 3.4: Illustration of the inverted indices data structure on four sets (a, b), each with three or four elements (e.g. Lingo ids or fingerprint entries). Given a query \( x \) (c), the counting vector (d) is calculated by traversing the lists \( I_2, I_4 \) and \( I_5 \).

matrix, only counting highly shared bits stored in internal nodes once. If all bits are shared between all vectors this algorithm runs in time \( O(sn + s^2) \) and if none are shared it runs in time \( O(s^2n) \) and it is therefore only effective if many bits are shared.

- If few bits are shared a better strategy is to store the binary vectors as inverted indices and traverse the lists corresponding to the bits set for each rows molecules binary vector. An intersection count is kept between the target vector and each of the other vectors, and this intersection count is used to calculate the Tanimoto coefficient. Figure 3.4 illustrates the inverted indices data structure and the calculation the intersection sizes. More details can be found in Chapter 10. With this strategy the similarity matrix can be calculated in time proportional to the size of all intersections in the data set which is \( O(sn + s^2) \) when no bits are shared and \( O(s^2n) \) when all bits are shared, rendering it the multibit trees dual.

Experiments in which the similarity matrix is calculated have been performed on the three presented strategies on data from Lingos (generated from ZINC with CDK SMILES generator and a \( q \) of four), IMDB (\( q \)-grams of actor names) and compressed and uncompressed fingerprints (generated from ZINC with CDK, 1024 bits for compressed).

The experiments indicated that the inverted indices method was superior on all data sets, and when run on Lingos, the algorithm performs ten times better
than the strategy in Haque et al. and even outperforms their implementations running on a Tesla T10 GPU. An article containing the inverted indices method and its performance on Lingo data has been submitted to *Journal of Chemical Information and Modeling* [66]. The article has been accepted on the condition of some minor modifications. A revision of the first submitted manuscript can be found in Chapter 10.

### 3.4 Summary

This chapter introduced the SMILES string representation of molecules, motivated its existence and briefly summarised how it is derived from a molecule. It went on to introduce Lingos which are multisets of substrings from modified SMILES strings along with the LINGOsim measure. Previous studies concerning the merit of the LINGOsim as a tool for screening and studies concerning the acceleration of LINGOsim calculations were introduced. Lastly, novel contributed research concerning both SMILES strings and Lingos was presented, namely a screening study using the edit distance of SMILES strings and an acceleration study which used inverted indices for rapidly calculating the LINGOsim similarity matrix.

### 3.5 Future work

An obvious study would be to compare the SMILES string edit distance method with the LINGOsim method, to see how highly they are correlated, and if the information loss when going from SMILES to Lingos are an advantage or a disadvantage.

Two novel strategies for calculating the similarity matrix have been presented, and both have cases in which they obtain the optimal asymptotic running time of $O(s^2 + sn)$. Whether there exists an algorithm which always yields an optimal result remains to be seen. Such an algorithm would also be usable for binary and counting vectors, potentially making it possible to analyse very large data sets, by using similarity matrix based methods such as clustering algorithms.

Finally, as previously mentioned the LINGOsim measure has been used in an pseudo-evolutionary algorithm where molecules were adopted based on their Lingos similarity. Another study has demonstrated that similar molecules do not always have similar Lingos which has the potential of trapping this approach. A study in which the method is generalised to other computationally fast similarity measures such as fingerprint comparison is clearly needed to assess if the Lingos are appropriate for exploring neighbourhoods of biologically similar chemical compounds.
Chapter 4

Graph comparison

Vector and string based methods have been shown to yield very high enrichment in some virtual screening studies [38, 44], but unfortunately they do have some limitations. Fingerprint methods were originally developed for substructure searching, and was never intended to be used as virtual screeners [73]. Furthermore, both vector and string similarity methods capture local information about the investigated molecules and fail to investigate similarities on a larger scale. They are mainly suited for identifying structural close analogues, which makes it difficult for them to expand their search into compounds with similar biochemical properties and dissimilar structure. Another way of putting this is that the methods are unable to perform scaffold hopping. That is, they can not escape a patent space to identify unpatented novel drugs, and they are unable to identify novel lead series which can be investigated when existing lead series have poor absorption, distribution, metabolism, and excretion properties. Some 3D methods address these problems, and studies have shown that these can find different complementary sets of active compounds to those found by vector and string methods [80]. However, these methods are very time consuming, especially on very large databases and they require the investigated molecules to be defined in some rigid conformation(s).

Research into graph based methods tries to address these issues by developing methods that represent molecules as interconnected structures containing both local and global information without the need of conformational information. Going from vectors to graphs introduces a computational overhead but the methods are often faster than the 3D methods and can yield better results. Some existing methods represent molecules as trees to make use of even faster algorithms for performing virtual screening.

The material in this chapter covers my poster presentation Largest Common Chemical Feature Subtree as a Virtual Screening Method [72] that is also described in an unpublished manuscript [67], and my work with Practical Research Project student Anders Johnsen that is described in his final report [50].

This chapter will focus on graph based methods by first covering some basic methodology and techniques; next it will cover related work in the field; and finally it will present the research contribution of this dissertations work: namely work with largest common subtrees and clique histograms.
4.1 Background

There are different levels of representations for defining graphs over molecules. The simplest is to define a graph where nodes correspond to atoms of the molecule and edges correspond to covalent bonds. These nodes can be annotated with their atom type and the edges with binding order. However, different atoms might have the same property (such as hydrogen donor) depending on local structure of the molecule and nodes with the same atom type might have different properties (such as ring and non-ring atoms), again because of local structure. In the field, this has given rise to feature graphs in which nodes are annotated with feature types such as “hydrogen acceptor” and “aromatic ring atom”. Two examples of this are feature trees and reduced graphs. Feature trees merge ring atoms to generate tree representations of molecules, in which nodes are annotated with either steric or chemical types. Different levels of tree representations exists, ranging from representing ring systems as nodes and other atoms as individual nodes, to representing the entire molecule as one node. Reduced graphs are defined on four different levels where the information stored in nodes ranges from simple ring information to bonding possibility information. In the transition from level to level nodes can get merged yielding simpler graphs which are easier to handle.

Given graph representations of a set of molecules, one possible comparison strategy is to find the maximum common sub graph between these. If the graphs are trees, polynomial time algorithms for calculating the maximum common subtree can be used. If, however, the graphs contain cycles the problem of finding (the size of) their maximum common sub graph becomes NP-complete and therefore much harder. Algorithms based on clique detection have been used for solving the problem on molecular graphs. These algorithms work by, given two graphs $G_1$ and $G_2$, constructing a correspondence graph in which nodes are the Cartesian product of the nodes from $G_1$ and $G_2$. Nodes in the correspondence graph are connected if and only if the nodes they represent have the same edge distance in the original graphs, and if they are compatible, e.g. by both being hydrogen donors. A maximum clique in the correspondence graph corresponds to a maximum common sub graph between the two original graphs. Please consult Figure 4.1 for an example. Once the size of the maximum common sub graph has been calculated it can be normalised by using the normalisation techniques in Section 1.1.

4.2 Related work

There has been several studies investigating feature trees and reduced graphs. The paper introducing feature trees [93] defined two comparison algorithms called split-search and match-search. Both of these match edges according to some metric, and normalisation was applied before ranking the molecules. The results were tested on a database of ten targets, yielding results comparable to that of Daylights fingerprints using the Tanimoto coefficient. A later study [94] used dynamic programming to accelerate match-search, yielding a speedup of
4.2. Related work

Figure 4.1: Two graphs and their associated correspondence graph. The highlighted parts of $G_1$ and $G_2$ corresponds to the highlighted clique in the correspondence graph, where $b$ is matched with $i$, $c$ with $j$ and $d$ with $k$. several orders of magnitude. The study also validated that the algorithm was able to identify known actives.

The article introducing reduced graphs [112] defines a similarity measure based on largest common sub graphs and finds that it is able to identify antihistamines. As stated earlier general graph comparison is NP-complete, and other methods have therefore been investigated. For example, a later study [35] used the reduced graphs as templates for generating pseudo-SMILES which were used to generate fingerprints using Daylight. These fingerprints were compared to normal Daylight fingerprints in a screening experiment on a data set using the Tanimoto coefficient. The method had a slightly lower enrichment, but it was able to identify unique hits, not found by Daylights fingerprints. Using the pseudo-SMILES to generate fingerprints looses much information, and a later study [11] therefore defined a set of fingerprints based on paths in reduced graphs. Experiments, in which data fusion was used to combine Daylight fingerprints with reduced graph fingerprints, resulted in better result in 10 out of 11 cases, with up to 65% more known actives found than by either techniques. Another way to address the slow reduced graph comparison algorithm is to con-
Chapter 4. Graph comparison

Figure 4.2: An example of a conversion from a molecule to a tree. Ring systems are collapsed, but the number of atoms in the ring system is saved in the resulting node.

struct limits for the maximum similarity achievable. By using two such limits a procedure called RASCAL was developed [95] to filter out dissimilar graphs without explicitly calculating their similarity. An implementation revealed a significant speedup. Using this procedure a later study [10] demonstrated that it was possible to perform scaffold hopping and identify known active binders dissimilar to the query molecule. Furthermore, the program had a higher average enrichment than Daylights fingerprints and found more unique classes than these.

4.3 Research contributions

In my studies I have investigated both tree and graph based methods. This section presents a traditional tree based method and a graph based method based on distributions.

4.3.1 LarCCS

The Largest Common Chemical Subtree (LarCCS) method is based on a tree representation, in which all ring systems have been replaced with nodes. If two ring systems share atoms they are collapsed into the same node to avoid cycles in the resulting tree, as illustrated in Figure 4.2. Besides these ring nodes the tree contain a node for each non-hydrogen atom. Each node stores three attributes: the number of atoms in the node, their combined charge and donor acceptor properties. This representation is similar to feature tree, but nodes contain more fine-grained information and not every ring is stored in its own node.

The trees are compared using a variation of maximum common subtrees, in which nodes are allowed to be paired if their attributes agree, i.e. the number of atoms they represent is similar, their charges do not deviate more than a certain threshold and their hydrogen donor/acceptor properties are similar. The maximum subtree is calculated using a dynamic programming algorithm similar to the match-search algorithm. The algorithm examines pairs of oriented edges, matching their subtrees using the Munkres worker-task assignment algorithm.
4.3. Research contributions

The maximum common subtree sizes are normalised using the product normalisation from Section 1.1.

The method has been implemented and tested on the DUD and MUV data sets. The resulting rankings were compared to those generated by CDK fingerprints and Chemical Feature Distance Matrix (CFDM) descriptors. CDK fingerprints are implemented in a manner similar to Daylight fingerprints and were compared using the Tanimoto coefficient. The CFDM descriptors is developed by Molegro [83] and describe distances between functional groups in a molecule and are compared using the Tanimoto coefficient. Results from the experiments revealed that the MUV data set was much harder than the DUD set for all methods. LarCCS was superior to the CFDM method and comparable to the CDK fingerprints when measured on the one percent enrichment and AUC values. Further investigations revealed that it clearly supplemented the CDK method by finding unique hits; this finding is similar to those found for reduced graphs.

Details concerning the methods, experiments and results can be found in the unpublished manuscript Virtual Screening using a Largest Common Chemical Subtree Method [67] in Chapter 11. The results were presented at a poster session at the 17th Annual International Conference on Intelligent Systems for Molecular Biology and the 8th European Conference on Computational Biology [72].

4.3.2 Clique distribution

This work is based on the graph defined by a chemical compound without hydrogen atoms. That is, nodes are atoms annotated with the atom type and edges are bonds. Unlike the reduced graphs the nodes are not classified according to biochemical features. Similar to RASCAL, the method constructs a correspondence graph in which the largest cliques are found using the Bron-Kerbosch algorithm [17]. The Bron-Kerbosch algorithm yields all non-expandable largest cliques from the correspondence graph which corresponds to all the possible subtree matchings. In the RASCAL algorithm the normalised size of the largest of these is used as the similarity between the molecules. More fine grained information can be extracted by analysing the distribution of the sizes of these subtrees. If two graphs share a few very large subtrees they are more similar than if they shared many small subtrees as illustrated in Figure 4.3. This skewness can be quantified using a variety of measures such as counting the number of subtrees of size one, or the weighted mean of the distribution.

Experiments in which similarity measures were calculated on the basis of the clique size distribution and used for virtual screening have been performed on the DUD data set. The results were compared to those produced by fingerprint and Lingo methods using the Tanimoto coefficient. The fingerprint method and using the size of the largest clique (as in RASCAL) seemed to yield the best results when measured using the enrichment. However, another similarity measure which looked promising was the standard deviation of the clique size distribution. Closer inspection of the enrichment factors of individual targets revealed that this method often outperformed the fingerprint method.
Chapter 4. Graph comparison

4.4 Summary

Graph methods are based on graph representations which are either trees or general graphs, the nodes of which can be atom- or feature based. This chapter has presented some existing methods, namely feature trees which are trees with feature information in the nodes, and reduced graphs which are allowed to have cycles but which also stores feature information in the nodes. These two representations can be used as screening tools, and existing results which demonstrated that they supplement, and sometimes are superior to fingerprint based methods were presented. This chapter also introduced two new methods, namely the tree based LarCCS method, which stores more information in each node than just a feature class, and the clique distribution method which is based on clique size distributions of molecular graphs without features. Both of these methods are a valuable supplement to existing methods, as they are able to find unique hits in retrospective screening studies.

4.5 Future work

The LarCCS method has been observed to yield results that supplement fingerprint based methods well, and it would be interesting to examine if data fusion methods could be used to improve the screening results, as demonstrated in the reduced graph experiments [11]. This would also give a more fine grained similarity measure than the normalised subtree intersection size. Another way to get a more fine grained similarity measure would be to allow partial nodes match and weight a node match according to the number of matched properties between nodes. Allowing these partial matches could reveal actives which share less feature characteristics but which are still binders. The clique distribution work on molecular graph also presents a novel method for looking at the similarity between two LarCCS trees. Instead of simply measuring the largest
common subtree new measures based on the distribution of largest possible
common subtrees could be used to measure the similarity between trees.

The clique distribution experiments focus only on molecular graphs in
which every node corresponds to an atom. As previously mentioned some
atoms not of the same type might share some properties, and atoms that are
of the same type might not share some properties, depending on the local
structure of the molecule. The methodologies developed for examining clique
distributions could be used on feature graphs to measure the similarity between
these.
Chapter 5

Ligand alignment

The methods described in the previous chapters are cheap to calculate and will return a similarity score well suited for identifying structural similar molecules. However, biochemical characteristics depend on the three-dimensional structure of a molecule along with its properties, such as electrostatics. Leach and Gillet [73] present a good example based on the four opioids in Figure 5.1. The similarity between morphine and the three other molecules measured using the Tanimoto coefficient on Daylight fingerprints is also presented. The high similarity scores between morphine, codeine and heroin indicates the high structural similarity between the molecules. Methadone has a low similarity score, reflecting the fact that it is not structural similar to the other three compounds. It is, however, also active against opioid receptors – a fact not indicated by the simple similarity methods, but which some 3D similarity methods are able to identify.

Three-dimensional similarity methods require one or more conformations which make them more computational expensive; even more so if torsional angles are allowed to be flexible during similarity calculations. Some methods are independent of the alignment of molecules and might use descriptors based on 3D co-ordinates (e.g. USR [9] and ElectroShape [3]) or matrices of inter-atom distances (see Leach and Gillet [73]). My studies have been focused on alignment-methods in which an optimal orientation and spatial placement has to be identified. I will not go into depth with alignment independent methods; for a discussion of these see Leach and Gillet [73].

The material in this chapter covers my poster presentation Optimal Overlay of Ligands with Flexible Bonds Using Differential Evolution [69] and my conference papers Optimal Overlay of Ligands with Flexible Bonds Using Differential Evolution [70] (Chapter 12) and Recombining Angles in Differential Evolution [62] (Chapter 13).

This chapter will introduce alignment methods and different classes of these, based on overlap measures, alignment types and algorithms. It will describe some existing methods and characterise them according to these classes. Finally, it will present my research contributions, covering investigations in Evolutionary Computation for ligand alignment and related problems.
Chapter 5. Ligand alignment

5.1 Background

The goal of alignment dependent methods is to find an optimal alignment between a set of molecules, by maximising some type of overlap between these. Such an alignment depends on the orientation and spatial placement of the molecules. If two or more molecules are to be aligned, one of them can be kept still to reduce the degrees of freedom of the problem. If this reference molecule is bounded to a crystal structure it is possible to investigate if native binding modes can be reconstructed. This is done by examining the deviation from the natively bound ligand in the crystal structure.

There are three main components to an alignment method: an overlap measure, to quantify how well two or more molecules overlap; an alignment type, to specify the terms of the alignment; and an algorithm which seeks to find an optimal alignment. This section presents a brief overview of current measures, alignment types and algorithms.

5.1.1 Overlap measures

The overlap between molecules can be based on atomic types and their distances, or a more feature based methodologies, such as pharmacophores, can be used. A pharmacophore is a set of feature points in 3D, which can be used to identify properties, such as rings or hydrogen donors, which should be matched by screened ligands. Some feature points might be negative features, stating that some parts of space should not be occupied by a molecule. Pharmacophores can be defined from one or more aligned molecules and might be used...
5.1. Background

for guiding a molecular alignment, either by aligning the molecules to pharmacophores or by directly aligning pharmacophores. If a crystallographic structure is known, this can also be used for extracting a pharmacophore.

Several metrics for measuring the alignment of two or more molecules exists. If a mapping between atoms from one of the molecule to the other exists the root mean square deviation (RMSD) can be used. The RMSD between two lists of points $P$ and $Q$, each containing $n$ points, is defined as

$$\sqrt{\frac{1}{n} \sum_{i=1}^{n} ||p_i - q_i||^2}.$$ 

It is often used to measure the fitness of an alignment or of a conformation by calculating the RMSD between a solution and some observed natural configuration. In that case, the mapping between the atoms is obvious (up to molecular symmetries). The RMSD can also be used to calculate the distance between two different molecules in which case an appropriate mapping between atom sets has to be calculated first, e.g. by using a correspondence graph on 3D points and a clique detection algorithm [16]. If pharmacophores are used the RMSD of a feature point pairing can be calculated.

Finding a good mapping for calculating the RMSD is far from trivial, and even with filtering algorithms for removing unlikely matches, the problem is still NP-hard. One atom-mapping independent method is the hard sphere model, in which every atom is replaced with a sphere with some radius, e.g. the van der Waals radius. The quality of an alignment between molecules is then simply the volume of the overlap between their spheres. While being highly intuitive, this measure does have some drawbacks. First of all, it is computational very expensive: if two spheres from one molecule overlap, their shared volume has to be accounted for; if three spheres overlap it becomes even more complicated. This makes calculating the overlap quite complicated and fairly expensive. A more subtle problem has to do with calculating the optimal orientation and spatial placement which can not be solved analytically and for which heuristic optimisers therefore often are used. These rely (explicitly or implicitly) on gradient information to guide their search, but when two spheres do not intersect, the value of the overlap is simply zero, no matter if the spheres are 10 Ångström or 10 meters from each other.

Using Gaussian [39] approximations solve both of these problems by defining a cheap measure which has gradient information – even when two atoms are far apart. The volume of a molecule when calculated using Gaussians can be parameterized to approximate that of a hard sphere model within one percent [39], while still having non-zero volume for non-overlapping hard spheres. However, blindly replicating the hard sphere overlap might not yield optimal results when performing a virtual screening. A recent study [20] uses a comprehensive set of aligned ligands binding in the same pocket to optimise the parameters of a Gaussian based method with impressive results. The Gaussian overlap can be calculated with different expansions, the trade-off being between accuracy and computational cost.
Chapter 5. Ligand alignment

5.1.2 Types of alignments

There are different ways of categorising alignment types, one of which is to differentiate between pairwise and multiple alignment. In pairwise alignment, exactly two molecules are aligned, whereas multiple alignment seeks to align several molecules, typically to derive a common pharmacophore. By nature, multiple alignments are more expensive than pairwise alignments. A multiple alignment can be performed simultaneously on all the molecules, or it can be derived from multiple pairwise alignments, either to one selected target molecule or by trying different references, e.g. by using the work of Jones et al. [53]. Any multiple aligner can of course be used as a pairwise aligner.

Another way to categorise alignment types is to differentiate between rigid and flexible molecules. By keeping molecules in a rigid conformation the degrees of freedom are reduced significantly. These rigid conformations will typically be generated using conformation samplers, such that multiple representations of the same molecule is used in multiple alignment runs. If the molecules are allowed to be flexible during alignment, the degrees of freedom increase and another optimisation objective is introduced, namely the energy of the molecules. If a conformation of a molecule in its active state is known, e.g. from crystallographic structures, it can be held rigid and tested molecules can be flexible. Typically, for both conformation and flexible alignment, molecules are built using standard fragments (such as rings), and the only free parameter is rotations around torsional angles, as illustrated in Figure 5.2. The angles induce a strain on the system and moves the atoms of the molecules closer to or further away from each other. The total energy of the system is typically divided up into the following terms (Appendix A2, [73])

\[
\sum_{\text{bonds}} \frac{k_i}{2} (l_i - l_{i,0})^2 \sum_{\text{bonds}} \frac{k_i}{2} (\theta_i - \theta_{i,0})^2 \\
+ \sum_{\text{torsions}} \frac{V_n}{2} (1 - \cos(n\tau - \mu)) \\
+ \sum_{\text{atom pairs}} \frac{q_i q_j}{4\pi \epsilon_0 r_{ij}} + \sum_{\text{atom pairs}} 4\epsilon_{ij} (\frac{\sigma_{ij}}{r_{ij}})^{12} - (\frac{\sigma_{ij}}{r_{ij}})^6
\]
5.1. Background

The two first terms are related to bond length and bending, which is not relevant if only flexible torsional angles are allowed. The third term penalises unfavourable rotations around torsional angles, parameterized by $V_n, n$ and $\mu$ for different systems. The last two terms define non-bonded interactions due to electrostatic and van der Waals interactions. All terms are approximations and their parameters are derived from experiments or quantum mechanical calculations. The energy model here is more rough that quantum mechanical models and can take different forms than the one presented here. Also, the parameters can have different values, yielding different force fields. The energy and overlap of a system can be combined by using a simple scale function, and the total value of an alignment is

$$s \cdot \text{overlap} - (1 - s) \cdot \text{energy}$$

where $s$ is some scalar between zero and one. Note that the overlap is sought maximised and the energy is sought minimised. The energy and overlap can also be used as two conflicting goals in a multi objective optimisation in which an optimal answer is to be found on the Pareto front. Such methods will not be covered in this dissertation.

5.1.3 Algorithms

The different alignment score measures give rise to different algorithms for finding an optimal alignment. If the objective is to minimise the RMSD, one possibility is to use Kabsch algorithm [57, 58], which is an analytical form for finding an optimal rotation by calculating the singular value decomposition. As the RMSD measure itself, this method requires a mapping between points. Furthermore, it does not change the spatial placement of the molecules, which are often centered around their atomic mass center before using the algorithm. Finally, the method is only applicable to pairwise rigid alignment.

If there is no obvious mapping between two sets of points (atom positions or feature points) one option is to try all pairings. If every point is allowed to match any other point this will result in an exponential number of pairings. One way to avoid this is by defining a maximum distance deviation for pairs of points. If, for example, the distance between two points $p$ and $q$ from one molecule is 4.5, and the distance between two points $p'$ and $q'$ from another molecule is 5.6, then the pairing $(p, p'), (q, q')$ is only legal if the maximum deviation is 1.1 or above. Given such a maximum deviation, a correspondence graph can be built and clique detection can be used to find optimal pairings. This is analogous to the correspondence graph over feature graphs as described in the previous chapter, and the graph is defined in a similar manner: each node corresponds to a pair of points from the two molecules, and nodes are connected if the distance the represent deviates less than the specified threshold. Also, edges can be omitted if the points do no match, due to different atom or feature types. For a more detailed description, see Brint et al. [16]. The generated pairings have the drawback that they are dependent on the threshold which is chosen more or less arbitrarily. If the value is too small the pairings can be too restrictive, and the lowest RMSD values might be overlooked by the Kabsch algorithm. If the
Chapter 5. Ligand alignment

Figure 5.3: Illustration of how the simplex is used to select a new point for the search in NM.

value is too high the algorithm will generate all possible pairings, and nothing will be gained by running the algorithm.

A number of methods for aligning using a pairing independent overlap measure exist. One of the simplest uses the moments of inertia to align sets of points by aligning the axis defined by these. This will only yield one solution, which has very little probability of being optimal by any of the overlap measures. However, the solution can be used as a starting point for another class of pairing independent optimisers, namely heuristic numerical optimisers of which this chapter will cover three: Simulated Annealing (SA), Nelder-Mead (NM) and Evolutionary Computation (EC).

Simulated Annealing [59] is a variation of the Metropolis-Hastings algorithm, which is a Monte Carlo method. It is inspired by annealing in metallurgy where metals are heated and then cooled to align bonds and increase durability. The method maintains a single solution point, and in every step of the algorithm a new point is sampled from the neighbourhood of the original point. The algorithm moves to the new point if it gives rise to a better solution or according to some probabilistic process. The possibility of going to a worse state decreases during the execution of the algorithm, according to some temperature parameter $T$. The strategy will therefore perform a global search at first, but will move on to a local search.

Given an objective function in $\mathbb{R}^n$, the Nelder-Mead method [87] maintains $n + 1$ solution points. These span a simplex, which is used for generating a new point in each iteration, e.g. by reflecting the current worst solution in the simplex as illustrated in Figure 5.3. Many other variations of the general algorithm exist. As the simplex shrinks the algorithm will go from global search to local search and it will eventually converge to a (local) optimum. To avoid premature convergence of both SA and NM, the algorithms are often restarted with new initial point(s).

A broad class of heuristics inspired by natural selection is called Evolutionary Computation. They work by maintaining a population of candidate solutions which are evolved over a number of generations. In each generation, new candidate solutions are generated by letting the current population reproduce using techniques such as gene copying, crossover and mutation, inspired
5.2. Related work

Many different methods for aligning ligands have been implemented and described in the literature. This section will cover some of these, and it will classify them using the terminology introduced in the previous section.

Many early methods were dependent on a point map being present for aligning ligands. The DISCO method [40, 79, 91] represents every ligand as a set of feature points divided into “ligand points” and “site points”. The ligand points represent the atom feature of the ligand, which can be charged atoms, hydrogen bond donor/acceptor and hydrophobic properties. The site points represent the atom feature of the receptor which are derived from the ligand. The feature points are paired using the Bron-Kerbosch clique detection algorithm on conformations generated before the alignment takes place. The ligands are kept rigid during the alignment procedure, in which they are aligned pairwise to conformations of the ligand with the fewest different conformations. Every pairing is examined to identify a pharmacophore in which all feature points are shared among all ligands. If such a model is impossible to identify (this might happen when a receptor has several binding pockets) the program will generate pharmacophore models which omit some of the input ligands.

The HipHop module of Catalyst [40,91] is similar to DISCO, in that it uses a set of conformations to derive a common pharmacophore, based on feature points which can represent both ligand and receptor atoms. Unlike DISCO, HipHop treats each ligand as a reference in turn, and instead of finding large pairings using clique detection, pairings are grown by iteratively adding more feature points. Every time a new point is added the pharmacophore model is checked against all the ligands. The method seeks to identify models which match all the features, but it can also be set to return smaller, incomplete models if the requirement is impossible to fulfil.

Another rigid aligner is PHASE from Schrödinger [27]. Pharmacophores are extracted from given conformations and matched using a decision tree based method to group related models. Each pharmacophore in a group is used, in turn, as a reference for the rest of the pharmacophores, and a representation from the group is chosen based on an RMSD minimisation. Ligands are also compared to this representative using a RMSD approach, but it is unclear how points are paired.

A more recent rigid conformation pairwise alignment is the LigandScout method [130]. This method is also dependent on a pairing of feature points and uses distance bins along with the Hungarian worker assignment algorithm [84] to pair points. Unlike DISCO and HipHop, LigandScout does not try to generate a common pharmacophore, but focuses on rapidly aligning pairs of molecules.
such that the method can be used interactively.

GASP [54,91] is a method which also requires a point mapping, but which, unlike the three previous methods, allows the ligands to be flexible while aligning them. Feature points are extracted and used for aligning the molecules using the ligand with the fewest feature points in a Kabsch alignment [57]. The algorithm uses an EA strategy in which the optimisation problem is represented as genes encoding torsion angles, and the mapping between feature points needed for performing a RMSD minimisation. This strategy might give rise to illegal pairings, and many evolved genes are invalid. The method has several objectives, such as maximising the number of paired points and keeping the energy low, which are combined using a weighted sum. A later version of the method (MOGA [33]) uses a multi objective version of the algorithm to simultaneously optimise the goals. A more recent version (GAPE [52]) addresses several issues, among these are the problem with genes representing invalid feature point pairings and a bias toward the ligand with the fewest features.

Another recent point mapping dependent method using EAs for aligning ligands is FLAME [22]. As in GASP, molecules are flexible during alignment, but the pairing is performed using a clique detection algorithm. As in DISCO, the fitness of an alignment is a weighted sum of overlap and energy.

Mapping independent methods have become increasingly popular, but these methods can, of course, not use the RMSD for measuring the overlap of an alignment. Instead, the ROCS method [98] uses Gaussian overlap to measure the alignment of a set of rigid conformations of ligands. Atom centers are used for calculating the overlap, and in the original article, no distinction was made between atom of different type or with different features. A later version (ROCS-color [43]) adds the Gaussian overlap of groups with the same type (hydrogen donor/acceptor, hydrophobe, cation, anion and ring) to the “uncoloured” overlap. The overlap is normalised using Tanimoto normalisation before it is used for ranking the ligands. Ligands are aligned pairwise, by first aligning their moments of inertia, followed by rotations around these. Using conformation generated by OpenEyes OMEGA, the original ROCS study was able to identify four new and significantly different inhibitor scaffolds for ZipA, thereby proving the methods ability to scaffold hop.

There are also some mapping independent flexible alignment methods. One of these is QUASI [116], which evolves a pharmacophore iteratively by repeatedly aligning a set of flexible ligands to a pharmacophore, deriving a new pharmacophore for that alignment and repeating the procedure. In each iteration, ligand points are matched with pharmacophore sites and a least squares alignment is performed. The scoring function takes the RMSD as well as the van der Waals interactions into account. It also penalises complex models to derive a general pharmacophore that is not over-fitted.

Another method which allows flexible torsional angles is pharmACOphore [60]. The implements another nature inspired optimisation strategy, ant colony optimisation [28], to align multiple flexible ligands. An ant in the algorithm encodes ligand rotation, translation and torsional angles and is evaluated using a weighted sum of a similarity, clash and torsional energy term. Similarity is based on an interaction score ranging from 0 to $w$ (attraction) or $-w$ to 0.
(repulsion) for some feature type dependent variable \( w \). The ACO algorithm works by placing an amount of pheromone in the tested points, which will make it more likely that new sampled points will be in that area. Solutions found by the method are in discrete space, e.g. by restricting the torsional angles to 1° resolution, but the solutions returned by the algorithm is subsequently subject to a Nelder-Mead optimisation.

5.3 Research contributions

During my PhD studies I have worked with several aspects of ligand alignment. This section will briefly describe my main research contributions within the field. These cover benchmarks of optimisation methods on molecular alignment; investigations into tailoring methods to work on problems involving angles and orientations; and the effectiveness of a methods for flexible aligning a set of ligands.

5.3.1 Optimal overlay using Differential Evolution

Some of my early work was centered around constructing a multiple flexible aligner that was mapping independent and which took ligand energy into account when aligning ligands. I implemented a method which used Gaussian overlap between atoms of ligands. Atoms were separated into four types, which correspond to those found in ROCS-color, and the overlap was only calculated for atoms of the same type. The types were: hydrogen donor, hydrogen acceptor, both and neither. Energy was calculated using a piece-wise linear potential, which has previously been used in molecular docking studies [114], and clashing atoms in the same ligand were furthermore penalised if they were within 2 Ångström. The overlap and energy were combined using simple subtraction, to form a fitness function.

The ligands orientation, spatial placement and torsional angles were encoded in vectors, much as in the pharmACOphore study, but with the difference that real numbers were used instead of discrete angles. The dimensionality of the encoding depended on the number of ligands and the number of torsional angles in these. To find an optimal vector, an EC method called Differential Evolution (DE) [108] was used as the optimisation method. Differential Evolution maintains a population of individuals, and new generations are formed by recombining existing individuals, based on their vector difference. Figure 5.4 illustrates how a new individual is created from \( p^{(1)} \), \( p^{(2)} \) and \( p^{(3)} \) by adding the weighted vector difference between \( p^{(2)} \) and \( p^{(3)} \) to \( p^{(1)} \). If \( p^{(2)} \) and \( p^{(3)} \) are far apart, the difference between new individuals will be large, and the algorithm will perform an exploratory search. If the difference is small the search will be more exploiting, refining the current solution. The new points fitness is evaluated, and the point replaces an existing point in the population if it has a better fitness. This explanation is of course simplified, and detailed pseudo code can be found in Figure 5.5, and a more detailed presentation can be found in Chapter 12.
Figure 5.4: An illustration of how DE can create a new individual, using four existing points. The new point is generated using the weighted difference between $p^{(2)}$ and $p^{(3)}$, which is added to $p^{(1)}$. The offspring $p$ replaces $p^{(i)}$ if it is more fit.

The main focus of my studies then were to evaluate the performance of the DE algorithm. It was therefore compared to a Nelder-Mead implementation and random sampling. The NM method was restarted every time it converged to a local optimum. All three methods were tested on the same subset of the FlexS data set [74] consisting of known binders from concanavalin, dhfr and fructose targets. The methods were run using standard parameters, and they were all given 750,000 fitness evaluations before being terminated. Their performance was evaluated using the minimum and mean fitness value of the optimal solution over 20 runs. The standard deviation was also calculated to quantify the robustness of the methods.

The conclusion of the study was, that both NM and DE outperformed random sampling, and that DE outperformed NM. DE also had far the lowest standard deviation, indicating that it is far more stable. These results were presented at a poster session at 17th Annual International Conference on Intelligent Systems for Molecular Biology and the 8th European Conference on Computational Biology [69]. A paper presenting the results was presented at the International Joint Conference on Bioinformatics, Systems Biology and Intelligent Computing 2009 [70], and can be found in Chapter 12.

5.3.2 Optimal angles using Differential Evolution

In the study using DE for finding an optimal overlay, the algorithm was used as a black box optimisation method. The problem encoding contained angles, and there is a potential pitfall when using DE to create angle vectors: as angles are periodic, two angles might have a large numerical difference, yet still represent two angles that are very close. For an illustration, please consult Figure 5.6, in which $\theta$ and $\theta'$ are very close, but have an extreme numerical difference. This might hinder the DE algorithm from going into the exploitation phase of an optimisation, which could degrade the quality of the returned result.

I have conducted a study to investigate the effect of two strategies designed for overcoming this problem in DE. The first solution, proposed by Molegro,
5.3. Research contributions

DifferentialEvolution($NP, CR, F$)
1. Initialize $NP$ random individuals $p^1, \ldots, p^{NP}$
2. while termination criterion is not met
   3. do for each individual $p^i$
      4. $p \leftarrow \text{CREATEOFFSPRING}(p^i)$
      5. if $p$ is better than $p^i$, replace $p^i$ with $p$

CREATEOFFSPRING($p^i$)
1. Randomly select parents $p^{(1)}, p^{(2)}$ and $p^{(3)}$
2. Initialize empty offspring $p$
3. Let $j$ be a random number between 1 and $n$.
4. for $k \leftarrow 1, \ldots, n$
5.   do if $\text{RANDOM()} \leq CR$ or $k = j$
6.      then $p_k \leftarrow p^{(1)}_k + F \cdot (p^{(2)}_k - p^{(3)}_k)$
7.      else $p_k \leftarrow p^i_k$
8. return $p$

Figure 5.5: Pseudo code from the DE algorithm with the DE/rand/1/bin strategy.

![Diagram showing angles and their differences](image)

Figure 5.6: Illustration of how angles with a small angular distance can result in a large numerical difference.

simply replaces differences larger than $\pi$ with $\pi$ minus the difference and gives the result a random sign. The second strategy, proposed by me, works in a similar manner but uses a consistent selection of signs; see Chapter 13 for details.

These two strategies, along with the unaltered strategy, were tested on three optimisation problems. The first two were constructed angle problems, in which a virtual folding rule was to be unfolded to cover a set of points. The last problem was an energy minimisation problem taken from Handle et al. [49], in which the energy of the alanine dipeptide was sought minimised using the MMFF94 force field.

The result on the folding rule problems indicated that the consistent sign difference performs slightly better than the two other methods; both when measured using the average number of evaluations per success and the number
of successful runs. However, on the ligand optimisation experiments the best method was the unmodified method, and there was little discrepancy between the methods. The more uneven landscape of an energy minimisation seems to call for a more exploratory approach. The results of the study was presented at Congress on Evolutionary Computation 2009 [62], and the paper from the proceedings can be found in Chapter 13.

Another cause of concern when performing ligand alignment is the choice of orientation representation. In my progress report [61] I presented investigations into this by examining three representations: axis-angle pairs, Euler angles and quaternions. Experiments were performed on an extension of the folding rule problem to 3D, and the results indicates that the quaternion representation is best suited for recombining orientations.

5.3.3 Aligning pharmacophores using DE and CMA-ES

My recent studies have been focused on developing methods for performing multiple alignment of flexible ligands. The goal is to (1) recreate the binding mode of a set of ligands without the aid of a target structure, and to (2) obtain a pharmacophore model suitable for performing a virtual screening. Seven feature point types were used for measuring the RMSD between ligands: hydrogen donor, hydrogen acceptor, both donor and acceptor, aromatic ring centers, aliphatic ring centers, positive and negative charged. Feature points were matched using Bron-Kerbosch and Kabsch for every configuration (unlike GASP, which encodes the pairing in the problem domain). Initial studies indicated that this pairing dependent method was too time consuming, and it was therefore abandoned.

Instead, a mapping independent Gaussian overlap was used for measuring the feature overlap. The steric interactions were also added to yield a measure more similar to ROCS-color. The ligand conformations were represented by vectors similar to those described in Subsection 5.3.1. The expensive energy calculations were abandoned in favour of a simple linear declining steric clash penalty function, which was combined with the overlap measure by using a weighted sum.

Optimal alignments have been sought using DE and the Covariance Matrix Adaption Evolutionary Strategy (CMA-ES) [4], another EC algorithm. The CMA-ES works by maintaining a covariance matrix which is used to draw new populations from. The fitness of the population determines how the covariance matrix is updated for the next generation. The two methods were compared by performing pairwise optimisation on the FlexS set [74], which contains sets of known actives in their native binding mode, superimposed according to their common protein target. By keeping one of the ligands orientation and spatial placement stationary, it is possible to perform an alignment which can be used for measuring the RMSD to the known correct conformation. These RMSDs were measured and compared to those of the pharmACOphore study, which uses the same data set. If the RMSD between a found conformation and its known binding mode was below 2.5 Ångström, the alignment of that particular ligand was marked as a success.
5.4 Summary

Three-dimensional ligand comparison methods are sometimes necessary for capturing similarities not demonstrated by simpler methods. This chapter has presented background concerning alignment based comparison methods, including mapping dependent and independent overlap measures, the difference between rigid and flexible alignments, as well as pairwise and multiple alignments. Different algorithms were presented, some based on a point mapping (Bron-Kerbosch and Kabsch) and some mapping independent (moments of inertia, SA, NM and EAs). Using this terminology, existing methods and my research contribution were described and categorised. My research contributions include investigations into the effectiveness of DE as an alignment tool; a study investigating angels effects on DE optimisation; and recent experiments into recreating the native binding mode of ligands, without the aid of a protein structure.

5.5 Future work

Choice of orientation representation seems to have an impact on the DE algorithm for simple problems. No experiments have been performed on the ligand alignment, and it would be very interesting to see the impact of e.g. switching to quaternion representations. DE is probably not the only EA which is sensitive to angle and orientation problems, and an investigation into this would be interesting. In general, other EA methods than DE and CMA-ES could be investigated in a ligand alignment framework.

Many of these algorithms require an initial population, and in all of my experiments these are initialised to uniformly random. Using low energy conformations as a seed for these algorithms has the potential to speed up the methods significantly. To further speed up the optimisation, the conformations could be aligned using a computational cheap alignment method, such as moments of inertia or rigid alignment.

The developed alignment method presented in this chapter could be used for generating a common pharmacophore, which can be used for performing
virtual screening or identifying the correct binding mode of ligands in a large database. This could illustrate the effectiveness of the method further.

Lastly, the method does not take the multiple binding modes into account. As some target proteins have several active pockets, this shortcoming is a potential problem. Taking these pockets into account could be done by performing a clustering of the known binders before aligning them, using a similarity measure such as fingerprints.
Chapter 6

Conclusions and summary

The research presented in this dissertation has been motivated by the development of new virtual screening algorithms that are either faster than existing methods, or which yield better screening results. My work has resulted in two journal and four conference articles, along with three poster presentations. The main contributions will be repeated in the following paragraphs.

A common molecule representation is binary vectors, which can be used for rapidly calculating the similarity between molecules using the Tanimoto coefficient. Together with Jesper Nielsen I have developed two new data structures for accelerating queries into databases of binary vectors [64, 65]. The first of these, the \(kD\) grid, is a generalisation of two previous methods. The second, the multibit tree, extends the \(kD\) grid and offers significantly better pruning degrees than previously known methods. The data structures have been observed to offer significant speedups over existing methods on standard databases, using standard methods for generating the binary fingerprints.

The Tanimoto coefficient generalises to vectors with arbitrary entry value types. Previously, no methods for accelerating queries into databases of these were known. I have constructed a reduction from Tanimoto queries to distance queries in Euclidian space [63] – a problem for which several data structures exist. I have implemented the \(kd\) tree (for accelerating distance queries), the vp tree and the GNAT (for accelerating general metric queries) and performed experiments to assess if they could be used for accelerating Tanimoto queries, with positive results [71].

Another representation is simple strings of characters, which can be broken into evenly sized fragments called Lingos. These fragment sets can be compared using the LINGOsim similarity measure; a measure for which several efficient calculation methods exist. Together with Jesper Nielsen I developed a method based on inverted indices for rapidly calculating the similarity matrix for a set of Lingos. Experiments on the implementation revealed that it outperforms the current best method, which utilises specialised hardware. A manuscript describing the methods and results has been accepted on the condition of minor revisions.

Molecular graphs can also be used for representing and comparing molecular structures. The comparison is often based on some variation of the largest common sub graph, which measures the global structural similarity between
graphs. I have developed a tree based comparison method using chemical features for identifying common nodes. The method was observed to outperform or supplement existing methods when used as a ranking tool. Together with Anders Johnsen I developed new chemical graph similarity measures based on the distribution of all common sub graphs between two graphs, instead of only the largest. Experiments indicate that using the standard deviation of the clique size distribution can outperform existing methods.

If 3D structures are used for representing molecules, they are often compared by first aligning them and then measure their overlap. I have compared Differential Evolution and Nelder Mead on the problem of aligning small molecules and found that Differential Evolution outperformed Nelder Mead on this subset of problems [70]. This study let me to investigate optimisation problems involving angles in Differential Evolution on a variety of optimisation problems, including minimising the free energy of a molecule system [62]. Finally, I have developed an unpublished method for performing multiple flexible alignment for generating pharmacophore models.
Part II

Papers
Chapter 7

A tree-based method for the rapid screening of chemical fingerprints

The paper *A tree-based method for the rapid screening of chemical fingerprints* presented in this chapter has been published in part as a conference paper [64] and a journal article [65].


The paper was first presented at *Algorithms in Bioinformatics 9th International Workshop* and selected for an extended publication in *Algorithms for Molecular Biology*. The article presented here is the extended version which contains more experiments into the nature of the presented algorithms and methods. Apart from minor typographically changes, this article is identical to the one published in *Algorithms for Molecular Biology*. 

57
A Tree-Based Method for the Rapid Screening of Chemical Fingerprints

Thomas G. Kristensen* Jesper Nielsen†
Christian N. S. Pedersen‡

Abstract

The fingerprint of a molecule is a bitstring based on its structure, constructed such that structurally similar molecules will have similar fingerprints. Molecular fingerprints can be used in an initial phase of drug development for identifying novel drug candidates by screening large databases for molecules with fingerprints similar to a query fingerprint.

In this paper, we present a method which efficiently finds all fingerprints in a database with Tanimoto coefficient to the query fingerprint above a user defined threshold. The method is based on two novel data structures for rapid screening of large databases: the \(kD\) grid and the Multibit tree. The \(kD\) grid is based on splitting the fingerprints into \(k\) shorter bitstrings and utilising these to compute bounds on the similarity of the complete bitstrings. The Multibit tree uses hierarchical clustering and similarity within each cluster to compute similar bounds. We have implemented our method and tested it on a large real-world data set. Our experiments show that our method yields approximately a three-fold speed-up over previous methods.

Using the novel \(kD\) grid and Multibit tree significantly reduce the time needed for searching databases of fingerprints. This will allow researchers to (1) perform more searches than previously possible and (2) to easily search large databases.

7.1 Introduction

When developing novel drugs, researchers are faced with the task of selecting a subset of all commercially available molecules for further experiments. There are more than 8 million such molecules available [48], and it is not feasible to perform computationally expensive calculations on each one. Therefore, the need arises for fast screening methods for identifying the molecules that are most likely to have an effect on a given disease. It is often the case that

---

*Bioinformatics Research Centre (BiRC), Aarhus University, C. F. Møllers Allé 8, DK-8000 Århus C, Denmark. E-mail: tgk@birc.au.dk.
†Bioinformatics Research Centre (BiRC), Aarhus University, C. F. Møllers Allé 8, DK-8000 Århus C, Denmark. E-mail: jn@birc.au.dk.
‡Bioinformatics Research Centre (BiRC), Aarhus University, C. F. Møllers Allé 8, DK-8000 Århus C, Denmark. E-mail: cstorm@birc.au.dk.
a molecule with some effect is already known, e.g. from an already existing
drug. An obvious initial screening method presents itself, namely to identify
the molecules which are similar to this known molecule. To implement this
screening method one must decide on a representation of the molecules and
a similarity measure between representations of molecules. Several represent-
ations and similarity measures have been proposed [35, 73, 127]. We focus on
molecular fingerprints. A fingerprint for a given molecule is a bitstring of size $N$
which summarises structural information about the molecule [73]. Fingerprints
should be constructed such that if two fingerprints are very similar, so are the
molecules which they represent. There are several ways of measuring the sim-
ilarity between fingerprints [127]. We focus on the Tanimoto coefficient, which
is a normalised measure of how many bits two fingerprints share. It is 1.0 when
the fingerprints are the same, and strictly smaller than 1.0 when they are not.
Molecular fingerprints in combination with the Tanimoto coefficient have been
used successfully in previous studies [129].

We focus on the screening problem of finding all fingerprints in a database
with Tanimoto coefficient to a query fingerprint above a given threshold, e.g. 0.9.
Previous attempts have been made to improve the query time. One approach
is to reduce the number of fingerprints in the database for which the Tanimoto
coefficient to the query fingerprint has to be computed explicitly. This includes
storing the fingerprints in the database in a vector of bins [110], or in a trie
like structure [103], such that searching certain bins, or parts of the trie, can be
avoided based on an upper-bound on the Tanimoto coefficient between the query
fingerprint and all fingerprints in individual bins or subtries. Another approach
is to store an XOR summary, i.e. a shorter bitstring, of each fingerprint in
the database, and use these as rough upper bounds on the maximal Tanimoto
coefficients achievable, before calculating the exact coefficients [7].

In this paper, we present an efficient method for the screening problem,
which is based on an extension of an upper bound given in [110] and two novel
tree based data structures for storing and retrieving fingerprints. To further
reduce the query time we also utilise the XOR summary strategy [7]. We have
implemented our method and tested it on a realistic data set. Our experiments
clearly demonstrate that it is superior to previous strategies, as it yields a
three-fold speed-up over the previous best method.

7.2 Methods

A fingerprint is a bitstring of length $N$. Let $A$ and $B$ be bitstrings, and let $|A|$ denote the number of 1-bits in $A$. Let $A \land B$ denote the logical and of $A$ and $B$, that is, $A \land B$ is the bitstring that has 1-bits in exactly those positions where both $A$ and $B$ do. Likewise, let $A \lor B$ denote the logical or of $A$ and $B$, that is, $A \lor B$ is the bitstring that has 1-bits in exactly those positions where either $A$ or $B$ do. With this notation the Tanimoto coefficient becomes:

$$S_T(A, B) = \frac{|A \land B|}{|A \lor B|}$$

Figure 7.1 shows an example the usage of this notation. In the following,
7.2. Methods

A $\begin{array}{cc} 1 & 0 \\ 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{array}$ | $|A| = 4$
B $\begin{array}{cc} 1 & 1 \\ 0 & 1 \\ 0 & 0 \end{array}$ | $|B| = 3$

\[
A \land B = \begin{array}{cc} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{array} \quad |A \land B| = 2
\]
\[
A \lor B = \begin{array}{cc} 1 & 1 \\ 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{array} \quad |A \lor B| = 5
\]

\[S_T(A, B) = \frac{2}{5}\]

Figure 7.1: Example of calculation of the Tanimoto coefficient $S_T(A, B)$, where $A = 101101$ and $B = 110100$.

we present a method for finding all fingerprints $B$ in a database of fingerprints with a Tanimoto coefficient above some query-specific threshold $S_{\text{min}}$ to a query fingerprint $A$. The method is based on two novel data structures, the $kD$ grid and the Multibit tree, for storing the database of fingerprints.

7.2.1 $kD$ grid

Swamidass et al. showed in [110] that if $|A|$ and $|B|$ are known, $S_T(A, B)$ can be upper-bounded by

\[S_{\text{max}} = \min(|A|, |B|) / \max(|A|, |B|).\]

This bound can be used to speed up the search, by storing the database of fingerprints in $N+1$ buckets such that bitstring $B$ is stored in the $|B|$th bucket. When searching for bitstrings similar to a query bitstring $A$ it is sufficient to examine the buckets where $S_{\text{max}} \geq S_{\text{min}}$.

We have generalised this strategy. Select a number of dimensions $k$ and split the bitstrings into $k$ equally sized fragments such that

\[
A = A_1 \cdot A_2 \cdot \ldots \cdot A_k
\]
\[
B = B_1 \cdot B_2 \cdot \ldots \cdot B_k,
\]

where $X \cdot Y$ is the concatenation of bitstrings $X$ and $Y$.

The values $|A_1|, |A_2|, \ldots, |A_k|$ and $|B_1|, |B_2|, \ldots, |B_k|$ can be used to obtain a tighter bound than $S_{\text{max}}$. Let $N_i$ be the length of $A_i$ and $B_i$. The $kD$ grid is a $k$-dimensional cube of size $(N_1 + 1) \times (N_2 + 1) \times \ldots \times (N_k + 1)$. Each grid point is a bucket and the fingerprint $B$ is stored in the bucket at coordinates $(n_1, n_2, \ldots, n_k)$, where $n_i = |B_i|$. An example of such a grid is illustrated in Figure 7.2. By comparing the partial coordinates $(n_1, n_2, \ldots, n_i)$ of a given bucket to $|A_1|, |A_2|, \ldots, |A_i|$, where $i \leq k$, it is possible to upper-bound the Tanimoto coefficient between $A$ and every $B$ in that bucket. By looking at the partial coordinates $(n_1, n_2, \ldots, n_{i-1})$, we can use this to quickly identify those partial coordinates $(n_1, n_2, \ldots, n_i)$ that may contain fingerprints $B$ with a Tanimoto coefficient above $S_{\text{min}}$.

Assume the algorithm is visiting a partial coordinate at level $i$ in the data structure. The indices $n_1, n_2, \ldots, n_{i-1}$ are known, but we need to compute which
Figure 7.2: Example of a kD-grid with \( k = 3 \). \( B \) is split into smaller substrings and the count of 1-bits in each determines where in \( B \) is placed in the grid. The small inner cube shows the placement of \( B \).

\[ n_i \] to visit at this level. The entries to be visited further down the data structure \( n_{i+1}, \ldots, n_k \) are, of course, unknown at this point. A bound can be calculated in the following manner.

\[
\begin{align*}
S_T(A, B) &= \frac{|A \land B|}{|A \lor B|} \\
&= \frac{\sum_{j=1}^{k} |A_j \land B_j|}{\sum_{j=1}^{k} |A_j \lor B_j|} \\
&\leq \frac{\sum_{j=1}^{k} \min(|A_j|, n_j)}{\sum_{j=1}^{k} \max(|A_j|, n_j)} \\
&= \frac{\sum_{j=1}^{i-1} \min(|A_j|, n_j) + \min(|A_i|, n_i) + \sum_{j=i+1}^{k} \min(|A_j|, n_j)}{\sum_{j=1}^{i-1} \max(|A_j|, n_j) + \max(|A_i|, n_i) + \sum_{j=i+1}^{k} \max(|A_j|, n_j)} \\
&\leq \frac{\sum_{j=1}^{i-1} \min(|A_j|, n_j) + \min(|A_i|, n_i) + \sum_{j=i+1}^{k} |A_j|}{\sum_{j=1}^{i-1} \max(|A_j|, n_j) + \max(|A_i|, n_i) + \sum_{j=i+1}^{k} |A_j|} \\
&= S_{\text{grid}}^\text{max}
\end{align*}
\]
7.2. Methods

\[ n_i = 1 \cdot 0 \cdot 1 \cdot 2 \]

\[ n_i = 2 \cdot 0 \cdot 1 \cdot 2 \]

\[ n_i = 3 \cdot 0 \cdot 1 \cdot 2 \]

\[ n_i = 4 \cdot 0 \cdot 1 \cdot 2 \]

\[ \forall i \]

\[ B = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \]

\[ B' = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \]

\[ B'' = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \]

\[ B''' = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \]

\[ i = 1 \]

\[ i = 2 \]

\[ i = 3 \]

\[ i = 4 \]

Figure 7.3: Example of a 4D grid containing four bitstrings, stored as in our implementation. The dotted lines indicate the splits between \( B_i \) and \( B_{i+1} \).

The \( n_i \)'s to visit lie in an interval and it is thus sufficient to compute the upper and lower indices of this interval, \( n_u \) and \( n_l \) respectively. Setting \( S_{\text{min}} = S_{\text{grid}} \) isolating \( n_i \) and ensuring that the result is an integer in the range 0...\( N_i \) gives:

\[ n_l = \max \left( \left\lceil S_{\text{min}} (A_{\text{max}}^i + |A_i| + A_i^{\mid \mid}) - (A_{\text{min}}^i + A_i^{\mid \mid}) \right\rceil, 0 \right) \]

and

\[ n_u = \min \left( \left\lfloor A_{\text{min}}^i + |A_i| + A_i^{\mid \mid} - S_{\text{min}} (A_{\text{max}}^i + A_i^{\mid \mid}) \right\rfloor, N_i \right) \]

where \( A_{\text{min}}^i = \sum_{j=1}^{i-1} \min(|A_j|, n_j) \) is a bound on the number of 1-bits in the logical and in the first part of the bitstrings. \( A_{\text{max}}^i = \sum_{j=1}^{i-1} \max(|A_j|, n_j) \) is a bound for the logical or in the first part of the bitstrings. Similarly, \( A_i^{\mid \mid} = \sum_{j=i+1}^k |A_j| \) is a bound on the last part.

Note that in the case where \( k = 1 \) this datastructure simply becomes the list presented by Swamidass et al. [110], and in the case where \( k = N \) the datastructure becomes the binary trie presented by Smellie [103].

We have implemented the \( k \)D grid as a list of lists, where any list containing no fingerprints is omitted. See Figure 7.3 for an example of a 4D grid containing four bitstrings. The fingerprints stored in a single bucket in the \( k \)D grid can be organised in a number of ways. The most naive approach is to store them in a simple list which has to be searched linearly. We propose to store them in tree structures as explained below.

7.2.2 Singlebit tree

The Singlebit tree is a binary tree which stores the fingerprints of a single bucket from a \( k \)D grid. At each node in the tree a position in the bitstring
is chosen. All fingerprints with a zero at that position are stored in the left subtree while all those with a one are stored in the right subtree. This division is continued recursively until all the fingerprints in a given node are the same. When searching for a query bitstring $A$ in the tree it now becomes possible, by comparing $A$ to the path from the root of the tree to a given node, to compute an upper bound $S_{\text{single}}$ on $S_T(A, B)$ for every fingerprint $B$ in the subtree of that given node.

Given two bitstring $A$ and $B$ let $M_{ij}$ be the number of positions where $A$ has an $i$ and $B$ has a $j$. There are four possible combinations of $i$ and $j$, namely $M_{00}$, $M_{01}$, $M_{10}$ and $M_{11}$.

The path from the root of a tree to a node defines lower limits $m_{ij}$ on $M_{ij}$ for every fingerprint in the subtree of that node. Let $u_{ij}$ denote the unknown difference between $M_{ij}$ and $m_{ij}$, that is $u_{ij} = M_{ij} - m_{ij}$. Remember that $|B| = \sum_{i=1}^{k} n_k$ is known when processing a given bucket.

By using

\[
\begin{align*}
    u_{10} + u_{11} & = |A| - m_{10} - m_{11} \\
    u_{01} + u_{11} & = |B| - m_{01} - m_{11} \\
    u_{11} & \leq \min(u_{01} + u_{11}, u_{10} + u_{11}) \\
    u_{01} + u_{10} + u_{11} & \geq \max(u_{01} + u_{11}, u_{10} + u_{11})
\end{align*}
\]

an upper bound on the Tanimoto coefficient of any fingerprint $B$ in the subtree can then be calculated as

\[
S_{T}(A, B) = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}
\]

\[
\leq \frac{m_{11} + \min(u_{01} + u_{11}, u_{10} + u_{11})}{m_{01} + m_{10} + m_{11} + \max(u_{01} + u_{11}, u_{10} + u_{11})}
\]

\[
= \frac{\min(|A| - m_{10}, |B| - m_{01})}{m_{01} + m_{10} + \max(|A| - m_{10}, |B| - m_{01})}
\]

\[
= S_{\text{single}}^{\text{max}}.
\]

When building the tree data structure it is not immediately obvious how best to choose which bit positions to split the data on, at a given node. The implemented approach is to go through all the children of the node and choose the bit which best splits them into two parts of equal size, in the hope that this creates a well-balanced tree. It should be noted that the tree structure that gives the best search time is not necessarily a well-balanced tree. Figure 7.4 shows an example of a Singlebit tree.

The Singlebit tree can also be used to store all the fingerprints in the database without a kD grid. In this case, however, $|B|$ is no longer available and thus the $S_{\text{single}}^{\text{max}}$ bound cannot be used. A less tight bound can be formulated, but experiments, not included in this paper, indicate that this is a poor strategy.
Figure 7.4: Example of a Singlebit tree. The black squares mark the bits chosen for the given node, while the grey squares mark bits chosen at an ancestor. The grey triangles represent subtrees omitted to keep this example simple. Assume we are searching for the bitstring $A$ in the example. When examining the node marked by the arrow we have the knowledge shown in $B^?$ about all children of that node. Comparing $A$ against $B^?$ gives us $m_{00} = 0$, $m_{01} = 0$, $m_{10} = 1$ and $m_{11} = 2$. Thus $S_{\text{single}} = \frac{4}{5}$. Indeed we find that $S_T(A, B) = \frac{3}{7}$ and $S_T(A, B') = \frac{4}{6}$.

### 7.2.3 Multibit tree

The experiments illustrated in Figure 7.7 in Section 7.3 unfortunately show that using the $kD$ grid combined with Singlebit trees decreases performance compared to using the $kD$ grid and simple lists. The fingerprints used in our experiments have a length of 1024 bits. In our experiments no Singlebit tree was observed to contain more than 40,000 fingerprints (see Figure 7.6). This implies that the expected height of the Singlebit trees is no more than 15 (as we aim for balanced trees cf. above). Consequently, the algorithm will only obtain information about 15 out of 1024 bits before reaching the fingerprints. A strategy for obtaining more information is to store a list of bit positions, along with an annotation of whether each bit is zero or one, in each node. The bits in this list are called the match-bits.

The Multibit tree is an extension of the Singlebit tree, where we no longer demand that all children of a given node are split according to the value of a single bit. In fact we only demand that the data is arranged in some binary tree. The match-bits of a given node are computed as all bits that are not a match-bit in any ancestor and for which all fingerprints in the leaves of the node have the same value. Note that a node could easily have no match-bits. When searching through the Multibit tree, the query bitstring $A$ is compared
Figure 7.5: An example of a Multibit tree. The black squares marks the match-bits and their annotation. Grey squares show bits that were match-bits at an ancestor. Grey triangles are subtrees omitted to keep this example simple. When visiting the node marked by the arrow we get \( m_{00} = 1, m_{01} = 1, m_{10} = 1 \) and \( m_{11} = 2 \), thus \( S_{\text{multi}} = \frac{4}{6} \). Still \( S_T(A,B) = \frac{3}{7} \) and \( S_T(A,B') = \frac{4}{6} \).

to the match-bits of each visited node and \( m_{00}, m_{01}, m_{10} \) and \( m_{11} \) are updated accordingly. \( S_{\text{multi}} \) is computed the same way as \( S_{\text{single}} \) and only branches for which \( S_{\text{multi}} \geq S_{\text{min}} \) are visited.

Again, the best way to build the tree is not obvious. Currently, the same method as for the Singlebit trees is used. For a node with a given set of fingerprints, choose the bit which has a 1-bit in, as close as possible to, half of the fingerprints. Split the fingerprints into two sets, based on the state of the chosen bit in each fingerprint. Continue recursively in the two children of the node. Figure 7.5 shows an example of a Multibit tree. To reduce the memory consumption of the inner nodes, the splitting is stopped and leaves created, for any node that has less than some limit \( l \) children. Based on initial experiments, not included in this paper, \( l \) is chosen as 6, which reduces memory consumption by more than a factor of two and has no significant impact on speed. An obvious alternative way to build the tree would be to base it on some hierarchical clustering method, such as Neighbour Joining [100].

7.3 Experiments

We have implemented the \( kD \) grid and the Single- and Multibit tree in Java. The implementation along with all test data is available at

http://www.birc.au.dk/~tgk/TanimotoQuery/.
7.4. Results

Using these implementations, we have constructed several search methods corresponding to the different combinations of the data structures. We have examined the 4D grid for $k = 1, 2, 3$ and $4$, where the fingerprints in the buckets are stored in a simple list, a Singlebit tree or a Multibit tree. For purposes of comparison, we have implemented a linear search strategy, that simply examines all fingerprints in the database. We have also implemented the strategy of “pruning using the bit-bound approach first, followed by pruning using the difference of the number of 1-bits in the XOR-compressed vectors, followed by pruning using the XOR approach” from [7]. This strategy will hereafter simply be known as Baldi. A trick of comparing the XOR-folded bitstrings [7] immediately before computing the true Tanimoto coefficient, is used in all our strategies to improve performance. The length of the XOR summary is set to 128, as suggested in [7]. An experiment, not included in this paper, confirmed that this is indeed the optimal size of the XOR fingerprint. We have chosen to reimplement related methods in order to make an unbiased comparison of the running times independent of programming language differences.

The methods are tested on a real-world data set by downloading version 8 of the ZINC database [48], consisting of roughly 8.5 million commercially available molecules. Note that only 2 million of the molecules have actually been used, due to memory constraints. The distribution of one-bits is presented in Figure 7.6, where it can be seen there are many buckets in the 1D grid that will be empty.

The experiments were performed on an Intel Core 2 Duo running at 2.5GHz and with 2GB of RAM. Fingerprints were generated using the CDK fingerprint generator [105] which has a standard fingerprint size $N$ of 1024. One molecule timed out and did not generate a fingerprint. We have performed our tests on different sizes of the data set, from 100,000 to 2,000,000 fingerprints in 100,000 increments. For each data set size, the entire data structure is created. Next, the first 100 fingerprints in the database are used for queries. We measure the query time and the space consumption.

Figure 7.6: Distribution of the number of bits set in the 1024 bit CDK fingerprints from the ZINC database.
Figure 7.7: Different strategies tested with $k = 1, \ldots, 4$. Each experiment is performed 100 times, and the average query time is presented. All experiments are performed with a $S_{\text{min}}$ of 0.9. The three graphs (a) – (c) show the performance of the three bucket types for the different values of $k$. The best $k$ for each method is presented in graph (d) along with the simple linear search results and Baldi.

### 7.4 Results

Figure 7.7 shows the average query time for the different strategies and different values of $k$ plotted against the database size. We note that the Multibit tree in a 1D grid is best for all sizes. Surprisingly, the simple list, for an appropriately high value of $k$, is faster than the Singlebit tree, yet slower than the Multibit tree. This is probably due to the fact that the Singlebit trees are too small to contain sufficient information for an efficient pruning: the entire tree is traversed, which is slower than traversing the corresponding list implementation. All three approaches (List, Singlebit- and Multibit trees) are clearly superior to the Baldi approach, which in turn is better than a simple linear search (with the XOR folding trick).
7.4. Results

Figure 7.8: Experiments with simple lists for $k = 1, \ldots, 10$. Each test is performed 100 times, and the average query time is presented. All experiments are performed with a $S_{\text{min}}$ of 0.9. Missing data points are from runs with insufficient memory.

From Figure 7.7a we notice that the List strategy seems to become faster for increasing $k$. This trend is further investigated in Figure 7.8, which indicate that a $k$ of three or four seems optimal. As $k$ grows the grid becomes larger and more time consuming to traverse while the lists in the buckets become shorter. For sufficiently large values of $k$, the time spent pruning buckets exceeds the time visiting buckets containing superfluous fingerprints. The Singlebit tree data in Figure 7.7b indicates that the optimal value of $k$ is three. It seems the trees become too small to contain enough information for an efficient pruning, when $k$ reaches four. In Figure 7.7 we see the Multibit tree. Again, a too large $k$ will actually slow down the data structure. This can be explained with arguments similar to those for the Singlebit tree. Surprisingly, it seems a $k$ as low as one is optimal.

Figure 7.9 shows the memory usage per fingerprint as a function of the number of loaded fingerprints. The first thing we note is that the Multibit tree uses significantly more memory than the other strategies. This is due to the need to store a variable number of match-bits in each node. The second thing to note is the space usage for different $k$'s. In the worst case, where all buckets contain fingerprints, the memory consumption per fingerprint, for the grid alone, becomes $O\left(\frac{1}{n} \left(\frac{N}{k}\right)^k\right)$, where $n$ is the number of fingerprints in the database. Thus we are not surprised by our actual results.

Figure 7.10 shows the search time as a function of the Tanimoto threshold. In general we note that the simpler and more naive data structures performs better for a low Tanimoto threshold. This is due to the fact that, for a low
Chapter 7. A tree-based method for screening chemical fingerprints

<table>
<thead>
<tr>
<th>$k$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>line type</td>
<td></td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

List | Singlebit tree | Multibit tree

(a) ![Graph (a)](image)

(b) ![Graph (b)](image)

(c) ![Graph (c)](image)

(d) ![Graph (d)](image)

Figure 7.9: The memory consumption of the data structure for different strategies tested with $k = 1, \ldots, 4$. The three graphs (a) – (c) show the performance of the three bucket types for the different values of $k$. The $k$ yielding the fastest query time for each method is presented in graph (d) along with the simple linear search results and Baldi.

Tanimoto threshold a large part of the entire database will be returned. In these cases very little pruning can be done, and it is faster to run through a simple list than to traverse a tree and compare bits at each node. Of course we should remember that we are interested in performing searches for similar molecules, which means large Tanimoto thresholds.

The reason why linear search is not constant time for a constant data set is that, while it will always visit all fingerprints, the time for visiting a given fingerprint is not constant due to the XOR folding trick.

The running times of the different methods depend on the number of Tanimoto coefficients between pairs of bitstrings that must be calculated explicitly. This number depends on the method and not on the programming language in which the method is implemented, and is thus an implementation independent
7.5. Conclusion

In this paper we have presented a method for finding all fingerprints in a database with Tanimoto coefficient to a query fingerprint above a user defined performance measure. Figure 7.11 presents the fraction of coefficient calculated for varying number of fingerprints and a Tanimoto threshold of 0.9. Each method seems to calculate a fairly constant fraction of the fingerprints: only the Multibit tree seems to vary with the number of fingerprints. This is most likely due to the fact that more fingerprints result in larger trees with more information.

The result is consistent with the execution time experiments: the methods have the same relative ranking when measuring the fraction of coefficients calculated as when measuring the average query time in Figure 7.7.

The fraction of coefficients calculated has also been measured for varying Tanimoto thresholds with 2,000,000 fingerprints. The result is presented in Figure 7.12. It seems that the relation between the methods is consistent across Tanimoto thresholds. Surprisingly, the Multibit tree seems to reduce the fraction of fingerprints for which the Tanimoto threshold has to be calculated even for small values of the Tanimoto threshold: the three other methods seem to perform very similar up till a threshold of 0.8, whereas the Multibit tree seems to differentiate itself at a threshold as low as 0.2.

The results seems to be consistent with the average query time presented in Figure 7.10.

7.5 Conclusion

In this paper we have presented a method for finding all fingerprints in a database with Tanimoto coefficient to a query fingerprint above a user defined
Chapter 7. A tree-based method for screening chemical fingerprints

Our method is based on a generalisation of the bounds developed in [110] to multiple dimensions. Our generalisation results in a tighter bound, and experiments indicate that this results in a performance increase. Furthermore, we have examined the possibility of utilising trees as secondary data structures in the buckets. Again, our experiments clearly demonstrate that this leads to a significant performance increase.

Our methods allow researchers to search larger databases faster than previously possible. The use of larger databases should increase the likelihood of finding relevant matches. The faster query times decreases the effort and time needed to do a search. This allow more searches to be done, either for more molecules or with different thresholds \( S_{\text{min}} \) on the Tanimoto coefficient. Both of these features increase the usefulness of fingerprint based searches for the researcher in the laboratory.

Our method is currently limited by the rather larger memory consumption of the Multibit tree. Another implementation might remedy this situation somewhat. Otherwise we suggest an I/O efficient implementation where the tree is kept on disk.

To increase the speed of our method further we are aware of two approaches. Firstly, the best way to construct the Multibit trees remain uninvestigated. Secondly, a tighter coupling between the Multibit tree and the \( kD \) grid would
Figure 7.12: The fraction of the database for which the Tanimoto coefficient is calculated explicitly, measured for a varying Tanimoto threshold and 2,000,000 fingerprints.

allow us to use grid information in the Multibit tree: in the $kD$ grid we have information about each fragment of the fingerprints which is not used in the current tree bounds.
Chapter 8

Transforming Tanimoto queries on real valued vectors to range queries in Euclidian space

The article *Transforming Tanimoto queries on real valued vectors to range queries in Euclidian space* presented in this chapter has been published as a journal article [63].


This article was published in *Journal of Mathematical Chemistry*. The title is slightly misleading at the transformation is into distance queries, not range queries. It has not been changed in the presentation here. Apart from minor typographically changes, this article is identical to the one published in the journal.
Transforming Tanimoto queries on real valued vectors to range queries in Euclidian space

Thomas G. Kristensen*

Vectors are often used for representing molecular structures when searching for chemical compounds with similar properties. The entries of these vectors can be binary, in which case they are referred to as fingerprints, or real valued, in which case they are referred to as descriptors. A diverse set of similarity measures are available for dealing with these vectors [129]. This note focuses on the Tanimoto coefficient, which is applicable to fingerprints as well as descriptors.

Previous studies have focused on decreasing the query time into databases of fingerprint vectors when the Tanimoto coefficient is used [7, 65, 110]. In this note we focus on decreasing the time for Tanimoto queries into databases of real valued descriptors.

The Tanimoto coefficient $T(A, B)$ between two vectors $A, B \in \mathbb{R}^n$ is calculated as

$$T(A, B) = \frac{AB}{||A||^2 + ||B||^2 - AB}$$

A Tanimoto query consists of a target vector $A$ and a minimum coefficient $t$. The result of a Tanimoto query is the set of vectors $B$ in a database for which $T(A, B) \geq t$.

If $A$ and $B$ are binary, that is if their entries take on values either zero or some entry specific value, $T(A, B)$ will lie in the interval $[0, 1]$. In that case it has been proven that the Tanimoto distance, defined as $1 - T$, is a metric [77, 104]. This means that the triangle inequality holds and standard data structures, such as $\mu$-, vp-, M- and GNAT-trees [15, 23, 46, 132], can therefore be used for accelerating Tanimoto queries.

If the entries of the vectors are allowed to take on arbitrary values, the codomain of $T$ extends to $[-\frac{1}{3}, 1]$ [129] and $1 - T$ ceases to be a metric [77]. We can therefore no longer apply standard data structures for accelerating queries. However, it is possible to convert the query into that of a range query in Euclidian space, probably the best known metric. This not only allows the use of data structures based on the triangle inequality, but it also enables the use of data structures tailored for Euclidian space, such as the kD-tree [12], even for binary valued vectors.

The reduction is as follows: Let $A, B \in \mathbb{R}^n$ and $t \in [0, 1]$. Then $T(A, B) \geq t$ iff

$$||t + \frac{1}{2t} A - B|| \leq \frac{\sqrt{-4t^2 + (t + 1)^2}}{2t} ||A||$$

*Bioinformatics Research Centre (BiRC), Aarhus University, C. F. Møllers Allé 8, DK-8000 Århus C, Denmark. E-mail: tgk@birc.au.dk.
Chapter 8. Transforming Tanimoto queries to range queries

Figure 8.1: An example in the plane. All the vectors inside the circle labelled 0.9 have a Tanimoto coefficient larger than 0.9 to A. All vectors outside the circle labelled −0.3 have a Tanimoto coefficient larger than −0.3 to A.

The proof is by simple rewriting; be aware that we use the fact that $||A||^2 + ||B||^2 - AB \geq 0$.

\[
\begin{align*}
||t + 1\frac{1}{2t} A - B|| &\leq \frac{\sqrt{-4t^2 + (t + 1)^2}}{2t} ||A|| \\
\frac{(t + 1)^2}{4t^2} ||A||^2 + ||B||^2 - 2t + 1\frac{1}{2t} AB &\leq \frac{-4t^2 + (t + 1)^2}{4t^2} ||A||^2 \\
||A||^2 + ||B||^2 &\leq \frac{t + 1}{t} AB \\
t(||A||^2 + ||B||^2) &\leq tAB + AB \\
t &\leq \frac{AB}{||A||^2 + ||B||^2 - AB}
\end{align*}
\]

The previous result only covers the case in which $t \in [0,1]$. If $t \in [-\frac{1}{3}; 0]$ then $T(A, B) \geq t$ iff

\[
||t + 1\frac{1}{2t} A - B|| \geq \frac{\sqrt{-4t^2 + (t + 1)^2}}{2t} ||A||
\]

The proof is analogous to that on the interval $[0; 1]$. 8.1 illustrates both relations in the plane.

The results are applicable when all vectors with a coefficient above some threshold $t$ are sought. If instead the $n$ nearest neighbours are sought, decreasing values of $t$ can be used until the number of vectors reaches $n$. 
Acknowledgement

The author thanks Pierre Baldi for comments concerning the manuscript.
Chapter 9

Data structures for accelerating Tanimoto queries on real valued vectors

The article *Data structures for accelerating Tanimoto queries on real valued vectors* presented in this chapter has been published as a conference paper [71].


This article was presented at *Algorithms in Bioinformatics, 2010*. Apart from minor typographically changes, this article is identical to the one presented.
9.1 Introduction

When developing novel drugs, researchers are faced with the task of selecting a subset of all commercially available molecules for further experiments. There are more than 8 million available molecules [48], and it is therefore not possible to perform computationally expensive calculations on each one. The need therefore arise for fast screening methods for identifying the molecules that are most likely to have an effect on a disease or illness. It is often the case that a molecule with some effect is already known, e.g. from an already existing drug. An obvious initial screening method presents itself, namely to identify the molecules which are similar to this known molecule. To implement this screening method one must decide on a representation of the molecules and a similarity measure between representations of molecules. Several representations and similarity measures have been proposed [35,73,127]. This study focuses on real valued molecular descriptors.

Vectors are often used for representing molecular structures when searching for chemical compounds with similar properties. The entries of these vectors can be binary, in which case they are referred to as fingerprints, or real valued, in which case they are referred to as descriptors. A diverse set of similarity measures are available for dealing with these vectors [129]. This study focuses on the Tanimoto coefficient, which is applicable to fingerprints as well as descriptors.

Data structures for accelerating Tanimoto queries on real valued vectors

Thomas G. Kristensen*    Christian N. S. Pedersen†

Abstract

Previous methods for accelerating Tanimoto queries have been based on using bit strings for representing molecules. No work has gone into examining accelerating Tanimoto queries on real valued descriptors, even though these offer a much more fine grained measure of similarity between molecules. This study utilises a recently discovered reduction from Tanimoto queries to distance queries in Euclidean space to accelerate Tanimoto queries using standard metric data structures. The presented experiments show that it is possible to gain a significant speedup and that general metric data structures are better suited than a data structure tailored for Euclidean space on vectors generated from molecular data.
Chapter 9. Data structures for accelerating queries on real valued vectors

The Tanimoto coefficient $T(A, B)$ between two vectors $A, B \in \mathbb{R}^n$ is calculated as

$$T(A, B) = \frac{AB}{||A||^2 + ||B||^2 - AB}.$$ 

A Tanimoto query consists of a target vector $A$ and a minimum coefficient $t$. The result of a Tanimoto query are the vectors $B$ in a database for which $T(A, B) \geq t$.

If $A$ and $B$ are binary, that is if their entries take on values either zero or some entry specific value, $T(A, B)$ will lie in the interval $[0, 1]$. In that case it has been proven that the Tanimoto distance, defined as $1 - T$, is a metric $[77, 104]$. This means that the triangle inequality holds and standard data structures, such as $\mu$-, vp-, M- and GNAT-trees $[15, 23, 46, 132]$, can be used for accelerating Tanimoto queries.

If the entries of the vectors are allowed to take on arbitrary values, $[129]$ states that the codomain of $T$ extends to $[-\frac{1}{3}, 1]$, and $1 - T$ ceases to be a metric $[77]$. The metric data structures for dealing with fingerprints are therefore no longer applicable. However, it is possible to convert the query into that of a distance query in Euclidean space, probably the best known metric. This not only allows the use of data structures based on the triangle inequality, but it also enables the use of data structures tailored for Euclidean space, such as the $kd$-tree $[12]$, even for binary valued vectors.

Previous studies have focused on decreasing the query time of Tanimoto queries into databases of fingerprint vectors $[7, 65, 110]$. This article focusses on speeding up Tanimoto queries into databases of real valued descriptors. These descriptors are able to contain more fine grained information such as molecular weight. Results from fingerprints can not be used on real valued descriptors, as those techniques were tailored for binary valued vectors.

9.2 Data structures

A distance query consists of a query point $q$ and a query radius $q_r$. When performed on a set of points $P$, a distance query should return all the points $p \in P$ for which $||q - p|| \leq q_r$. A Tanimoto query can be transformed into a distance query using the result from $[63]$ which states that if $A, B \in \mathbb{R}^n$ and $t \in [0, 1]$. Then $T(A, B) \geq t$ if and only if

$$|| \frac{t + 1}{2t} A - B || \leq \sqrt{-4t^2 + (t + 1)^2} ||A||.$$ 

Furthermore, if $t \in [-\frac{1}{3}, 0]$ then $T(A, B) \geq t$ if and only if

$$|| \frac{t + 1}{2t} A - B || \geq \sqrt{-4t^2 + (t + 1)^2} ||A||.$$ 

Figure 9.1 illustrates both relations in the plane.

Distance queries can be accelerated using a variety of data structures. This study examine three of these, namely $kd$-trees, vp-trees and GNATs.
9.2. Data structures

Figure 9.1: An example in the plane. All the vectors inside the circle labelled 0.9 have a Tanimoto coefficient larger than 0.9 to $A$. All vectors outside the circle labelled $-0.3$ have a Tanimoto coefficient larger than $-0.3$ to $A$.

Figure 9.2: Illustration of the three different data structures using the same five points. The $kd$-tree first splits on the $x$-axis and thereafter on the $y$-axis. The $vp$-tree has $p_1$ as its first vantage point. The GNAT has two reference points, $p_1$ and $p_4$.

A $kd$-tree is constructed by recursively dividing a set of points based on one of their entries [12]. On the $i$th level of the $kd$-tree the $(i \mod k)$th entry of the points is used to construct a hyper plane dividing the points into two equally sized parts. Figure 9.2a illustrates this for a set of five points in two dimensional space. $kd$-trees are used for searching for points, retrieving nearest neighbours or, as in this case, performing distance queries. A distance query can discard parts of the tree if the search radius of the distance query does not overlap with the split plane defined by the nodes in the tree. $kd$-trees are memory efficient, easy to construct and easy to implement. However, some studies indicate that they have poor performance on high dimensional data [122].

$vp$-trees are based on vantage points, stored in the inner nodes of the
Chapter 9. Data structures for accelerating queries on real valued vectors

trees [132]. Each node has two subtrees: one for the points closest to the nodes vantage point, and one for those further away. Figure 9.2b illustrates a vp-tree on a set of points, with \( p_1 \) as the vantage point. vp-trees are constructed by recursively selecting vantage points and splitting the points into two subsets according to their distance from the vantage points. Each node stores the vantage point \( p \) along with the vantage point radius \( p_r \) within which all the closest points are located. A distance query in a vp-tree is performed by traversing the tree, pruning away subtrees when it can be proven that the points in the subtree cannot possibly fall within the query radius. This is done by calculating the distance from the vantage point \( p \) to the query point \( q \) and using this along with the query radius \( q_r \) and the vantage point radius \( p_r \). If \( ||p - q|| + q_r < p_r \) it is the case that the query area falls fully within the closest points and it is therefore possible to exclude the subtree representing points far away from any further search. Likewise, if \( ||p - q|| - q_r > p_r \), the closest points can be skipped. Unlike the \( k_d \)-tree, vp-trees are designed for general metrics.

A GNAT (Geometric Near-neighbor Access Tree) is similar to a vp-tree in that it is based on assigning points to a closest reference point [15]. GNAT nodes have \( m \) subtrees, each with a reference point \( p \). All other points are assigned to the subtree to whose reference point they are closest, as illustrated in Fig. 9.2c. As in the vp-tree, a radius \( p_r \) is stored with each reference point. In the simplest case, \( m \) is two and the points are divided into two sets, \( U \) and \( V \), according to their distance to the two reference points \( u \) and \( v \). Given a query point \( q \) and a query radius \( q_r \) it is possible to decide if the points in the two sets should be visited using the reference points. When calculating \( ||u - q|| \) it is possible to skip all the points \( v_i \in V \) if

\[
\min_{v_i \in V} (||u - v_i||) > ||u - q|| + q_r
\]

or

\[
\max_{v_i \in V} (||u - v_i||) < ||u - q|| - q_r.
\]

Therefore, each node stores \( \min_{v_i \in V} (||u - v_i||) \) and \( \max_{v_i \in V} (||u - v_i||) \) to accelerate queries. These extra calculations require more computations but the gain is more discriminatory power.

Some additional speed can be obtained by collapsing subtrees with under max-points points into leafs. For \( k_d \) and vp-trees, this is the only parameter, whereas the GNAT also has the extra parameter \( m \). Searching for points outside the query radius (when \( t \) is less than zero) can be done by modifying the pruning technique slightly for all three data structures. This has been done in the experiments; how it is done is trivial and outside the scope of this paper. The three trees are illustrated on a larger data set in Fig. 9.3.

9.3 Experimental setup

To examine the performance of the methods, experiments have been performed on real descriptors, calculated from commercially available molecules from ZINC version 8 [48]. Two different sets of descriptors were generated: one using the
9.4 Results

Initial experiments were focused on parameter tuning of the three data structures. In these experiments the number of descriptors were kept at 100,000 and the Tanimoto threshold was varied from −0.3 to 1.0 in 0.05 increments. $\text{max-points}$ was varied from two to 20 in increments of two; for the GNAT, $m$s of 2, 4, 6 and 8 were tested. Judging from the query times (not presented) the best overall $\text{max-points}$ for both $kd$- and vp-trees was chosen to be 14, while $\text{max-points}$ for GNAT was chosen to be 18 and $m$ was chosen to be four. Of the three data structures, the vp-tree seems to have the most stable query time for different $\text{max-points}$.

Figure 9.4 presents the three methods along with a linear scan measured on query time and the number of distance calculations performed. In the upper part of the graph, where query time is presented, it is clear that the $kd$-tree performs much worse than both the vp-tree and the GNAT. For some thresholds,
particularly on the CDK set, it is even worse than a linear scan. The reason for this is easily explained by the bottom part of Fig. 9.4, in which it is clear that the \(kd\)-tree performs many more distance calculations than the other two data structures. Not surprisingly, the GNAT performs significantly fewer distance calculations than the \(vp\)-tree, due to its tighter bound when pruning subtrees. This does not, however, grant the GNAT faster query times on the tested data, as seen from the top part of Fig. 9.4.

The data structures were also tested with a fixed threshold of 0.9 and a database size that varied from 100,000 to 1,000,000 in 100,000 increments. The result of this comparison is presented in Fig. 9.5, from which it is seen that the internal relationship between the three methods seems to repeat itself for larger database sizes, and that all three methods are far superior to a simple linear scan. Interestingly, the \(vp\)-tree becomes faster than the GNAT when moving from the MVD to the CDK data base. This is explained by the number of distance calculations which is practically the same for the two methods on the CDK set, while the \(vp\)-tree still has a much smaller overhead. However, this is not to be expected for all thresholds.

The data used in the experiments was generated from real descriptors, but an interesting question is if the underlying structure of the data sets matter, or if random entries would give rise to the exact same results. Observing the data structures on random data would also render it possible to examine if the only reason for the difference between the results on MVD and CDK is solely due
9.4. Results

Data structures compared for different database sizes

MVD

CDK

Query time (s)

Calculations

Database size

Database size

kd-tree — vp-tree —— GNAT ———— scan all

Figure 9.5: Comparison of the three data structures on both data sets. The comparison is on both time and number of distance calculations. The database size is varied on the primary axis while the threshold is kept steady at 0.9. The secondary axis is kept on a log scale to highlight the difference between the data structures.

to dimensionality of descriptors.

As the MVD and CDK data is normalised, random descriptors were drawn from a standard normal distribution. One hundred thousand vectors of length nine and 14 were generated for comparison with MVD and CDK data respectively. The methods were tested with a range of different thresholds, as in the tuning experiments with the real data. Experiments were performed on all three data structures, but for brevity only vp-tree results are presented in Fig. 9.6; the results for kd-trees and GNATs are similar.

From Fig. 9.6 it is clear that the vp-tree is far slower and prunes far less on the random data than on the original MVD and CDK data. A possible explanation could be that many entries (e.g. number of bonds) in the MVD and CDK sets are discrete, and therefore data points cluster together rather than being distributed evenly; and therefore they are easy to discriminate against.

To examine if the discrete nature of the data explains the observations, experiments in which descriptors were drawn from a binomial distribution (eight trials, each with success probability of 0.5), were performed. Experiments with other numbers of trials were also performed with similar results. The discrete descriptors were normalised as the original data before the experiments were carried out.

Surprisingly, the queries on discrete random data are just as slow and perform just as many distance calculations as on the data from the normal dis-
Chapter 9. Data structures for accelerating queries on real valued vectors

90

vp-trees on random data from normal distribution

MVD

CDK

Query time (s)

Calculations

Tanimoto threshold

−1\frac{1}{3} 0 1

Tanimoto threshold

−1\frac{1}{3} 0 1

original —— generated ———

Figure 9.6: Comparison of vp-trees performance and pruning degree on real data and data drawn from a normal distribution. The data from the normal distribution has the same dimensionality as the vectors generated by MVD and CDK.

The explanation of these observations might very well be highly correlated entries within the MVD and CDK data. Closer examination of the MVD and CDK data reveals that some entries of the vectors are very highly correlated. For example, the number of aromatic bonds and the number of aromatic atoms of the CDK descriptors have a \( r^2 \) of 0.995.

To test if high correlation has an influence on the execution time of queries random data was generated by, for each vector, setting all entries for that vector to the same random number from a standard normal distribution. This entails that all points lie on a line in \( \mathbb{R}^9 \) for the MVD data and \( \mathbb{R}^{14} \) for the CDK data. Running the same experiments show that the vp-tree and the GNAT are very fast on this data; vp-tree results are presented in Fig. 9.8. The reason why this happens is, that both data structures use a strategy in which the entire subtree is accumulated without performing any distance calculations when it can be reasoned that all points in a subtree lie within a query radius. The \( kd \)-tree can not use this strategy and its query time is almost identical to that on the real data (Fig. 9.9).

9.5 Conclusion

The work presented in this paper allows for very fast querying of chemical databases in which molecules are represented as real valued descriptors. The
9.5. Conclusion

Figure 9.7: Comparison of vp-trees performance and pruning degree on real data and data drawn from a binomial distribution. The data has the same dimensionality as the vectors generated by MVD and CDK.

Experiments indicate that the vp-tree or the GNAT are the best choice as accelerating data structure, and that \( kd \)-trees do not work well on chemical data. Furthermore, it seems vp-trees are least affected by changing parameters, and they are therefore recommended as the data structure to use, especially for larger dimensional data where the pruning degree becomes closer to that of GNATs which have a larger overhead. All programs developed and experimental data generated as part of this paper, are available upon request.

Future research could focus on more closely examining the highly correlated data to find out what the underlying dimensionality of the data is, and if special data structures could be created for handling this property. The methods presented here should also work on fingerprints and it would be interesting to see a comparison with data structures tailored to handle these. There are also other metric data structures not covered by this study, and especially IO efficient data structures would be interesting as more data arrives. Parallelising the data structure and the queries would also be a potential area of further investigation.
Figure 9.8: Comparison of vp-trees performance and pruning degree on very highly correlated data. The data has the same dimensionality as the vectors generated by MVD and CDK.

Figure 9.9: Comparison of kd-trees performance and pruning degree on very highly correlated data. The data has the same dimensionality as the vectors generated by MVD and CDK.
Chapter 10

Using inverted indices for accelerating LINGO calculations

The article *Using inverted indices for accelerating LINGO calculations* presented in this chapter has been submitted and accepted to *Journal of Chemical Information and Modeling* [66] on the condition on minor revisions. A revised version of the paper addressing all the considerations in the original review has been submitted. The article presented here is the revised edition.
10.1. Introduction

A common task in drug discovery is the computational analysis of chemical compounds which can take the form of e.g. predicting numerical properties such as the logP value or performing screening studies in which new drug candidates are sought from a large database of available molecules. Both problems are often managed by representing molecules as either graphs or 3D structures. The number of available molecules is increasing rapidly: the ZINC database contains more than 13 million molecules [48] and the GDB-13, containing all synthesizable molecules up to size 13, contains 970 million chemical compounds [13]. Therefore, novel vector models such as fingerprints and numerical descriptors have been proposed and tested as predictors and screening tools. Several studies have examined the effectiveness of these methods [80,121] and other studies have
Chapter 10. Using inverted indices for accelerating LINGO calculations

\[ S = c1cccc1Cl \]

(a) Example SMILES

\[ S' = c0cccc0L \]

(b) Simplified string

<table>
<thead>
<tr>
<th>LINGO</th>
<th>freq.</th>
<th>LINGO</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>c0cc</td>
<td>1</td>
<td>c0cc</td>
<td>54</td>
</tr>
<tr>
<td>0ccc</td>
<td>1</td>
<td>0ccc</td>
<td>22</td>
</tr>
<tr>
<td>cccc</td>
<td>2</td>
<td>cccc</td>
<td>30</td>
</tr>
<tr>
<td>ccc0</td>
<td>1</td>
<td>cccc'</td>
<td>42</td>
</tr>
<tr>
<td>cc0L</td>
<td>1</td>
<td>ccc0</td>
<td>7</td>
</tr>
<tr>
<td>c0L</td>
<td>1</td>
<td>cc0L</td>
<td>101</td>
</tr>
</tbody>
</table>

(c) LINGOs

(d) Verbose rep.

(e) Inverted indices rep.

Figure 10.1: Example of generating LINGO multisets in the verbose and inverted indices representation. The inverted index lists contain references to the original SMILES string \( S \).

already examined the acceleration of queries into molecular databases represented as vectors [7,63,65,103,110]. If a database is stored as canonical SMILES strings, both fingerprints and numerical descriptors require an explicit model of the chemical structure as a graph or as a 3D structure to be constructed. The LINGO representation [119] avoids this problem by generating a representation of a molecule directly from canonical SMILES strings [124] of the molecules.

Given a canonical SMILES string for a molecule the set of LINGOs in this study is generated by replacing all ring closure numbers with zero and all occurrences of Br and Cl with R and L respectively. The LINGOs are defined as all substrings of size \( q \) in the resulting string. Figure 10.1 (a, b, c) illustrates the generation of the LINGOs for a small molecule with \( q = 4 \), as this value of \( q \) is suggested as optimal by two previous empirical studies [38, 119]. In the article introducing LINGOs the model was used to predict ADME (absorption, distribution, metabolism and excretion) properties with an \( r^2 \) correlation of 0.93 to experimentally observed values of \( \log P \) [119]. The study furthermore compared the intermolecular similarity between bioisosteric molecules with that of randomly sampled pairs of molecules using the LINGOsim measure, finding a large discriminatory power.
10.2 Previous work

A multiset is a set where elements are allowed to occur multiple times. Given two multisets of LINGOs the LINGOsim is defined as the Tanimoto coefficient between the sets. If \( A = \{ 'cccc', 'c0cc', 'cccc', 'cccc' \} \) and \( B = \{ 'cScc', 'cccc', 'cccc' \} \) then

\[
\text{LINGOsim}(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|\{ 'cccc', 'cccc' \}|}{|\{ 'cccc', 'c0cc', 'cccc', 'cccc', 'cScc' \}|} = \frac{2}{5}
\]

Grant et al. used the LINGOsim as a ranking tool in a screening study in which it performed comparable to Daylight fingerprints [38].

Whereas most previous work concerning fingerprints and numerical descriptors has been focused on accelerating queries into large databases [7, 65, 103, 110] most of the work concerning LINGOs has been focused on the calculation of LINGOsim between pairs of molecules. The first fast LINGOsim method was based on constructing a finite state machine (FSM) based on the transformed SMILES string in Figure 10.1 (b). In the initial study of this FSM it was described as being constructed as repeatedly inserting LINGOs into a trie [38], which can be done in linear time [2].

A less complicated method called SIML is based on encoding the frequency table from Figure 10.1 (c) as two word-arrays: one containing the codes for the LINGOs and one for encoding their frequency [41]. One study suggests that a size of four is optimal [38], and as every ASCII character can be encoded in 8 bit, a LINGO of size four can be encoded in 32 bits which is precisely a word on common hardware. If the lists are sorted by LINGO code the LINGOsim can be calculated by a parallel run through two pairs of lists which is linear in the number of distinct LINGOs in the two strings. The SIML study used SIMD instructions and GPU hardware and compared its performance to that of a commercial implementation by OpenEye with speedups in the order of a factor of 80. It is, however, limited to a \( q \) of four or less on 32-bit hardware and requires a very fast GPU to obtain a significant speedup.

10.3 Verbose representation

A problem with the SIML encoding scheme is that it is bound to both \( q \) and the word size of the underlying hardware. Instead of encoding each character of the LINGO, we propose generating ids for each LINGO in the SMILES strings as they are observed when reading the strings. Multiple occurrences of the same LINGO in one SMILES string is given different ids as illustrated in Figure 10.1 (d). Generating these ids is done by using a trie as in the FSM method, storing ids in the leaves. Inserting LINGOs in the trie is done in linear time and the entire set of SMILES strings is inserted in linear time. If each id is stored in one word the verbose representation can store more different LINGOs than the SIML encoding in the same word size. This is because non-occurring LINGOs are not stored, in contrast to SIML which also encodes impossible LINGOs such as ‘h43f’ or ‘c(]S’. The SIML representation uses two
Chapter 10. Using inverted indices for accelerating LINGO calculations

<table>
<thead>
<tr>
<th>Times LINGOs repeated</th>
<th>Maybridge avg. per molecule</th>
<th>ZINC avg. per molecule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>26.353</td>
<td>42.793</td>
</tr>
<tr>
<td>2</td>
<td>2.434</td>
<td>8.391</td>
</tr>
<tr>
<td>3</td>
<td>0.159</td>
<td>3.352</td>
</tr>
<tr>
<td>more than 4</td>
<td>0.000</td>
<td>9.001</td>
</tr>
</tbody>
</table>

Table 10.1: Frequencies of size four LINGOs in the Maybridge and ZINC data.

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Maybridge</th>
<th>ZINC</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIML rep.</td>
<td>64.4</td>
<td>127.1</td>
</tr>
<tr>
<td>Verbose rep.</td>
<td>31.8</td>
<td>155.4</td>
</tr>
</tbody>
</table>

Table 10.2: The average word consumption of size four LINGO multisets in the SIML and verbose representation on the Maybridge and ZINC data. The ZINC data is generated using CDK which adds explicit hydrogen information, which accounts for them being larger than the Maybridge data.

arrays for representing a LINGO multiset but the multisets can be represented more verbosely with only one array with multiple occurrences of the same LINGO represented as multiple different ids, as illustrated with the two 'cccc' LINGOs in Figure 10.1 (d). This verbose representation has the downside that it will potentially take up more space: the SMILES string 'ccccccc' would take up two words in SIML but five in the verbose representation. However, analysing the test data from Maybridge presented in the Experiments section reveals that the average SMILES length is 35.2 (yielding 64.4 words in SIML) whereas the average number of ids in the verbose representation is only 31.8 as demonstrated in Table 10.2. The ZINC data has slightly longer SMILES strings with more repetitions (Table 10.1) and the SIML representation is slightly shorter than the verbose representation. If 4 byte integers are used for representing the ids the verbose representation takes up less than four times the memory of the original SMILES strings.

The presented verbose representation can be interpreted as sparse fingerprints in which case the LINGOsim is the Tanimoto coefficient between fingerprints. This implies that all data structures developed for effectively performing queries into fingerprint databases [7, 63, 65, 103, 110] can be used on LINGOsim queries into databases of LINGO multisets.

10.4 Inverted indices method

This paper proposes calculating the LINGOsim between a target LINGO multiset and a database of LINGO multisets by storing the database as inverted indices. Inverted indices are also used to handle the $T$-occurrence problem from string algorithms: given a query string and a database of strings, identify all database strings that share more than $T$ substrings of size $q$ with the query [76, 102]. The
10.4. Inverted indices method

$x_1 : 0,1,2,3 \quad x_2 : 0,2,3,4$
$x_3 : 2,4,5 \quad x_4 : 1,3,4$

(a) Input verbose representation.

<table>
<thead>
<tr>
<th>$I_0$</th>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$I_3$</th>
<th>$I_4$</th>
<th>$I_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$x_1$</td>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_2$</td>
<td>$x_3$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>$x_4$</td>
<td>$x_2$</td>
<td>$x_3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$x_3$</td>
<td>$x_4$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) Inverted indices datastructure.

$x : 2,4,5$

(c) Query verbose representation.

$\begin{array}{cccc}
  x_1 & x_2 & x_3 & x_4 \\
  1 & 2 & 3 & 1 \\
\end{array}$

(d) The $C$ counting vector.

Figure 10.2: Example of the data structure and algorithm. The input (a) is used to build an inverted indices table (b), which lists all molecules associated with a given id. To perform a query with a molecule $x$ (c) the counting vector $C$ is created by running through list $I_2, I_4$ and $I_5$ in the inverted indices table and increasing the entries corresponding to the encountered molecules (d). From this vector the similarities can be computed.

The idea is to store the LINGO multisets as a vector where each cell represents one of the LINGO ids from the verbose representation. A database of $n$ multisets $x_1, \ldots, x_n$ can be stored as a vector $I$ where each cell $I_k$ stores a list of all the multisets containing $k$, as illustrated in Figure 10.1 (e). The LINGOSim between a multiset $x_i$ and every other multiset $x_j$ in the database can be calculated by first calculating the intersection sizes between $x_i$ and every $x_j$. To do this, let $C$ be a counting vector of length $n$ initialised with all zeros and let $x_i$ contain the $m$ ids $x_{i,1}, \ldots, x_{i,m}$. Next, traverse the inverted indices lists $I_{x_{i,1}}, \ldots, I_{x_{i,m}}$ and increment the counter $C_{x_j}$ every time $x_j$ is observed. An example of this is shown in Figure 10.2. Afterwards $C_{x_j}$ will contain $|x_i \cap x_j|$ from which the LINGOSim can be calculated by using $|x_i \cup x_j| = |x_i| + |x_j| - |x_i \cap x_j|$. Note that this strategy only works if the verbose representation contains unique ids for multiple occurrences of LINGOs within the same multisets as they would otherwise be counted multiple times when the inverted indices lists are traversed.

The inverted indices data structure is not limited to LINGO multisets but can also be used to calculate the Tanimoto coefficients for general fingerprints. The inverted indices are constructed by first identifying the largest id in the data set. This id is used to find the size of $I$ so that this can be allocated. Next, run trough all the LINGO multisets $x_i$ and insert them into the lists $I_{x_{i,1}}, \ldots, I_{x_{i,m}}$. All this takes linear time and takes up memory linear in the
size of the verbose representation.

10.5 Experiments

Experiments were performed on data from Maybridge, taken from the SIML study and on data generated using the Chemistry Development Kit [105] canonical SMILES generator on molecules from the ZINC database version 8, subset 10 [48]. The SIML data contains 4,096 SMILES strings with an average length of 35.2 and the ZINC data contains the first 65,536 SMILES strings from ZINC with an average length of 158.4. LINGOs were generated for \( q = 4 \) as in previous studies using the method from the original paper [119]. Three methods were evaluated: the LINGOsim calculator from OpenEye [89] (OE), the SIML implementation without GPU support and a C implementation of our inverted indices method (IIM). The OpenEye LINGOsim calculator was chosen because it is used elsewhere in the literature [119]. It is not FSM based. Each method was measured as in the SIML study, that is on the time it takes to fill out a similarity matrix for a set of LINGO multisets, not including the time to load the LINGOs into memory or outputting the similarity matrix to disk or screen. As the matrix is symmetric, it is only necessary to compute half the entries. However, the matrix in our experiments is too big to fit in memory, and it is therefore streamed, which means that the old values can not be read and every entry has to be computed explicitly by all three methods. The resulting matrix can be used for performing statistical calculations, clustering or similar data analysis.

All experiments were performed on an Intel Core 2 Quad Q9450 2.66 GHz machine with 4 GB of RAM running Ubuntu version 9.04 with GCC 4.3.3 using the \(-O3\) optimisation flag. Presented results are from a single experiments.

10.6 Results

As presented in Table 10.3, re-running the OE and SIML implementations on the Maybridge benchmark data yields results very similar to those in the original study. The data in the first column is taken from Table 2 and Table 3 in the SIML article [41] – the GPU measurements are not from the same machine as the first four rows. As can be seen from Table 10.3 the IIM method is faster than all the tested methods, even those using multiple cores and GPUs. For one core the speedup compared to the OpenEye implementation is of a factor of 37 and drops to 31 when the implementations use all four cores. Compared to the SIML implementation the speedup is of a factor of 13 when both implementations are run on one core and 11 when the implementations are allowed to use all four cores. In all cases the IIM method outperforms the previous methods. Surprisingly, the one core version is faster than all but the GPU implementations.

Running the implementations on the ZINC data set yields the running times presented in Table 10.4. There is a large difference in the length of the SMILES strings between the two data sets and the observations from the ZINC set is
10.6. Results

<table>
<thead>
<tr>
<th>algorithm</th>
<th>Haque et al. [41] time (ms)</th>
<th>Replicated time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OE</td>
<td>15 060</td>
<td>14 860</td>
</tr>
<tr>
<td>SIML</td>
<td>5 460</td>
<td>5 421</td>
</tr>
<tr>
<td>OE 4 cores</td>
<td>3 880</td>
<td>3 860</td>
</tr>
<tr>
<td>SIML 4 cores</td>
<td>1 420</td>
<td>1 435</td>
</tr>
<tr>
<td>SIML GPU (GeForce GTS 250)</td>
<td>270</td>
<td>–</td>
</tr>
<tr>
<td>SIML GPU (Tesla T10)</td>
<td>215</td>
<td>–</td>
</tr>
<tr>
<td>IIM</td>
<td>–</td>
<td>407</td>
</tr>
<tr>
<td>IIM 4 cores</td>
<td>–</td>
<td>125</td>
</tr>
</tbody>
</table>

Table 10.3: Results from experiments replicated from Haque et al. [41] on the Maybridge data containing 4096 SMILES strings, along with the results of using the IIM method. The GPU running times were not replicated as we did not have access to the same hardware.

<table>
<thead>
<tr>
<th>SMILES</th>
<th>OE 1 core</th>
<th>OE 4 cores</th>
<th>SIML 1 core</th>
<th>SIML 4 cores</th>
<th>IIM 1 core</th>
<th>IIM 4 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 384</td>
<td>1 493</td>
<td>378</td>
<td>176</td>
<td>44</td>
<td>67</td>
<td>17</td>
</tr>
<tr>
<td>32 768</td>
<td>5 928</td>
<td>1 502</td>
<td>703</td>
<td>178</td>
<td>270</td>
<td>72</td>
</tr>
<tr>
<td>65 536</td>
<td>23 791</td>
<td>6 022</td>
<td>2 812</td>
<td>718</td>
<td>1 093</td>
<td>294</td>
</tr>
</tbody>
</table>

Table 10.4: Timing results in seconds from running the implementations on the CDK generated SMILES strings on the ZINC database.

Therefore a bit slower than that of the Maybridge set and gives rise to different observations in speedup. For one core the speedup between OE and the IIM method drops to a steady factor of 22 across the tested database sizes and the speedup between the SIML implementation and the IIM method drops to a factor of 2.6. The speedups remain the same when using all four cores.

Experiments not presented here were performed to examine if the data size or the data structure is responsible for the decline in speedup. Experimental data was generated by transforming the Maybridge data, extending the lines using concatenation to match the length of the lines in the ZINC data. The ZINC lines were shortened to match those of the Maybridge data. The experiments revealed that both factors contributed equally to the deterioration of the speedup.

There is also a slight difference in the achieved speedup on four cores when going from the Maybridge to the ZINC data. On the Maybridge data, the IIM achieves 81% of linear speedup, while 93% of the achievable speedup is gained on the ZINC data (using all 65,536 SMILES strings). The previously mentioned experiments revealed that exactly half of the improvement was a result of the ZINC data containing longer SMILES strings. The other half was a result of the structure of the ZINC data, which contains explicit hydrogen, yielding an increased number of shared LINGOs.
Chapter 10. Using inverted indices for accelerating LINGO calculations

The presented tables do not include time to parse the SMILES files. Initial studies revealed that parsing files containing SMILES strings and converting them to the verbose representation takes less than 20% of the total running time on the Maybridge data, and less than 1% on the ZINC data. Converting this data to the inverted indices representation is included in the presented execution times, but accounts for less than 1% of the time spent in the algorithm.

10.7 Conclusion

The number of available molecules is ever increasing and new methods are needed to handle the large chemical databases. This paper presents a reduction from LINGO multisets to sparse fingerprints making it possible to implement methods for performing rapid queries in molecular databases with the LINGOsim similarity measure by using the Tanimoto coefficient in fingerprint databases.

This paper also presents the inverted indices method for storing LINGO multisets along with a method for rapidly calculating the similarity matrix for such a collection. The presented algorithm has been implemented and tested on standard hardware and was observed to be more efficient than other current methods, outperforming them in all tests. The SIML method tested against in this study was designed for a $q$ of four whereas the verbose representation is independent of $q$. The presented method makes it possible to analyse very large data sets without the need for GPUs or other types of specialised hardware. The tested implementation along with the test data is available at http://birc.au.dk/~tgk/ii.

There are two interesting directions for future research, namely statistical analysis of very large data sets using the inverted indices method and acceleration of queries into large data bases by using the reduction to sparse fingerprint presented in this paper.

Acknowledgement

The authors would like to thank Imran Haque for making the SIML implementation along with the SIML test data available and Open Eye for an academic license to the OEChem TK.
Chapter 11

Virtual screening using a largest common chemical subtree method

The article *Virtual screening using a largest common chemical subtree method* presented in this chapter is a work in progress. A poster presentation about the work was given at the 17th annual *International Conference on Intelligent Systems for Molecular Biology* and 8th European *Conference on Computational Biology* in Stockholm, 2009 [67].
Virtual Screening using a Largest Common Chemical Subtree Method

Thomas G. Kristensen*  Michael H. Christensen†
René Thomsen‡  Christian N. S. Pedersen§

Abstract
This article presents a novel ligand based virtual screening method called LarCCS and is based on a simplified representation of molecules in which molecules are reduced to trees and compared using a maximum common subtree algorithm. The tree comparison method is a variation of a standard maximum common subtree method, where chemical properties of the original molecules are taken into consideration when comparing trees. The performance of LarCCS has been evaluated and compared to a fingerprint based method (CDK) and a numerical descriptor based method (CFDM) on the directory of useful decoys (DUD) and maximum unbiased validation (MUV) data sets. The experiments indicate that LarCCS is slightly better than CFDM and slight worse than CDK when compared using the one percent enrichment and AUC values on the two data sets. However, many of the binders identified by LarCCS are different than those found by CDK, and the method may be a valuable supplement to the CDK screening method.

11.1 Introduction

Drugs are typically discovered by chance in a trial-and-error manner using high-throughput screening methods that use in vitro experiments to measure the activity of a large number of drug candidates against a given target. This process is very expensive and time consuming. Computational methods might therefore be used to lower the cost of drug discovery and accelerate the discovery of novel drugs. Virtual Screening (VS) is the computational technique in which computer models of the targets and/or potential binders are examined to identify promising drug candidates. Afterwards, lab experiments can be conducted to further examine the drug candidates.

*Bioinformatics Research Centre (BiRC), Aarhus University, C. F. Møllers Allé 8, DK-8000 Århus C, Denmark. E-mail: tgk@birc.au.dk.
†Molegro ApS, C. F. Møllers Allé 8, DK-8000 Århus C, Denmark. E-mail: mhc@molegro.com.
‡Molegro ApS, C. F. Møllers Allé 8, DK-8000 Århus C, Denmark. E-mail: rt@molegro.com.
§Bioinformatics Research Centre (BiRC), Aarhus University, C. F. Møllers Allé 8, DK-8000 Århus C, Denmark. E-mail: cstorm@birc.au.dk.
There are two major VS methods: target based and ligand based. The target based methods use a known 3D structure of the protein target, typically obtained by X-ray crystallography. Potential binders are typically docked into this structure to assess if they will bind to the target. This is often computationally very expensive but it can potentially identify novel binders with little structural similarity to already known binders. Some well known docking programs are DOCK, GOLD, Glide and MVD [30,32,55,114].

The ligand based methods require knowledge of existing active binders, which are used as templates for identifying new potential binders. This process is often faster than the docking method. Some studies show that ligand based virtual screening perform better than target based methods, but the result is often molecules that are very similar to the original binder [29]. Several types of ligand based methods exists, e.g. binary fingerprint screening (Daylight), molecular descriptor based techniques or molecular 2D/3D similarity [26,35,74].

This article presents a novel molecular structure similarity based method, in which molecules are represented as trees and compared using their largest common subtree. Restrictions based on the atomic properties of the molecule are imposed in order to obtain chemically similar molecules. The method is called LarCCS (Largest Common Chemical Subtree) and has in this study been evaluated on two commonly used benchmark data sets, DUD and MUV. The results are analysed and compared to those produced by a commercially available descriptor based method (CFDM) and an open source fingerprint based method (CDK).

The experimental results indicate that the LarCCS method performs slightly better than CFDM and that it produces results slightly worse than those produced by CDK. Further investigations reveal that many of the binders identified by LarCCS are not found by CDK, which indicates that the LarCCS method supplements CDK in a useful way.

11.2 Method

We use a method inspired by Feature trees [93] and reduced graphs [35] as a simplified model of chemical compounds. In our approach molecules are represented by trees in which nodes represent individual atoms or ring systems. Figure 11.1 illustrates an example in which a small molecule is converted to a tree. As illustrated in Figure 11.1, connected ring systems are collapsed to a single node instead of multiple connected nodes, as in reduced graphs [35]. By representing molecules as trees a higher computational efficiency is gained, as algorithms designed for trees and not general graphs can be utilised.

Each node contains information about i) the number of atoms it represents, ii) the charge of the atom(s) it represents, and if the node only represents one atom, iii) whether or not that atom is capable of hydrogen donor or acceptor interaction. The charge and hydrogen donor and acceptor activity is calculated using CDK [105]. Hydrogen atoms are not represented explicitly in the trees. Converting molecules to trees is done efficiently by repeatedly removing cycles using a depth first traversal [25].
11.2. Method

Figure 11.1: An example of a conversion from a molecule to a tree. Ring systems are collapsed, but the number of atoms in the ring system is saved in the resulting node.

Figure 11.2: The two depicted trees share a common subtree of size two. It is not possible to expand the subtree to cover any more nodes due to atom count and donor/acceptor restrictions.

Trees are compared based on the size of their largest common subtree. It is possible to match nodes between two trees if they conform to the following constraints:

- The number of atoms the nodes represent must not differ by more than a certain number of atoms.
- The charge must not differ by more than a certain limit.
- Hydrogen donor nodes only match other hydrogen donor nodes.
- Hydrogen acceptor nodes only match other hydrogen acceptor nodes.
- Nodes that are both hydrogen donors and acceptors only match nodes that are also both hydrogen donors and acceptors.

In Figure 11.2 two trees are depicted, which share a common subtree of size two. The algorithm for calculating the size of the largest common subtree is an extension of the Maximum Common Subtree Isomorphism algorithm [118] combined with the Munkres algorithm [14, 84]. The algorithm has been extended from unlabeled trees to trees labeled with the previously mentioned chemical information, which introduced a challenge as only chemically similar nodes can be matched. This has been handled by altering the node matching part of the algorithm. Calculating the number of shared nodes has the complexity $O(n^5)$,
Chapter 11. Virtual screening using a largest common subtree method

11.2.1 Chemical Feature Distance Matrix

The LarCCS method was inspired by the CFDM (Chemical Feature Distance Matrix) method developed by Molegro [83]. The CFDM descriptors are obtained by calculating the minimum, maximum, and mean topological distance between all pairs of chemical features. The topological distance is defined as the smallest number of covalent bonds between two given features.

The following chemical features are taken into account: hydrogen acceptors, hydrogen donors, positively and negatively charged atoms, and ring systems. A minimum charge of ±0.2 is required for an atom to be considered charged. See Figure 11.3 for a small example in which the smallest distance between a hydrogen donor and a hydrogen acceptor is five (highlighted).

Unit variance and mean centering [73] is applied to the set of resulting descriptor vectors to ensure no descriptor dominates the others. The descriptors are ranked according to the Tanimoto coefficient [113] as initial studies showed it to be superior to cosine, Euclidian and Manhattan distance measures.

Figure 11.3: Example of a Chemical Feature Distance Matrix calculated on an example molecule. The shortest path between a hydrogen donor and acceptor with length five is highlighted in the molecule and in the resulting matrix of shortest path. Two other matrices containing longest path and average path are also created, and the matrices are combined into one descriptor vector of the molecule.

where \( n \) is the maximum number of nodes in the largest of the trees. This is a very rough upper bound and not a practical problem as the trees are very small, typically with less than 20 nodes.

Given a query molecule, the screening is performed by ordering the set of molecules according to the number of shared nodes between the query tree and each molecule tree. Since large trees are likely to match all small query tree perfectly, a normalisation term inspired by Raymond et al [95] is used to penalise large trees and increase the number of distinct values. The LarCCS similarity between two trees is defined as

\[
S(t_1, t_2) = \frac{T(t_1, t_2)^2}{T(t_1, t_1) \cdot T(t_2, t_2)}
\]

where \( T(t_1, t_2) \) is the number of shared nodes between the trees \( t_1 \) and \( t_2 \).
11.3 Experimental setup

Experiments have been performed to compare the LarCCS method to the CFDM, CDK and DOCK method. The methods have been evaluated on the DUD (Directory of Useful Decoys) version 2 data set [47], which is widely used to benchmark virtual screening methods [21, 96, 115, 121]. The DUD set contains results from a docking study using DOCK [78, 81, 123], and we compare our methods to these results. The data set consists of known binders for 40 targets. For each known binder, the data set contains approximately 33 decoy molecules that are structurally similar to the known binder.

The DUD data set was designed for measuring the quality of target based methods, and it has been criticised for being unsuitable for comparing ligand based methods [24, 37, 42, 121], as the known binders are structurally very similar, thus favouring ligand based methods. Therefore, the methods have also been evaluated on the MUV (Maximum Unbiased Validation) data set [97]. The MUV data set contains 30 known binders and 15,000 decoys for each of the 17 targets. Each target has more than 21 scaffolds, i.e. more than 21 distinct groups of actives with distinct general topologies. This makes it a difficult data set for ligand-based methods. 3D conformations of the compounds were generated using CORINA [82] for use with the CFDM method. Cofactors were also removed prior to the screening experiments.

For each target in the DUD and MUV data set, each known binder was removed from the set of molecules and used as query. The known binders and decoys were ordered according to their similarity to the query, using the virtual screening methods tested.

The node matching methods have a set of parameters as described in the method section. These parameters were tuned using only the “Ace” target from the DUD data set. Each parameter was tuned individually without taking the rest of the criteria into account. The optimal number of atoms two nodes are allowed to differ was found to be four, and the optimal maximum charge difference allowed between nodes was found to be 0.2.

To evaluate the importance of topology similarity the algorithm was also run without restrictions on the underlying topology of the trees; that is, any node of the first tree could match any node of the other tree. See Figure 11.4 for an example. In these experiments all but one target performed worse than...
the experiments in which topology was taken into account, when measuring the one percent enrichment. Similarly, experiments have been performed where the restrictions have been switched off and only topology of the trees are taken into account. In all of these experiments, LarCCS without restrictions performed worse than with, when compared using the one percent enrichment. This indicates that taking topology and chemical features into account is important, and both have therefore been used in all the LarCCS experiments.

The LarCCS method has been tested against both target based and ligand based VS methods. For target based methods the DOCK data from the DUD data set [47] was used. In addition to the CFDM method, another ligand based method based on binary fingerprints was tested. The fingerprints were generated using CDK [105] fingerprints of 1024 bits and compared using the binary Tanimoto coefficient [113]. The CDK fingerprints are similar to Daylight fingerprints [31].

11.4 Results

A widely used measure for evaluating the performance of a virtual screening method is its enrichment obtained on a given data set. [42] The enrichment for a given target and one of its known actives is found by ordering the remaining actives and the decoys according to their similarity to the selected active. The enrichment is defined as the percentage of all the known binders found in the top of this ordered list.

The one percent enrichment does not describe the quality of the entire ranking. The overall quality of a ranking can be measured as the AUC (Area Under ROC Curve) which measures the area under the ROC (Receiver Operating Characteristic) curve. A random ranking is expected to have an AUC value of 0.5; an optimal ordering has an AUC value of 1.0.

It should be mentioned that both the one percent enrichment and AUC values have been criticised: the one percent enrichment threshold for being arbitrarily chosen and for not describing the quality of the entire ranking; the AUC value for not highlighting early enrichment [88].

In Figure 11.5 the average enrichment is presented for the three ligand based methods on both the DUD and the MUV data set. The DUD data set also includes data from the target based DOCK method (not available for the MUV set). The enrichment is averaged over all targets, and for each target it is averaged over all the known actives. The first axis (% of ranked database) is presented on a log scale to highlight early enrichment.

If the three methods are compared on the DUD set (Figure 11.5 top) they all perform fairly well, and all are clearly superior to the DOCK method. The fingerprint based method performs slightly better than the other two ligand based methods, and the LarCCS method seems to be slightly better than the CFDM method.

The reason why the three ligand based methods are superior to DOCK is due to the fact that the known binders the the DUD data set are structurally very similar. This also explains why the three methods perform as well as they
11.4. Results

Figure 11.5: Average enrichment for each method, across all 40 targets in the DUD set (top) and all 17 targets in the MUV set (bottom). The reason why the curve for an optimal ordering differs drastically between the two data sets is because of the ratio between actives and decoys (1::≈ 33 for DUD, 1:500 for MUV).

The MUV data set (Figure 11.5 bottom) reveals another pattern: here it seems all three methods perform close to that of a random ordering.

Figure 11.5 only supplies a very basic overview of the methods performance. To inspect early enrichment in detail the average enrichment at one percent for each target is plotted for each of the methods in Figure 11.6 (top) for the DUD data set. The fingerprint method, CFDM and DOCK is plotted against LarCCS to inspect correlation. There seems to be a correlation between LarCCS and
Chapter 11. Virtual screening using a largest common subtree method

DUD data set

Average one percent enrichment for each target

<table>
<thead>
<tr>
<th>Method</th>
<th>4%</th>
<th>17%</th>
<th>37%</th>
</tr>
</thead>
<tbody>
<tr>
<td>LarCCS</td>
<td>8%</td>
<td>37%</td>
<td>17%</td>
</tr>
<tr>
<td>CDK</td>
<td>4%</td>
<td>27%</td>
<td>15%</td>
</tr>
<tr>
<td>CFDM</td>
<td>0%</td>
<td>33%</td>
<td>8%</td>
</tr>
<tr>
<td>DOCK</td>
<td>0%</td>
<td>37%</td>
<td>23%</td>
</tr>
</tbody>
</table>

Average AUC values for each target

<table>
<thead>
<tr>
<th>Method</th>
<th>0.44</th>
<th>0.72</th>
<th>0.95</th>
</tr>
</thead>
<tbody>
<tr>
<td>LarCCS</td>
<td>0.61</td>
<td>0.79</td>
<td>0.97</td>
</tr>
<tr>
<td>CDK</td>
<td>0.55</td>
<td>0.71</td>
<td>0.97</td>
</tr>
<tr>
<td>CFDM</td>
<td>0.20</td>
<td>0.99</td>
<td>0.66</td>
</tr>
<tr>
<td>DOCK</td>
<td>0.66</td>
<td>0.99</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Figure 11.6: (Top) Scatterplot of the one percent enrichment on the DUD data set for LarCCS compared to the three other four methods. For the three ligand-based methods the enrichment is averaged over the known actives. (Bottom) Scatterplot of the AUC values on the DUD data set for each of the four methods. For the three ligand-based methods the AUC value is averaged over the known actives ROC curves.

The fingerprint based method has a higher average one percent enrichment than LarCCS on all but four targets, whereas LarCCS has a higher average one percent enrichment than CFDM on 26 of the 40 DUD targets. The fingerprint based method also has the highest minimum, maximum and average one percent enrichment.

A similar pattern can be observed on the MUV data set in Figure 11.7 (top). However, the enrichment values are much lower, indicating that the MUV data set is more difficult.

The average AUC value for each target in the DUD set is presented in Figure 11.6 (bottom). The plot indicates that there also is a correlation between LarCCS and the two other ligand based methods when using AUC as a measure on the DUD set. LarCCS has a higher average AUC than the fingerprint method on 11 of the targets, and on 22 of the 40 when compared to CFDM. The fingerprint method is still slightly superior when using the AUC measure on
11.4. Results

MUV data set
Average one percent enrichment for each target

<table>
<thead>
<tr>
<th>Target</th>
<th>LarCCS</th>
<th>CDK</th>
<th>CFDM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1%</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>3%</td>
<td>14%</td>
<td>6%</td>
</tr>
<tr>
<td></td>
<td>8%</td>
<td>5%</td>
<td>2%</td>
</tr>
</tbody>
</table>

Average AUC values for each target

<table>
<thead>
<tr>
<th>Target</th>
<th>LarCCS</th>
<th>CDK</th>
<th>CFDM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.49</td>
<td>0.52</td>
<td>0.51</td>
</tr>
<tr>
<td></td>
<td>0.73</td>
<td>0.68</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>0.56</td>
<td>0.60</td>
<td>0.71</td>
</tr>
</tbody>
</table>

Figure 11.7: (Top) Scatterplot of the one percent enrichment on the MUV data set for LarCCS compared to the two other methods. In the three ligand-based methods the enrichment is averaged over the known actives. (Bottom) Scatterplot of the AUC values on the MUV data set for each of the four methods. In the three ligand-based methods the AUC value is averaged over the known actives ROC curves.

From Figure 11.5, Figure 11.6 and Figure 11.7 there is a major difference between the DUD and the MUV data set. It is clear from the scatterplots that the DUD set is much easier than the MUV set for all three ligand based methods.

The results indicate that CDK has a slightly higher enrichment than LarCCS. However, the binders identified by LarCCS are different from those identified by CDK. The average percentage of unique known binders for each target found by LarCCS, averaged over all targets is 26% for the DUD data set, and 21% for the MUV data set. The distribution of percentage unique compounds

the DUD set, but not on the MUV set Figure 11.7 (bottom).
Chapter 11. Virtual screening using a largest common subtree method

Average percent binders found by LarCCS not found by CDK

<table>
<thead>
<tr>
<th></th>
<th>DUD</th>
<th>MUV</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 11.8: Distribution of average percent unique binders found by LarCCS compared to CDK in the top one percent of their ranked databases. The percentages are taken as the average over the known binders for each target.

found by LarCCS compared to CDK is presented in Figure 11.8. The hits reported by LarCCS therefore complements CDK in a useful way as a VS tool for identifying a diverse set of novel binders. Each tested ligand based method produced the results for both DUD and MUV in less than 24 hours on a standard laptop computer, indicating that they are all well suited for practical use in the industry. In that time frame, each method screened more than 10 million drug candidates.

11.5 Conclusions

This article has presented LarCCS, a novel screening method based on reduced tree representations of molecules. The method has been tested and compared to two existing ligand based methods, namely CFDM and CDK fingerprints. The methods were tested on the DUD and the MUV data set, both widely used data sets for performing virtual screening studies. The studies were carried out by screening a database of known binders and decoys for each target in the two benchmark data sets. For each binder, the molecules were ranked according to their similarity to this molecule, based on the different methods. The enrichment and AUC values were recorded for each binder on each target. The experiments clearly indicated that the MUV set was more difficult than the DUD data set for all the ligand based methods.

The experiments also indicated that LarCCS performed slightly better than CFDM when comparing one percent enrichment and AUC values. It performed slightly worse than the CDK fingerprints. Although the method did not identify as many known binders (on average) as CDK fingerprints, it did identify many that CDK failed to find. This indicates that the LarCCS method is a valuable supplement to existing virtual screening methods, where it would be desirable to obtain a varied set of binders.

Future work will investigate the effect of allowing partial matching of nodes in which a finer grained measure of similarity between trees will be used.
A Java implementation of the LarCCS method is available online at

cs.au.dk/~tgk/LarCCS/.
Chapter 12

Optimal overlay of ligands using Differential Evolution

The article *Optimal overlay of ligands with flexible bonds using Differential Evolution* presented in this chapter has been published as a conference paper [70]. Before that it was presented as a poster at the 17th annual *International Conference on Intelligent Systems for Molecular Biology and 8th European Conference on Computational Biology* in Stockholm, 2009 [69].


Apart from minor typographically changes, this article is identical to the one presented at the *International Joint Conference on Bioinformatics, Systems Biology and Intelligent Computing, 2009*. 

117
Optimal Overlay of Ligands with Flexible Bonds using Differential Evolution

Thomas G. Kristensen∗  Christian N. S. Pedersen†

Abstract

When designing novel drugs, the need arise to screen large databases of drug candidates (small synthesizable chemical structures) for structures that resemble active ligands, i.e. small chemical structures that are known to react with the target protein. If several active ligands are known one might improve the quality of the search by taking all of these into account. This can be done by generating a meta-structure which summarizes the active ligands and use this meta-structure for querying the database. In this paper we propose a method for making such a meta-structure by making a multiple spatial alignment of a set of active ligands taking the flexibility of chemical bonds into account. We present two implementations of our method. One using Differential Evolution (DE) for numerical optimization, and one using the Nelder-Mead method for numerical optimization. We investigate the quality of the two implementations on a data set from a previous study in the field and conclude that the DE based implementation outperforms the NM based implementation.

12.1 Introduction

In a drug discovery process, one can identify drug candidates by screening large databases of small synthesizable chemical structures (ligands) for structures that resemble active ligands known to react with the target protein in question. If several active ligands are known, the quality of the search might improve if all active ligands are taken into account simultaneously. This is typically done by creating a meta-structure that summarizes the set of active ligands. A step in creating this meta-structure is to superimpose the set of known active ligands. Each ligand has a position and orientation in space, see Fig. 12.1. Furthermore it can have a number of flexible bonds around which individual parts of the ligand may rotate, see Fig. 12.2. The problem of finding a good superimposition of a set of ligands is addressed in [74], which presents a fragment based construction mechanism which builds the superimposition by considering all pairwise ligand superimpositions where one of two ligands are kept rigid while the other is sought superimposed.

∗Bioinformatics Research Centre (BiRC), Aarhus University, C. F. Møllers Allé 8, DK-8000 Århus C, Denmark. E-mail: tgk@birc.au.dk.
†Bioinformatics Research Centre (BiRC), Aarhus University, C. F. Møllers Allé 8, DK-8000 Århus C, Denmark. E-mail: cstorm@birc.au.dk.
In this paper we present a method that computes a superimposition of a set of ligands by making a multiple spatial alignment of the ligands where all ligands are kept flexible. A good superimposition should not twist the ligands into unrealistic conformations. Our multiple alignment method therefore aims to fulfill two goals simultaneously: (1) maximize the overlap between a set of ligands, and (2) keep the ligands in sound conformations (i.e. equal binding modes). We propose a measure which quantifies the quality of a superposition in accordance to these goals and use Differential Evolution (DE) [108] to find a good superimposition. For comparison, we have also implemented our method where the Nelder-Mead method (NM) [87] is used instead of DE for numerical optimization. Experiments on a data set from a previous study in the field [74] show that the DE based implementation clearly outperforms the NM based implementation.

12.2 Objectives

Our objective when computing a multiple ligand alignment of a set of input ligands is to maximize the similarity between the set of ligands, while keeping them in sound conformations. We therefore define the fitness of a particular superimposition of the input ligands using a \textit{Similarity} term, which measures
12.2. Objectives

Figure 12.3: Black and white atoms are from different ligands. (a) The overlap between two atoms is the shared volume (grey). (b) An area that is counted twice (black). (c) Atom caught in non-intuitive optimum.

Figure 12.4: Different methods for calculating the overlap between pairs of atoms. In our method we use the Gaussian measure of overlap.

the overlap of the ligands, and a *Energy* term, which measures the soundness of the conformations of the individual ligands. We will define the terms such that a good multiple ligand alignment should have high similarity and low energy, i.e. its *Fitness* defined as the difference between its *Energy* and *Similarity* should be minimized.

We define the similarity of a superimposition of the ligands as the sum of the similarity between every pair of their atoms. If we are given two atoms we can imagine two spheres centered in these. A natural measure of similarity is the overlap defined by the volume of the intersection between these two spheres, see Fig. 12.3a. A problem with this measure is that it is only non-zero when the two spheres overlap, i.e. bringing the atoms closer together when they are far apart will have no impact of the measure, see Fig. 12.4a. A pairwise symmetric linear approximation of the overlap can be used to ease the calculation, but it does not solve the problem of the missing gradient information. To avoid the problem of missing gradient information, we in our method use a Gaussian measure as described in [90,98], which gives us information about the gradient, even when the atoms are far apart. The Gaussian measure is easy to calculate as

\[ e^{-r_{ij}^2/\alpha^2}, \]

where \( r_{ij} \) is the distance between the two atoms and \( \alpha \) is a parameter that controls the slope of the function. The three measure discussed above (overlap, approximation of overlap, and Gaussian) are illustrated in Fig. 12.4.

If an atom from one ligand is close to two atoms from another ligand, it risks contributing with the same area twice (see Fig. 12.3b). We can avoid this when using the overlap of spheres measure by selecting a radius so that two spheres from the same ligand cannot overlap. If an atom from one ligand is
Figure 12.5: Variations of $\alpha$ can change whether or not we have one or two extreme points.

Figure 12.6: The rings do not have the same number of atoms, resulting in some atoms being placed between two others.

found between two atoms (see Fig. 12.3c), it is trapped in a new optimum as shown in Fig. 12.5a and 12.5b. This pitfall can be an advantage when we try to overlap atomic rings with a different number of atoms (see Fig. 12.6).

Inspired by [90] we classify the different types of atoms, and only count their overlap if they are of the same type. The classifications are hydrogen donor, acceptor, both and non polar, as described in [114].

We define the energy of a superimposition of the input ligands in two steps. First, we penalize conformations where non-bonded atoms from the same ligand come into close contact with each other. This is done by penalizing every pair of non-bonded atoms within a distance less than 2Å of each other with a penalizing term of 1,000. We use $E_{\text{clash}}^\ell$ to denote the total sum of penalizing terms in input ligand $\ell$. Secondly, we take the internal energy of the ligands into account. Our calculations of internal energy are derived from the PLP scoring function first proposed in [34] and extended in [131]. In our method, we use the PLP implementation and parameters from [114].

12.3 Methods

To find a good multiple ligand alignment of the set of input ligands, we search for a superimposition of the ligands that minimize the fitness introduced in the previous section. This is a numerical optimization problem, which we solve using Differential Evolution (DE), first proposed in [108]. We use the DE/rand/1/exp
createOffspring(\text{Parent } p_i) 
\begin{algorithmic}[1]
\State \text{copy genotype of } p_i \text{ to } o_i
\State \text{randomly select parents } p_{i_1}, p_{i_2}, p_{i_3} \text{ where } i_1 \neq i_2 \neq i_3 \neq i
\State n \leftarrow \text{random number from } [0 : \text{dim}]
\State j \leftarrow 0
\While {j < \text{dim} \text{ and RANDOM < CR}}
\State o_i[n] \leftarrow p_{i_1} + F(p_{i_2} - p_{i_3})
\State n \leftarrow (n + 1) \mod \text{dim}
\State j \leftarrow j + 1
\EndWhile
\end{algorithmic}

Figure 12.7: Pseudo code of the recombination procedure for our DE implementation. RANDOM returns a random number between zero and one.

strategy for creating new offspring following the pseudocode in Fig. 12.7. We initially set the parameters \(NP\) (population size), \(CR\) (crossover rate) and \(F\) (scaling factor) to 50, 0.9 and 0.5, based on a previous study.

To investigate the performance of the DE based implementation of our method, we have implemented a version of our method, where we use the Nelder-Mead (NM) method from 1965 proposed in [87] for numerical optimization. The NM method converges to a local optimum rather fast, and it is therefore restarted with a new set of initial points whenever the difference between the best and worst point is below the threshold 0.1. We have experimented with lower thresholds, without an observed improvement in the results. The NM method furthermore has the four parameters \(\rho\), \(\chi\), \(\gamma\) and \(\sigma\), where we have used the standard parameters 1, 2, 0.5 and 0.5. Finally, as a simple benchmark, we have implemented a version, where the naïve method of randomly sampling of points (i.e. multiple ligand alignments) in the search space is used as optimization method.

Both NM and DE are numerical optimizers that need a real encoding on which to operate, and we therefore need to define an encoding. Our problem instance is a set of ligands, each with a position and orientation in space. Furthermore, each ligand has one or more flexible bonds (chosen as in [114], which we can rotate. An individual in our population corresponds to a conformation of this set of ligands. The location of each ligand in space can be represented by a vector in \(\mathbb{R}^3\), its rotation by a directional unity vector in \(\mathbb{R}^3\) and a rotation around this in \([-\pi; \pi] \subset \mathbb{R}\). If the ligand has \(k\) flexible bonds, a configuration of these can be represented by a vector in \([-\pi; \pi]^k \subset \mathbb{R}^k\). The total representation of a ligands configuration can be represented by the concatenation of all these vectors, that is, a vector in \(\mathbb{R}^{3+3+1+k}\). We represent a member of our population by the concatenation of each of its ligands representation. It is therefore a vector in

\[
\mathbb{R}^{(3+3+1+k_1)+\cdots+(3+3+1+k_n)} = \mathbb{R}^{7n+\sum k_i}
\]

where \(k_i\) is the number of rotatable bonds in the \(i\)'th ligand, and \(n\) is the number of ligands. Since our optimization algorithms might violate the constraints
on angles and the length of the normal vector, all encoding are subject to
a minor alternation before evaluation. We normalize the part of the point
that corresponds to a unit vector, and make sure that all angles are in the
interval $[-\pi; \pi]$.

12.4 Experiments

We have tested our approach for multiple ligand alignment on the FlexS-77
dataset [74], that consists of 77 ligands, grouped by the protein they bind to.
In total there are 14 groups of varying size. In all experiments reported below
each ligand is initialized with a random configuration where it is shifted up
to 5 Å from the original FlexS position. Each ligand is furthermore rotated
around a random normal vector, and each rotatable bond is rotated at random.
All atom pair similarities are calculated using an $\alpha$ value of 0.8. Finally, all
methods are terminated after 750,000 evaluations of our fitness function.

Our first experimental focus is to evaluate how well the DE based imple-
mentation of our method performs. To do so we compare its performance
against the NM based implementation and the naive random sampling based
implementation in multiple runs on the FlexS dataset, including randomized
representations of the ligands in the three smallest groups (concanavalain, dlfr
and fructose). We perform 20 runs for each method. Table 12.1 shows the
minimum, mean ($\bar{x}$) and standard deviation ($\sigma$) for the three methods. All
three methods have a low standard deviation across the 20 samples. As can
be seen, DE consistently beats the NM method, who in turn beats the random
sampling. We feared that DE would be trapped in a local optimum, where ran-
dom sampling and NM have the advantage that they sample many new starting
points. Our fear, however, seems to be unjust, as both random sampling and
NM never reach as low a fitness value as DE, as can be seen from the minimum
column.

12.5 Conclusion

In this paper we have presented Differential Evolution based method for com-
puting a multiple spatial alignment of a set of flexible ligands as a superim-
position that aims to maximize the overlap between the set of ligands, while
keeping them in sound conformations. We have investigate the performance
of our method on a realistic dataset and compared it to an implementation
using the Nelder-Mead method instead of Differential Evolution for numerical
optimization. Our experiments show that DE (as expected) is far superior to
the NM method in our application.

As future work, we want to identify an appropriate halting criteria which
could improve the DE based method, e.g. by looking at the change in fitness for
the best individual over a thousand generations. Another improvement could
be to select the initial population more carefully than randomly as done in
the current implementation. One could e.g. select the initial population as a
set of pairwise superimpositions of the input ligands. Our comparison to a
local optimizer (Nelder-Mead) could also look unfair and a goal is therefore to compare it to other global optimizers. Finally, it is of course important to test the performance of our multiple ligand alignment method when using the compute superimposition (meta-structure) for identifying drug candidates, e.g. by performing an enrichment study on a set of known ligands.
Table 12.1: Results of our experiments with mean (\(\bar{x}\)) and standard deviation (\(\sigma\)). Each algorithm performed 750,000 evaluations.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Minima</th>
<th>Mean</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nelder-Mead</td>
<td>2</td>
<td>7.28</td>
<td>1.67 2.86</td>
</tr>
<tr>
<td>Differential Evolution</td>
<td>2</td>
<td>2.28</td>
<td>1.67 2.86</td>
</tr>
<tr>
<td>Sampling</td>
<td>2</td>
<td>2.28</td>
<td>1.67 2.86</td>
</tr>
<tr>
<td>Concatenation</td>
<td>2</td>
<td>2.28</td>
<td>1.67 2.86</td>
</tr>
</tbody>
</table>

Each algorithm was run 20 times to generate the observed data.
Chapter 13

Recombining angles in Differential Evolution

The article *Recombining angles in Differential Evolution* presented in this chapter has been published as a conference paper [62].


Apart from minor typographically changes, this article is identical to the one presented at the *Eleventh Congress on Evolutionary Computation*. 
13.1. Introduction

Many real world optimization problems involve finding angles which optimize some objective function. Optimizing problems with angles may pose a problem when we try to optimize them with a general purpose heuristic that is not designed to handle these. In this paper we investigate if care should be taken when recombining angles in Differential Evolution, and what practitioners can do to handle these.

The paper is organized as follows. First we describe the DE algorithm which is the subject of our study. Next we describe more formally what our concerns are when asked to optimize a problem involving angles and how we will investigate if our concerns are justified. We then present our results along with a discussion and conclude with a short summary of our findings along with some ideas for further research in the area.

13.2 Differential Evolution

Storn and Price first proposed the Differential Evolution (DE) optimization algorithm in [107] and [92]; examples of usage can be found in [19]. The algorithm maintains a population of $NP$ individuals $\Theta_1, \ldots, \Theta_{NP}$. In each iteration (generation) of the algorithm, each of the members of the population is selected as parent of a new offspring. This offspring is evaluated and compared to its parent and whichever is fittest survives for the next generation. The strategy is summarized in the pseudocode below.
**DifferentialEvolution** \((NP, CR, F)\)

1. Initialize \(NP\) random individuals \(\Theta^1, \ldots, \Theta^{NP}\)
2. **while** termination criterion is not met
3. **do** for each individual \(\Theta^i\)
4. \(\Theta \leftarrow \text{CreateOffspring}(\Theta^i)\)
5. **if** \(\Theta\) is better than \(\Theta^i\), replace \(\Theta^i\) with \(\Theta\)

In **CreateOffspring** we have chosen to use the DE/rand/1/bin [106] strategy as our starting point. There are several other strategies [106] but we have chosen one that we know is used in other studies [56]. We do not believe that our choice alters the results in our study as our observations should generalize to the other recombination strategies. The below code illustrates the DE/rand/1/bin strategy with a crossover rate \(CR\) and a scaling factor \(F\). The procedure **Random()** returns a random number between 0 and 1.

**CreateOffspring** \((\Theta^i)\)

1. Randomly select parents \(\Theta^{(1)}, \Theta^{(2)}\) and \(\Theta^{(3)}\)
2. Initialize empty offspring \(\Theta\)
3. Let \(j\) be a random number between 1 and \(n\).
4. **for** \(k \leftarrow 1, \ldots, n\)
5. **do if** **Random()** \(\leq CR\) or \(k = j\)
6. **then** \(\theta_k \leftarrow \theta_k^{(1)} + F \cdot (\theta_k^{(2)} - \theta_k^{(3)})\)
7. **else** \(\theta_k \leftarrow \theta_k^i\)
8. **return** \(\Theta\)

### 13.3 Angles

In the following we will be studying two angles \(\theta\) and \(\theta'\), both in the interval \([0, 2\pi)\). The angle difference between these is the length of the shortest arc between them on the unit circle and the numerical difference is the number \(\theta - \theta'\). In this paper, a positive numerical difference is illustrated by a counterclockwise oriented arc as illustrated on Figure 13.1.

Our concern with **CreateOffspring** is that it adds the numerical difference between the parent \(\theta^{(2)}\) and \(\theta^{(3)}\) to the parent \(\theta^{(1)}\). This procedure does not take into account the fact, that angles with a large numerical difference can actually have a small angle difference as can be seen on Figure 13.1.

The problem lies in the term \((\theta_k^{(2)} - \theta_k^{(3)})\) from **CreateOffspring**. We therefore propose two alternatives for this term, which take the fact that we are manipulating angles into account. The first approach utilizes the fact that if two angles have an absolute numerical difference \(|\theta - \theta'|\) that is larger than \(\pi\) as seen on Figure 13.1, we can define two numbers with an absolute value less than \(\pi\) by \(\pm(2\pi - |\theta - \theta'|)\) as illustrated in Figure 13.2. Our first strategy **RandomSignDifference** picks one of these two numbers with equal probability.
13.3. Angles

\[ \theta - \theta' > \pi \quad \theta - \theta' < -\pi \]

Figure 13.1: Two examples of angles with a small angle difference but with a large numerical difference.

\[ \frac{\theta - \theta'}{|\theta - \theta'|} = \frac{2\pi - (\theta - \theta')}{|\theta - \theta'|} \]

Figure 13.2: Two angles gives rise to two numbers that are numerically less than \( \pi \).

**RandomSignDifference(\( \theta, \theta' \))**

1. \( d \leftarrow |\theta - \theta'| \)
2. \( \text{if} \; d > \pi \)
3. \( \text{then} \; d \leftarrow 2\pi - d \)
4. \( \text{if} \; U_R(0, 1) < 0.5 \)
5. \( \text{then return} \; d \)
6. \( \text{else return} \; -d \)

We can rewrite line 6 of **CreateOffspring** to

\[ \theta_k \leftarrow \theta_k^{(1)} + F \cdot \text{RandomSignDifference}(\theta_k^{(2)}, \theta_k^{(3)}) \]

In the **RandomSignDifference** approach, the sign of the difference between the two sets of angles \( \Theta^{(2)} \) and \( \Theta^{(3)} \) is calculated in a random matter across the set of angles. An interesting question is if this is a good strategy, or if we should stick to one strategy across all the pairs of angles \( \theta_k^{(2)}, \theta_k^{(3)} \) in the two sets.

Let us define the **ConsistentSignDifference** between two angles \( \theta \) and \( \theta' \) as the numerical shortest distance between \( \theta \) and \( \theta' \), from \( \theta' \) to \( \theta \). As an example, examine the figures below.
Chapter 13. Recombining angles in Differential Evolution

ConsistentSignDifference($\theta, \theta'$)

Figure 13.3: The cases to consider when faced with two angles that we wish to recombine. Dashed lines indicate $\theta - \theta'$ and full lines indicate ConsistentSignDifference($\theta, \theta'$)

ConsistentSignDifference($\theta, \theta'$)

1. $d \leftarrow \theta - \theta'$
2. if $\theta' < \theta$
3. then if $d \leq \pi$
4. then return $d$
5. else return $d - 2\pi$
6. else if $d \geq -\pi$
7. then return $d$
8. else return $d + 2\pi$

It is worth noticing that we assume that the angles $\theta$ and $\theta'$ are in the interval $[0, 2\pi)$ which can easily be guaranteed by our DE implementation, by adding $2\pi$ to angles below zero, and by subtracting $2\pi$ from angles larger than $2\pi$. 
13.4 The Folding Rule Problem

We have constructed two simple optimization problems to evaluate our strategies. The arguments for constructing simple evaluation problems instead of evaluating on real world problems are that the problems are easy to understand and that the optimal values are easily calculated and therefore can be compared against.

Our aim has been to construct evaluation problems that are well defined for any number of angles and which are not separable. The resulting problem is best described by an everyday object, namely a folding rule. If we unfold a rule completely, the result will be a line along the \( x \)-axis.

Each joint of the folding rule correspond to an angle. We can adjust these angles to create a conformation of the folding rule.

If we remove the wood from the folding rule but retain the metal joints, we are left with a set of points

\[
\cdot \quad \cdot \\
\cdot 
\]

The Folding Rule Problem is, given such a list of points, to identify a conformation of a folding rule that best covers these.

More formally, given a set of \( n + 1 \) target points \( T = \{t_0, \ldots, t_n\} \), we wish to identify a list of \( n \) angles \( \Theta = \{\theta_1, \ldots, \theta_n\} \) which minimizes

\[
\sum_{t \in T} \min_{p \in P_\Theta} (|t - p|)
\]

where \( P_\Theta \) is the set of points obtained by folding up a virtual folding rule with unit length segments as illustrated on Figure 13.4 (a). This is done by fixing \( p_0 \) in \((0, 0)\) and choosing \( p_i \) as

\[
p_i = p_{i-1} + (\cos(\sum_{j=1}^{i} \theta_j), \sin(\sum_{j=1}^{i} \theta_j)).
\]
Chapter 13. Recombining angles in Differential Evolution

In a related problem, we demand the point $t_{i}$ to be matched with $p_{i}$ and we therefore seek to minimize the quantity

$$\sum_{i=0}^{n} |t_{i} - p_{i}|$$

as illustrated in Figure 13.4 (b). This related problem we name the Restricted Folding Rule Problem.

Our constructed problems have two properties: (1) if the target points are generated by a folding rule we know the optimal fitness (zero) and (2) changing an angle $\theta_{i}$ changes the points $p_{i}, \ldots, p_{n}$. The first property makes it easier to set goals for our tests and the second property ensures that the variables in the problems are not separable.

13.5 Experimental setup

We have compared the original unmodified recombination strategy (ORIGINAL) with the two proposed alternatives, RANDOMSIGNDIFFERENCE and CONSISTENTSIGNDIFFERENCE on our two folding rule problems.

We have performed tests for a varying number of angles $n$; for each $n$ we have performed 1,000 runs where the target points are generated by a rule and with random populations of 50 members. The three strategies are used in our DE implementation with a scaling factor $F$ of 0.5 and a crossover rate of 0.75. Both the Folding Rule Problem and the Restricted Folding Rule Problem are used as benchmarks in the tests.

An optimization is terminated when the best individual reaches a fitness below $10^{-3}$ (success) or when the number of fitness evaluations exceeds 10,000 (failure). We record the AES (Average Evaluations pr. Success) and the percent of successful runs.

We have furthermore tested our three recombination strategies on a simple energy minimization problem from computational chemistry, namely identifying the angles that minimize the energy of the alanine dipeptide [49]. We have used the same parameters as the other experiments on the structure which
13.6. Results

For clarity we present our results as shaded matrices where the shading of an entry illustrates how many successful runs were obtained or how many evaluations were required for success.

In Figure 13.5 we have illustrated the percentage of successful runs on our two folding rule problems; an omitted box means that no successful runs were observed. CONSISTENT-SIGNDIFFERENCE outperforms the two other strategies in all but two runs where it is beaten by one of the other strategies by exactly one point. The most impressive difference is where \( n \) is 6 in the Restricted Folding Rule Problem where the CONSISTENT-SIGNDIFFERENCE strategy has a success rate of 53% as opposed to the ORIGINAL strategy with only 22%.

When we look at the AES (Figure 13.6) we see that the CONSISTENT-SIGNDIFFERENCE has a lower number of fitness evaluations than the other two methods. The improvement is, however, not as impressive as one would hope. Tables with the exact numbers can be obtained at

\[ \text{http://daimi.au.dk/~tgk/}. \]

The alanine dipeptide experiments are summarized in Table 13.1 and indicate that the altered recombination strategies have less impact on problems from computational chemistry. As can be seen from the table we actually perform more evaluations before reaching the desired goal when using the two altered
Chapter 13. Recombining angles in Differential Evolution

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Success</th>
<th>AES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>12%</td>
<td>4,744</td>
</tr>
<tr>
<td>RandomSign</td>
<td>12%</td>
<td>4,810</td>
</tr>
<tr>
<td>Consistent</td>
<td>12%</td>
<td>5,176</td>
</tr>
</tbody>
</table>

Table 13.1: Summary of alanine dipeptide experiments.

strategies and this experiment is therefore contradictory to the previous, indicating that the original DE strategy is more robust across problems with more complex fitness landscapes.

13.7 Conclusion

To sum up, our experiments indicate that the choice of recombination strategy has a large effect on the percentage of successful runs in simple cases, but that it has little effect on the number of fitness evaluations in these. Our experiments furthermore illustrate that the choice of optimization problem can have a large impact on the observed difference between the recombination strategies and in some cases it can lead to contradictory conclusions. We cannot confirm nor refute our hypothesis that angles should not be handled naïvely.

Our studies poses two interesting questions: (1) is testing new ideas on sandbox problems a valid evaluation method and (2) is over engineering functions to specific optimization scenarios a problem in the field. It is our hope that others in the field will investigate these two interesting questions.
Bibliography


143


