Universidade do Minho
Escola de Engenharia
Departamento de Informática

Marta Sofia Saraiva Oliveira

Flexible Molecular Alignment

An Industrial case study on
Quantum algorithmic techniques

July 2020

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Marta Sofia Saraiva Oliveira

Flexible Molecular Alignment

**An Industrial case study on
Quantum algorithmic techniques**

Master dissertation
Master Degree in Physics Engineering

Dissertation supervised by
Prof. Luís Soares Barbosa

July 2020

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ACKNOWLEDGEMENTS

Starting from the people that firstly made an impact on me and, hence, this dissertation, I would like to thank all my close family. In my case it includes parents, sisters, grandparents, cousins, aunts and uncles.

Secondly, I would like to thank all my dearest friends and colleagues that made an impact in my life, in way or another, and always believed in me. A special appreciation is required to Vánia, who read my dissertation from beginning to end, to Joana, who always encouraged me, even in the most difficult time, and to Ricardo, my sweetheart and greatest admirer. With the same gratitude and special recognition, I have my colleagues and friends, Vítor and Michael, who accompanied me more closely in my last three years and from whom I will always have great memories and appreciation. To the last, I would also like to thank their inputs and patience during the making of this dissertation and this degree.

In a more academic ground, I would like to thank Carlos Tavares, without whom I would not be able to develop a very important section in this dissertation. Moreover, I would also like to thank professors Ana Jacinta Soares, Fernanda Costa and Alexandre Madeira for their advices and feedbacks in the early stage of the dissertation. In another ground, a special thank you to doctors Marco Neves and Nuno Palma, from BIAL, is also required. Finally, I would like to acknowledge the help of Ana Neri and the orientation of professor Luís Barbosa in the making of this dissertation.

In a much more personal and important note, I would also like to thank Sérgio Ferreira for helping me keep sane during this last year and for teaching me how to deal with myself and my anxiety.

ABSTRACT

Flexible molecular alignment is a complex and challenging problem in the area of Medicinal Chemistry. The current approach to this problem does not test all possible alignments, but makes a previous analysis of all the variables and chooses the ones with potentially greater impact in the posterior alignment. This procedure can lead to wrong "best alignments" since not every data is considered.

Quantum computation, due to its natural parallelism, may improve algorithmic solutions for this kind of problems because it may test and/or simulate all possible solutions in an execution cycle.

As a case study proposed by BIAL and in collaboration with IBM, the main goal of this dissertation was to study and create quantum algorithms able to refactor the problem of molecular alignment in the new setting of quantum computation. Additionally, the comparison between both classical and quantum solutions was defined as a subsequent goal.

During this dissertation and due to its complexity, in order to produce a practical solution to this problem, we resorted to a manageable number of conformations per molecule, revisited the classical solution and elaborated a corresponding quantum algorithm. Such algorithm was then tested in both a quantum simulator and a real device.

Despite the privileged collaboration with IBM, the quantum simulations were not produced in viable time, making them impractical for industry applications. Nonetheless, taking in consideration the current point of development of quantum hardware, the suggested solutions still has potential for the future.

RESUMO

O alinhamento de moléculas flexíveis é um problema complexo na área de Química Medicinal, onde, mesmo hoje em dia, é um desafio encontrar uma solução. A atual abordagem para este problema não testa todos os possíveis alinhamentos. Em vez disso, realiza uma análise prévia de todas as variáveis e escolhe aquelas com maior potencial de impacto no posterior alinhamento. Este procedimento pode levar a falsos "melhores alinhamentos" visto que nem todos os dados são considerados.

A computação quântica, devido ao seu natural paralelismo, pode melhorar as soluções algorítmicas deste tipo de problemas visto que poderá testar e/ou simular todas as possíveis soluções num ciclo de execução.

Partindo de um caso de estudo proposto pela BIAL, e em colaboração com a IBM, o objetivo principal desta dissertação foi estudar e criar algoritmos quânticos capazes reformular no contexto de computação quântica o problema de alinhamento de moléculas. Adicionalmente, e como objetivo subsequente, foi prevista a comparação entre os algoritmos clássicos e quânticos.

Durante esta dissertação e devido à sua complexidade, de modo a produzir uma solução prática para este problema, foi utilizado um número tratável de conformações por molécula, revisitada a soluçao clássica e desenvolvido um algoritmo quântico correspondente. Tal algoritmo foi depois testado tanto num simulador quântico como num dispositivo real.

Apesar da colaboração previligiada com a IBM, as simulações quânticas não foram produzidas em tempo viável, tornando-as impraticáveis para aplicações industriais. Não obstante, tendo em consideração o ponto atual de desenvolvimento dos dispositivos quânticos, as soluções propostas terão potencial para o futuro.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

1.1 MEDICINAL CHEMISTRY AND DRUG DESIGN

Drug discovery and development is generally done in the commercial rather than academic contexts, mainly because of the money and time-consuming processes involved (Verkman, 2004). In these processes, compounds with activity against a specified target or function are identified, evaluated, and optimized for clinical applications. The average preclinical costs, from lead identification to *investigational new drug* (IND) filing, illustrated in Figure 1, has been estimated to be as high as US$4 million per compound, and to take up to 3 years (Verkman, 2004).

The iterative step in Figure 1 is the optimization of each potential lead by medicinal chemistry, where the main goal is to generate structural analogs of lead compounds that are optimized in their potency, specificity, and pharmacological properties for entry into clinical trials. Generally, this involves iterative rounds of synthetic organic chemistry and compound evaluation that can take from months to years (Verkman, 2004). As illustrated in Figure 1, it starts by the collection and analyses of *Structure-activity relationship* (SAR) data.

Three major computational methods have been used for lead optimization:

- **Rational structural** methods, which generally use a high-resolution structure of the target to direct the synthesis of new analogs.
- **Pharmacophore** methods, which involves the definition of the minimal unit that leads to activity (usually a combination of hydrogen bond donor/acceptors, hydrophobic groups, and other functional groups) in a three-dimensional space. The consensus pharmacophore is then used to examine the allowed placement of groups in a set of candidate compounds.
- ***quantitative structure-activity relationship* (QSAR) models**, which relates calculated physicochemical properties of molecules to activity, rather than strictly structural properties.

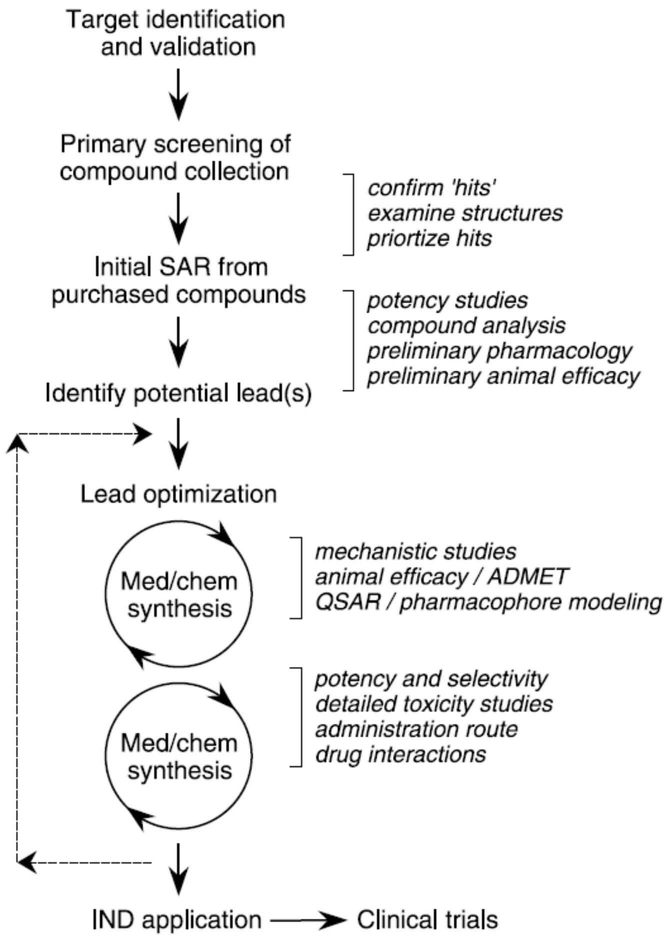


Figure 1: Strategy for preclinical drug discovery. See text for details and glossary for other definitions. *Figure 1. of Verkman (2004).*

The first method often involves the generation of a very large in silico library of potential derivatives and use of a computational docking method to select derivatives that may interact with the target on the basis of shape complementarities and charge placement. While intellectually appealing, there have been few examples of success when used by itself. On the other hand, pharmacophore analysis can be carried out without structural information. It is most useful in identifying new compounds with a desired activity based on a three-dimensional similarity to early leads. At last, a QSAR model relates more than just structural properties, but unfortunately it requires a set of structurally related compounds with a wide range of activities, ideally a 1000-fold variation in activity, which is often a difficult requirement to meet. (Verkman, 2004)

Since it can be difficult to visualize a drug design process only based on written words, it is relevant to take a look at Figure 2. The first image is the pharmacophore and structural properties of the target, below are represented the methods on which the drug design can

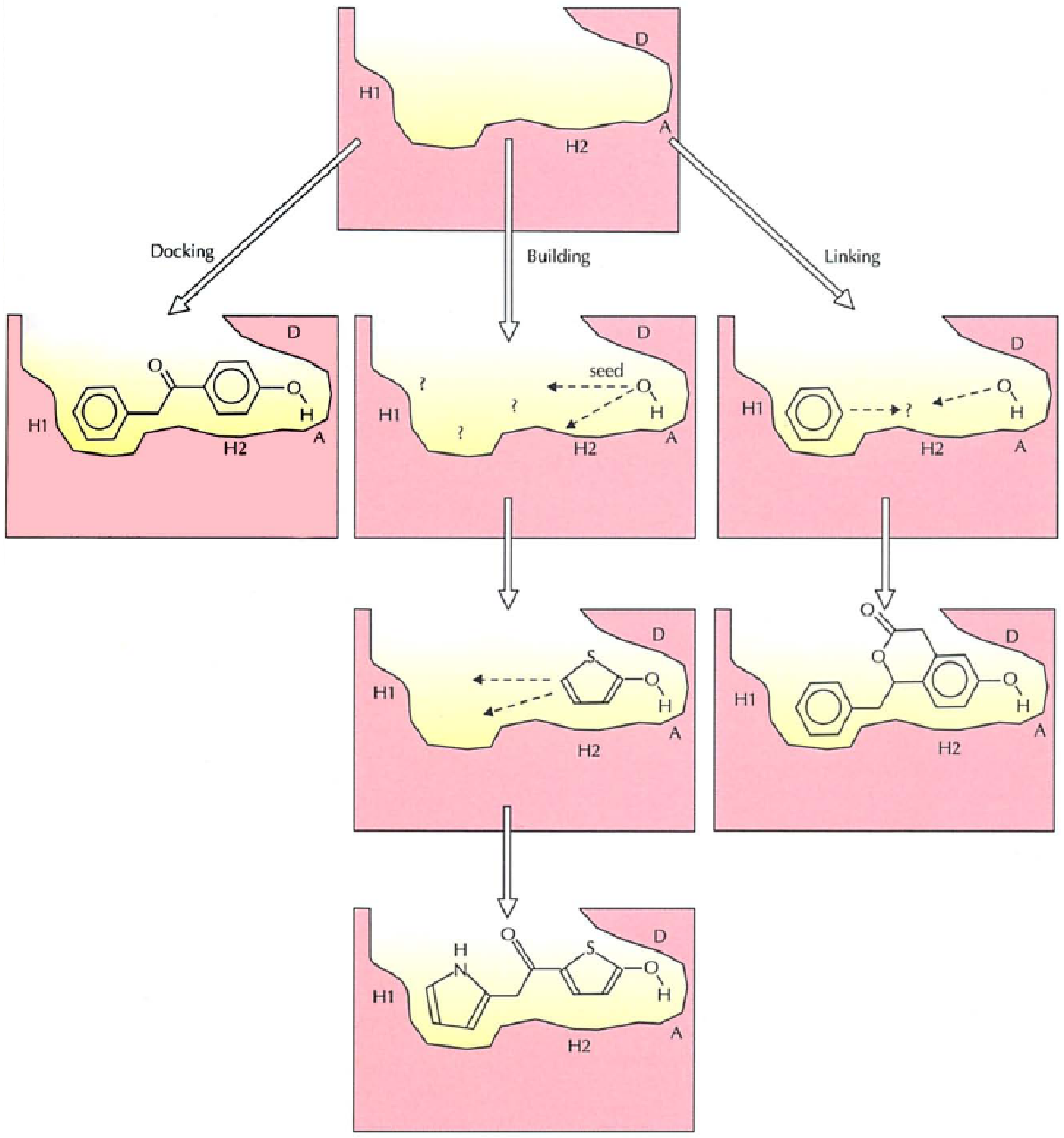


Figure 2: Methods for protein structure-based inhibitor drug design. All methods first characterized the target site in terms of shape and presence of specific surface properties, e.g. hidrophobic sites (H), hydrogen bond donors (D) and acceptors (A). Subsquently, docking, building or linking algorithms are applied. In docking, molecules are retrivied from a huge database and evaluated for complementary to the target site. Building starts from a highly complementary fragment, either found by docking or known from a previous protein-ligand structure. This fragment, called the 'seed', is appended with a myriad of different fragments, which each intern can be substituted further. This combinatorial explosion, which is also the all mark of the linking method, is contained by pruning techniques. In linking, highly complementary fragments are linked into one molecule. *Figure 2. of Verlinde and Hol (1994).*

be based. Throughout this work it was considered the docking method as the method used for drug design. In the lead optimization, and because this work is a study of this method, it was also only considered the pharmacophore modeling method.

Back in 1994, and as recalled in Verlinde and Hol (1994), it was usually only tested one ligand conformation in the docking method because of the inherent computational intractability. Nowadays, more than one conformation per ligand are tested because the current computational power is much higher. However, when considering an average organic molecule (which have eight rotational bonds), and assuming that 30 increments in dihedral angles define different conformations, then for just one average molecule, about 430 million conformations have to be examined (Verlinde and Hol, 1994). As one can imagine, even with the present computational power, the docking method becomes quickly impracticable when considering all conformations from every ligands. Some docking/alignment algorithms for entire molecules make an attempt to address the issue of ligand flexibility but the CPU-time requirements for docking/aligning large databases becomes unacceptable. As a more time-sustainable approach, instead of testing enormous numbers of conformations, one can investigate whether the conformation of a ligand can be altered to satisfy the constrains of a protein-binding site. Such algorithms test on simple 3-5 point pharmacophore, instead of a full three-dimensional protein context. Despite the huge reduction in information, this remains a highly demanding computational problem.

Another potential problem in drug design is the flexibility of the target, as is the case of proteins. A typical example, given in Verlinde and Hol (1994), is the triosephosphate isomerase where the so-called flexible loop of the enzyme is closed in the presence of inhibitors and open otherwise. The possibility of unexpected conformational changes of the protein upon ligand binding is one of the reasons that experimental verification of predicted binding modes of new ligands is a key step in a cyclic structured-based inhibitor drug design process.

Ideally, all possible conformations of the ligands in the database should be checked *versus* all possible conformations of the protein molecule. However, because it becomes an impracticable problem, one of the current key issues in this classical algorithm approach is to determine whether a sufficient number of ligand conformations from a sufficiently large reservoir of small compounds are tested for their fit *versus* a sufficiently large number of conformations of the protein. The continuous increase in computer power will allow the testing of more conformations but will never be sufficient to test all information. This challenge is the main aspect this dissertation aims to tackle, reduce the computational time required to test a large number of alignments in parallel in order to make feasible this classically intractable problem.

On these grounds and due to some other challenges, the number of approved drugs for the market has decreased in recent years, which entails the need for more effective methods to discover a higher number of successful compound candidates (Verkman, 2004; Dahms, 2017). It has been argued that drug discovery in an academic setting is urgently needed to explore alternative paradigms in the currently very inefficient drug discovery process (Verkman, 2004). Here, novel computational tools which could speed up drug developments and lead to promising compounds could be produced. An academic goal given as an example in Verkman (2004)'s work, is the identification of reasonably potent and specific reagents for protein inhibition/activation in animal models and the discovery and evaluation of drug candidates with the intention of partnering with the industry for further development.

As the most recent pandemic emerged, *COVID*-19, and the world felt its effects, it quickly became evident the importance of developing new drugs with quick and reliable methods.

This dissertation has in sight the development of a new (quantum) paradigm in the field of bioinformatics, more specifically, drug design.

1.2 QUANTUM COMPUTATION

Quantum computation is based on quantum physics, which explains reality at a very small scale (atomic and subatomic scale). The key aspect that makes its laws of interest is the ability of matter, at this scale, to be in a superposition of states[1]. As an example, consider an electron with two distinct possible configurations (spins), up (1) and down (0). In a classical computer, this electron represents a bit since it can be in two possible states, up **or** down (1 or 0). In an atomic scale, this electron is a qubit and it can also be in a superposition of states. It can be up (1), down (0), or simultaneously up **and** down (1 and 0). It is important to mention that a superposition is a linear combination of states, whose coefficients are related to the probability of being in either state. Theoretically, the qubit can **represent** an infinite number of different states but can only **be**, physically, in one (Wright, 2015). This is a *fundamental postulate* of quantum mechanics, which states that an object in superposition will collapse to a specific state, within a known probability, when measured (Nielsen and Chuang, 2011). The bit, on its turn, can only **represent and be** in one state. Of course, and due to practicalities, these number of representable different states are finite and not infinite because one can only have a finite certainty to distinguish states (Deconinck and Terhal, 2010). Refer to Figure 3 for a visual illustration and mathematical representation.

Computationally, superposition is relevant because it allows an algorithm to be naturally parallel. With a two bit register, a classic algorithm can only operate over one possible state,

1 A more interested reader may want to check DiVincenzo (1995) or, in more detail, Nielsen and Chuang (2011), for more information on this subject.

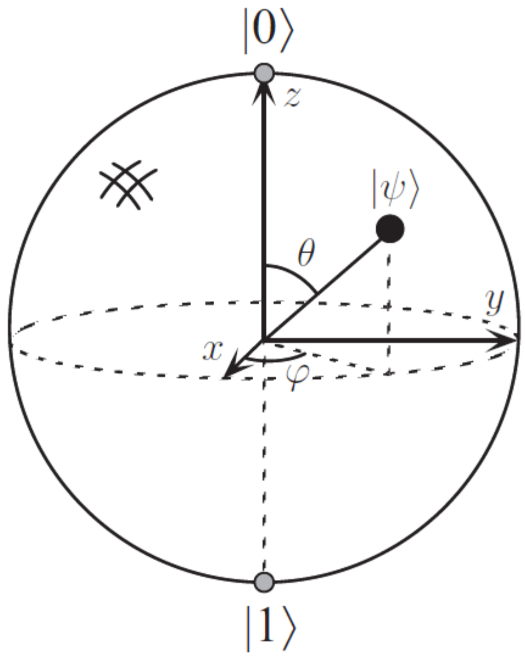


Figure 3: Bloch sphere representation of a qubit. It's state, ψ, is given by $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = cos\frac{\theta}{2}|0\rangle + e^{i\varphi}sin\frac{\theta}{2}|1\rangle$ where $|\alpha|^2$ and $|\beta|^2$ yields the probability to be in state $|0\rangle$ or $|1\rangle$, respectively. A classical bit can only either have $|\alpha|^2 = 1$ **or** $|\beta|^2 = 1$. *Figure 1.3. of Nielsen and Chuang (2011).*

while with two qubits an algorithm can operate over four possible states. To consider the same number of states with bits, it is required an exponential higher number of the latter, as depicted in Figure 4.

To conclude, quantum algorithms due to their natural parallelism can be beneficial to handle problems involving the analysis of a great volume of information, which would require an extremely large number of bits. With quantum algorithms, this number is exponentially reduced. Another relevant aspect to the flexible molecular alignment problem, in particular, is that most of industrially used software are optimized to a level that a function can converge to a local minimum and mistakenly give a wrong answer due to optimization processes that may lead the software to consider only part of the data and not all information. Since a quantum algorithm would be able to take into account more data, it could hopefully converge to a global minimum, rather then to a local one.

1.3 SUMMARY

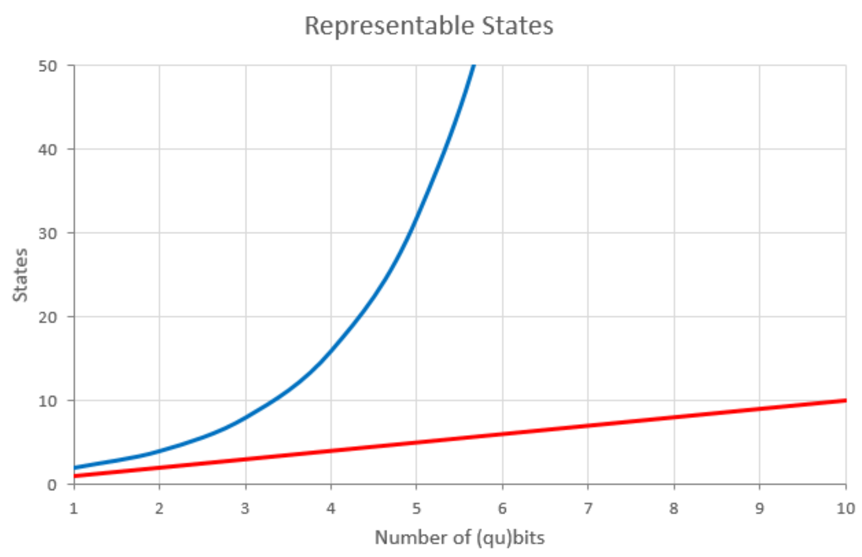


Figure 4: Number of representable states according to the number of bits (in red) and qubits (in blue). To represent 10 states it is required less than 4 qubits but 10 bits.

The aim of this dissertation is to study a real problem in industry, introduce a quantum approach as its solution, and analyze its possible impact and viability in the future. With this intuit, contact was established between University of Minho and BIAL (a major Portuguese pharmaceutical company) and a case study of the area of medicinal chemistry and bioinformatics appeared. The case study proposed by BIAL was the alignment of flexible molecules, as the title of this dissertation mentions, and subsequent discovery of pharmacophores. The goal, in more detail, was to find a quantum algorithm to align flexible molecules, to test it using a quantum computer and lastly, analyze the results and their viability compared to the current classical methods. Due to the already establish collaboration between the University and IBM (the international company with the first-ever commercially available quantum computing system (IBM, 2019)), it was possible to simulate (test the algorithm) in a **real** quantum device.

Along this period, I was also involved in another work on quantum chemistry. This piece of research was a case study on quantum simulation for two distinct molecules, hydrogen (H_2) and lithium hydride (LiH), at an actual commercially available quantum computer, the IBM Q. As a result, a paper was presented in a conference and another is waiting to be published in a journal. (Tavares et al.)

This dissertation is organized in eight sections. The opening sections introduce and contextualize the problem. The succeeding sections explain and breaks down the classic and quantum solutions. And the final sections exhibit the results, conclusions and future suggestions and approaches. After the bibliography, a glossary of brief definitions on more technical words and expressions is presented.

In more detail, the first and current section gives a brief introduction to both drug design and quantum computation. The following section provides a background on pharmacophores and on the interconnected flexible molecules and respective alignment. Next, the third section gives a brief resume of the pre-existing algorithms for molecular alignment. The fourth section, concerns the problem data, definition and approaches taken. The development of a classic solution is reported in section five, and a quantum solution in section six. Results and comparisons are presented in section seven and finally section eight concludes and makes some considerations on future work.

2 BACKGROUND

In broad terms, the topic of interest is related to computer-aided design of drugs. Therefore, it is important to give the proper background, as an introduction to medicinal chemistry, so that the reader can understand the *hows* and *whys* of this field.

2.1 PHARMACOPHORES

There are several definitions of **pharmacophores**. In plain language, a pharmacophore is the spatial arrangement of functional groups essential for biological activity; it is a three-dimensional pattern that emerges from a set of biologically active molecules (Drie, 2007; Neves, 2018). Another way to interpret a pharmacophore is as an abstract description of molecular features which are necessary for molecular recognition of a ligand by a biological macromolecule. It is important to emphasize that **a pharmacophore is not a real molecule**, but an abstract concept (Qing et al., 2014). Pharmacophore features include hydrophobic moieties, aromatic rings, hydrogen bond acceptors or donors, cations and anions (Karaman, 2016; Homeyer, 2007). Figure 5 depicts an example of a pharmacophore based on the respective molecule.

As stated in the IUPAC definition of the term:

Definition 2.1 *(Wermuth et al., 1998) Pharmacophore is the ensemble of steric and electronic features that is necessary to ensure the optimal supramolecular interactions with a specific biological target structure and to trigger (or to block) its biological response. The pharmacophore can be considered as the largest common denominator shared by a set of active molecules.*

Now that it is clear what pharmacophore means, it is important to know how a **pharmacophore model** is established. According to Yang (2010), it can either be established in a **structure-based** manner, by probing possible interaction points between the macromolecular target and ligands; or in a **ligand-based** manner, by superposing a set of active molecules and extracting common chemical features that are essential for their bio activity.

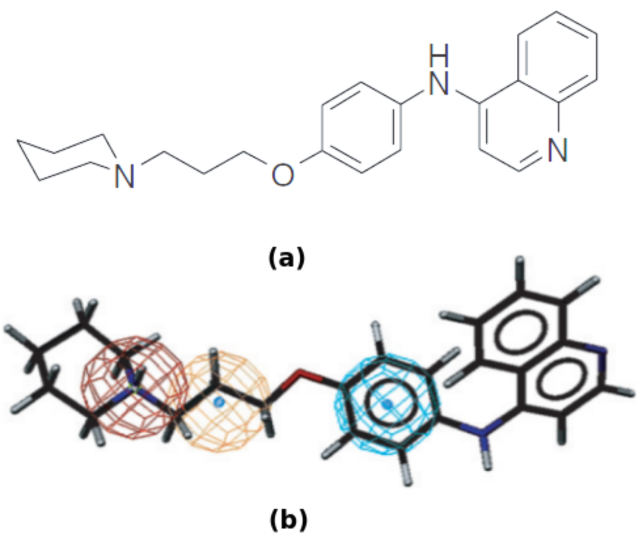


Figure 5: (a) Molecular Structure of hH_3R antagonist (b) Pharmacophoric features defined based on compound (a). (Red sphere = any positively charged element, orange sphere = aromatic or hydrophobic group, cyan sphere = aromatic ring). *Figure 28.7 and 28.9 of Sippl (2008).*

The **structure-based pharmacophore modeling** works directly with the structure of a macromolecular target or a macromolecule-ligand complex. The protocol for this technique involves an analysis of the complementary chemical features of the active site and their spatial relationships, and a subsequent pharmacophore model assembly with selected features. Furthermore, it is reasonable to divide this strategy in two categories:

- macromolecule-ligand-complex-based, where the structure of both the target and ligand are known, and
- macromolecule-based, where only the structure of the receptor molecule is known.

The first method builds a pharmacophore model by locating the ligand-binding site of the macromolecular target and then determining the principal interaction points between the ligands and macromolecule, whereas the second constructs this model by analyzing the chemical properties of the binding site of interest, which generates an interaction map converted posteriorly into pharmacophore features (Yang, 2010; Qing et al., 2014).

The macromolecule-ligand-complex-based approach is bound to the structure of the macromolecule-ligand-complex, as the name suggests. The implication is an impossibility of applying this method to cases where no compounds targeting the binding site of interest are known. Although this can be overcome with the macromolecule-based approach, that technique has its limitations as well. For example, the derived interaction maps generally consist of a large number of unprioritized catalyst features, which complicates its application in tasks such as database searches.

However, both strategies face a frequent problem related to the high number of chemical features that are identified for a specific binding site of the macromolecular target. A pharmacophore model composed of too many chemical features (*i.e.* more than 7) is not suitable for practical applications, making it necessary to select a limit number (*i.e.* between 3 and 7) of these features, which is not an easy task in most of the cases (Yang, 2010).

On the other hand, **ligand-based pharmacophore modeling** is an essential computational strategy for facilitating drug discovery in the absence of a macromolecular target structure (Yang, 2010). In this approach, only some chemical features are required to make a new drug model that "suits" the target. The freedom from the target structure in this modeling technique is of great use because, recurrently, the macromolecular target structure is unknown (Homeyer, 2007; Qing et al., 2014).

The above mentioned chemical features are found due to their presence in essential interactions between a set of ligands and a specific macromolecular target. There are two main steps necessary to find them:

1. Create the conformational space for each ligand in the set to represent conformational flexibility of ligands
2. **Align the multiple ligands in the set**

These steps allow determining the essential common chemical features to construct the pharmacophore model. The two steps mentioned above, *handling conformational flexibility* of ligands and conducting *molecular alignment*, are both the key techniques and the main difficulties in ligand-based pharmacophore modeling (Yang, 2010; Homeyer, 2007).

The ligand-based pharmacophore modeling approach is the reason for the subject of study of this dissertation, flexible molecular alignment. The following sub section will introduce the topic of *flexible molecules*.

2.2 FLEXIBLE MOLECULES

As it was previously said, alignment of molecules is one of the main difficulties in ligand-based pharmacophore modeling, the technique used in BIAL's computer-aided drug design software. The alignment of *flexible* molecules is a problem of even higher proportions. But to understand why this is such a hard problem, one may need to understand what is a flexible molecule and how they can be superposed. This will be discussed in the next sub section.

To clarify, molecules are structures composed of two or more bonded atoms. A flexible molecule is a molecule where these bonds have a certain degree of freedom, *i.e.* the bonds can move (by rotation) within a certain physically-allowed degree (A. Y. Grosberg, 2011; MacDowell, 2003).

Each possible position of a given flexible molecule, based solely on rotations about single bonds, is named a **conformation**. According to IUPAC Recommendations (Moss, 1996), a conformation is formaly defined as:

Definition 2.2 *The spatial arrangement of the atoms affording distinction between stereoisomers which can be interconverted by rotations about formally single bonds.*

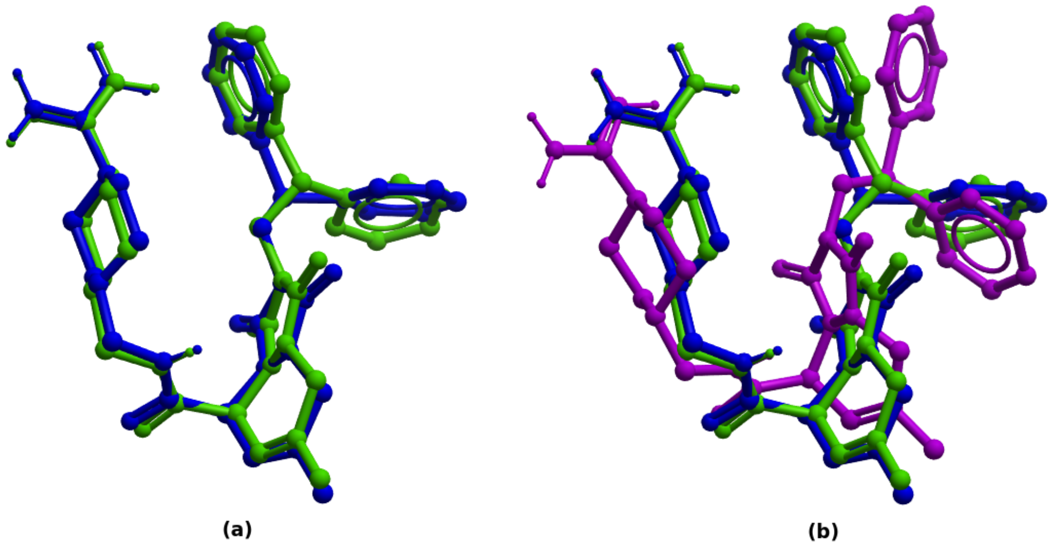


Figure 6: Superposition of **conformations of the same molecule** to visualize the differences between them. (a) 2 conformations (b) 3 conformations. Molecule: 1*C*4*V* from PDB (Protein Data Bank). *Images obtained through Molsoft's software, ICM-Pro.*

It is important to clarify that by the definitions of *stereoisomers*, constitutional isomers[1] and conformers, one may think that they stand for the same, despite they only have one similarity. All of these terms stand for compounds that have the same molecular formula but different atoms arrangements. Therefore, all of them are actual isomers. However, from this point forward, they have different meanings. If the isomers have the atoms connected in the same order, they are stereoisomers; if not, they are structural or constitutional isomers. Furthermore, if they are stereoisomers compounds and they can be interconverted by rotation about single bonds, they are conformational isomers or conformers, and otherwise, configurational isomers (Hunt, 2009b).

1 In the literature (Gunawardena, 2019; Hunt, 2009a), structural isomers appear as a synonymous of constitutional isomers. The International Union of Pure and Applied Chemistry (IUPAC) recommends that the term ”structural” should be abandoned when applied to constitutional isomers (Solomons and Fryhle, 2009).

When a conformation corresponds to a potential energy minimum, it is said to be a conformational isomer or conformer. Hence, all conformers have a correlated conformation but not all conformations represent a conformer (Book, 2014). To a better comprehension, check Hunt (2009b)'s diagram that summarizes these different types of isomers in a very intuitive way.

Based on the definitions above, it is now known that it is possible to have other positions of a given flexible molecule regarding other arrangements that deal with other than only rotations about single bonds. But because the whole purpose, in this work, is to align (superpose) flexible molecules to posteriorly make a pharmacophore model (and these only deal with conformations), those arrangements will be the only ones of interest. Henceforward, "all positions" will just mean the ones of interest.

Focusing only in conformations and conformers, it is clear that the number of different conformers per molecule depends on the number of single bonds. A simple rule of thumb, as stated by Smith (2010), is that every single bond multiplies the number of possible conformers by three. Thus, a molecule with one single bond has three conformers, which can be translated into an exponential growth in the order of 3^n (where n is the number of single bonds in a molecule). Consequently, even a reasonably sized organic molecule can exhibit hundreds to thousands of possible conformations (Smith, 2010). The alignment of flexible molecules without any kind of compromise in the conformational space (compromise that can mislead to a false *best fit*) may involve testing the alignment between all conformations from each molecule. The act of searching the whole conformational space is called *systematic search*. Naturally this can turn into a problem of difficult resolution since superposing hundreds of flexible molecules would involve the comparison of billions of possible "solutions" (Beusen and Marshall, 2000; Neves, 2018).

As mentioned, conformational analysis, *i.e.* handling the conformational flexibility of molecules, is one of the most challenging tasks in ligand-based pharmacophore modeling since the active conformations of the molecules are usually unknown (Homeyer, 2007). If this problem was solved, the necessity of testing all possible conformations of a molecule would be cross off the problem and one would only test the exact conformations of interest from each molecule, greatly reducing the number of possible solutions. This is then highly related to flexible molecular alignment and pharmacophore modeling (Diller and Merz, 2002) but, because it is, by itself, a complex problem, it will not be further explained in this text.

On the other hand, once classified what is a *flexible molecule* and all its implications, the next step is to understand how *molecular alignment* is actually done regarding flexible molecules.

2.3 MOLECULAR ALIGNMENT

There are a **variety of approaches** to molecular alignment, which can be classified according to different points of view. Two examples of different perspectives: one based on the aspect of molecular similarity, and another on the treatment of conformational flexibility of the molecules. The former is based on which common characteristics an alignment is based, for example, atoms, pharmacophoric points, shape or by looking at similarities of fields of various physicochemical properties (e.g. electron densities, charge distributions, hydrophobicity or hydrogen-bonding) (Homeyer, 2007). Furthermore, the latter perspective is based on how the approaches deal with 3D structures, that is, if the molecules are handled as rigid or flexible entities. This is the point of view most relevant to the ”story”.

In this classification point of view, there are still some other ways to handle structures in addition to the strictly rigid or flexible entities. Some of the techniques try to introduce conformational flexibility in an indirect way, *i.e.* they compare sets of precomputed conformations for one molecule using conformation generation programs in advance, and afterwards perform a rigid body alignment of the generated conformations. This is the so-called **semiflexible approach**. Another technique that tries to bring flexibility into the search process **generates conformations on-the-fly** by applying different algorithms. One class of algorithms performs a systematic search in the conformational space while others use stochastic methods to generate conformations for a molecule. (Homeyer, 2007; Lemmen and Lengauer, 2000)

A disadvantage of the semiflexible approach is that it is often difficult to decide *a priori* on the number of conformations used for the subsequent alignment. Besides, only metastable and low energy conformations are considered. Bent conformations, such as observed in the transition state of a chemical reaction, can not be detected with such an approach, which can be insufficient to make a good conformational coverage. The advantage of on-the-fly flexing is that the computed conformations are not restricted to low-energy conformers. Even though it does not consider the whole conformational space, the disadvantage is that it is still more time consuming than to use precomputed low-energy conformers. (Homeyer, 2007; Lemmen and Lengauer, 2000)

To recap, there are **two ways to deal with the alignment of flexible molecules**. One handles the molecules as flexible entities throughout the process of alignment. The other superposes the various conformations from each molecule and treats the alignment itself as not one, but multiple rigid molecular alignments.

Ideally, all conformations of every molecule should be considered and all the overlays taken into account (Feher and Schmidt, 2000). But this never happens in either of the methods mentioned above. It is possible to conclude that there is no *perfect way* to align flexible molecules, there is always a disadvantage.

As the disadvantage of the *semiflexible approach* is essentially related to which and how many conformations one should use to achieve the best solution, one way to solve this issue would be to select a great number of conformations from each molecule. This is not currently done because it is very time consuming, and it would involve a great number of rigid molecular alignments. The only way to improve the method ended up with the same disadvantage encountered in the *on-the-fly* method, the **great computational time required**.

Due to the natural parallelism of quantum computation, the time required to execute the multiple rigid molecular alignments, on the *semiflexible approach*, could be potentially reduced. But first is required a review of the current classical methods available.

3

STATE OF ART

Throughout this section, several flexible molecular alignment methods will be quickly reviewed, majorly based on classical (non-quantum) mechanisms. Recently, no relevant studies regarding new techniques for flexible molecular alignment have been released. Therefore, it is important to emphasize that this revision is entirely based on methods, in most cases, with more than ten years.

Lemmen and Lengauer (2000) have made an extensive review of the literature regarding computational methods for the structural alignment of molecules back in 2000. In *Table 2* of the same article, there is a relevant overview of several approaches described in the literature until then.

Since there are two ways to deal with flexible molecular alignment, as discussed in section 2.3, there will be a division in this section. Two subsections will be about these two approaches, and a third will be about a rigid molecular alignment method based on quantum properties, which may be relevant for a future quantum approach.

3.1 SEMIFLEXIBLE MOLECULAR ALIGNMENT

As discussed in section 2.3, *semiflexible molecular alignment* introduces flexibility through the generation of pre-computed conformations and posterior enforcement of rigid molecular alignment. Therefore, it is relevant to get acquaintance with a few rigid molecular alignment algorithms. Due to the considerable number of algorithms in this field, only a selected few will be explained. To make it easier to understand the applications, the algorithms will be explained in context, as they appear in the literature about semiflexible molecular alignment.

Feher and Schmidt (2000) have created a method called *MultiSEAL*, an extension of the *steric and electrostatic alignment* (SEAL) method (Kearsley and Smith, 1990), that allows the overlay of multiple molecules and conformations. It makes a systematic study of possible alignments and gives information about conformational energies associated with a given overlay without any assumptions concerning initial alignment.

SEAL is a method developed to optimize the alignment of two rigid 3D structures using their atomic partial charges and steric volumes factors combined with a Monte Carlo[1] search procedure. The overlay is based on an alignment function that sums a set of atomic functions containing electronic terms (partial atomic charges) and spatial factors (van der Waals atomic radii). Possible alignments are identified by randomly rotating and translating structures with respect to the other and then minimizing the alignment function for each orientation. (Kearsley and Smith, 1990)

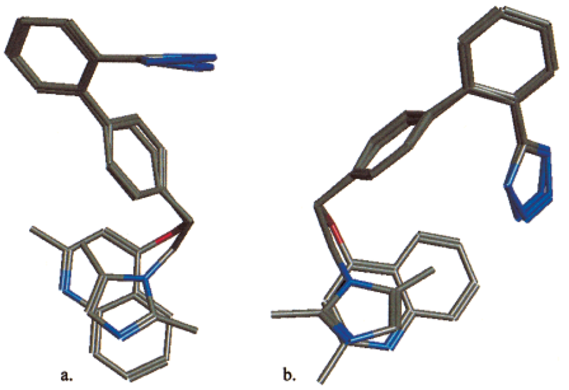


Figure 7: Best two alignments, using Feher and Schmidt (2000)'s work, of two angiotensin II antagonists. *Fig. 3 in (Feher and Schmidt, 2000).*

Since the method to assess the quality of a multi-molecule alignment, in MultiSEAL, is based on the sum of the scores of pairs of alignments, there is no guarantee that a good fit at an intermediate step will be able to contribute to a good overall fit at the end. Therefore, this is a main disadvantage of the MultiSEAL program. Another limitation of MultiSEAL is the same as in the SEAL method itself, *i.e.* they only consider atom-based properties and partial charges. In Figure 7 can be found an example of an alignment done by Feher and Schmidt (2000).

Baum (2005), in turn, described a new algorithm based on *clique detection* applied to each correspondence graph of two molecular structures to identify structural similarities. The results from these pairwise comparisons are merged using binary matching trees which, in turn, allows to find the best alignments (see Figure 8).

In computer science, *clique detection* is the detection of cliques (complete subgraphs) in a graph. In molecular alignment, is the detection of a common substructure between two molecular structures. According to Lemmen and Lengauer (2000), regarding multiple molecule matching, depending on a distance tolerance

1 To find more on the subject refer to Hammersley and Handscomb (1964).

δ, the *clique detection* algorithm generates a so-called *distance compatibility graph*. This graph contains a node for each type and length compatible distance in a reference structure and one conformer of every molecule. Two nodes are connected if they share a common point in each of the structures. The matching procedure utilizes *clique detection* to determine overall valid distance constraints.

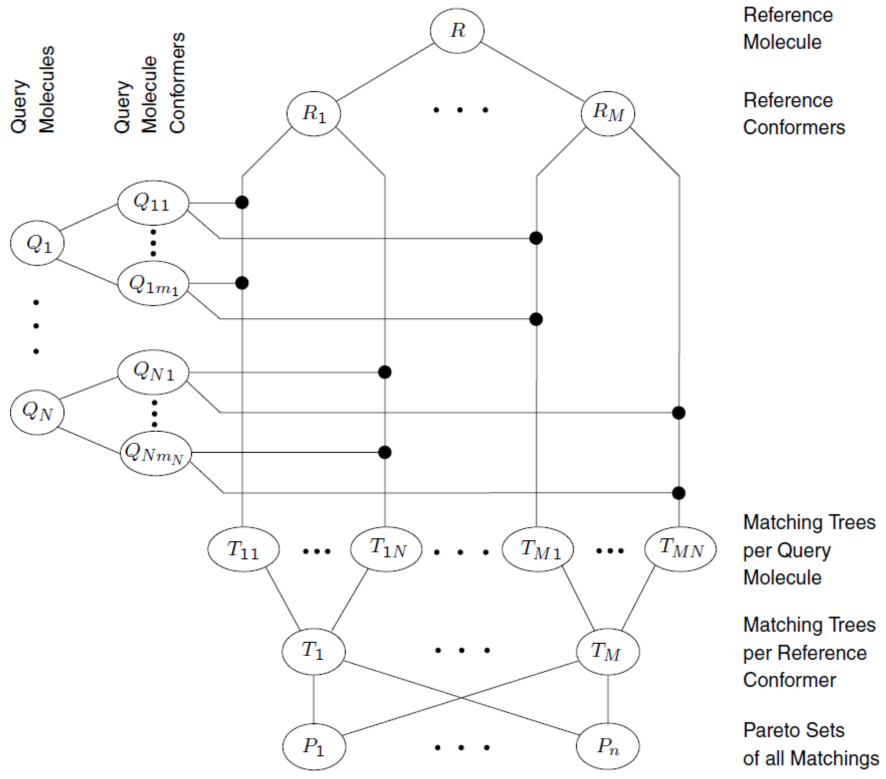


Figure 8: Overview of Baum (2005)'s algorithm. The reference molecule is denoted by R, its conformers by R_j. The query molecules are denoted by Q_i, their conformers by Q_{ij}. The • symbol denotes the computation of pairwise matchings. T_{ij} denotes the matching tree containing all pairwise matchings between molecule Q_i and reference conformer R_j. T_i is the final matching tree comprising all multiple matchings (matching clusters) with respect to R_i. Finally, P_i is a Pareto set containing multiple matchings of different reference conformers. *Fig.1. in (Baum, 2005).*

Since *clique detection* is limited to rather small sets of points, it was used an extra refining step to identify a larger number of common structures.

Furthermore, in this approach, all molecules are compared to a preselected reference molecule. This initial assumption is a disadvantage since it is impossible to evaluate *a priori* if the preselected molecule is a good reference structure. Another disadvantage lies in the extra refining step. Although the search for common substructures is made more thorough, it also requires a higher computational time, which forces a compromise between the runtime and the search detail. The runtime also depends on the number of conformers per molecule.

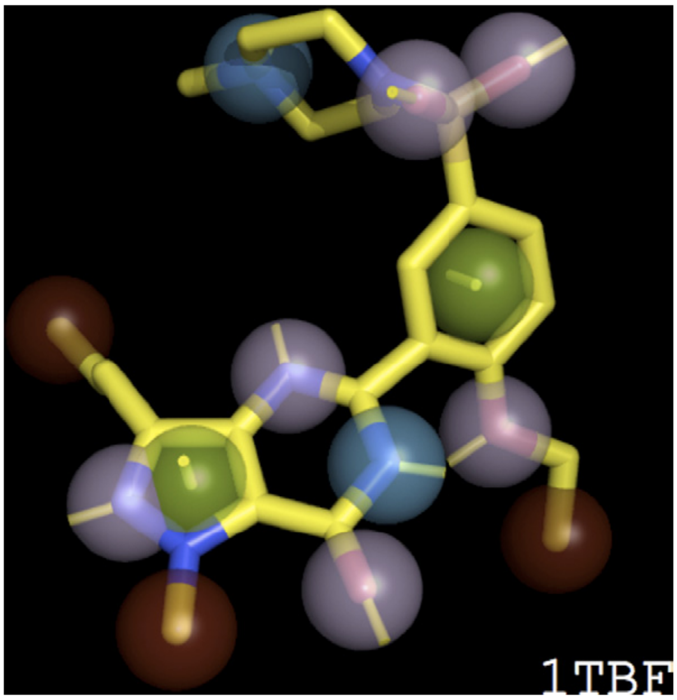


Figure 9: Compound *1TBF* together with its calculated pharmacophore points using Taminau et al. (2008)'s algorithm, *Pharao*. *Fig. 7. in (Taminau et al., 2008).*

Taminau et al. (2008) describes a pharmacophore alignment and optimization technique, ***Pharao***, based on Gaussian 3D Volumes. This method is inspired by an atom-based approach introduced by Grant and Pickup (1995), where the Gaussian volumes are attributed to every atom, instead of to every pharmacophore feature. Since the goal of this dissertation is to find flexible alignment of molecules based on their pharmacophore features, focus on the approach proposed by Taminau et al. (2008). Nonetheless, there are techniques and shape-based comparison programs (*Rapid Overlay of Chemical Structures*, ROCS) based on that atom-based approach (Grant et al., 1996; Rush et al., 2005).

Pharao is a tool that automatically detects several pharmacophore points within a molecule, assigns a respective volume to each, as pictured in Figure 9, and then begins the computation to find the subset of matching pharmacophore features that have the largest volume overlap. The usage of Gaussian volumes, due to the smooth nature of the continuous Gaussian functions, facilitates the computation of optimal alignments. No time or complexity related information was mentioned.

The software used by BIAL, *Internal Coordinate Mechanics software* (**ICM**), which was taken as a starting point for this dissertation, can be seen as a combination of all methods mentioned above. Molsoft's software, *ICM-Pro*, as *Pharao*, makes its own conformational sampling, based on a *biased probability Monte Carlo* (BPMC, (Abagyan and Totrov, 1994)), with local gradient minimization to optimize the docked ligand's internal variables, including six positional variables and all freely rotatable bonds. As MultiSEAL, it uses random moves to these variables followed by energy minimization based on a full energy function, represented in a specific data structure, in order to find a good ligand pose/conformation (Neves et al., 2012; Lam et al., 2018). And lastly, as Baum (2005), it uses a reference molecule and a special data structure, this time based on Gaussian property fields as representations of each ligand bond, to evaluate the best alignment (Lam et al., 2018).

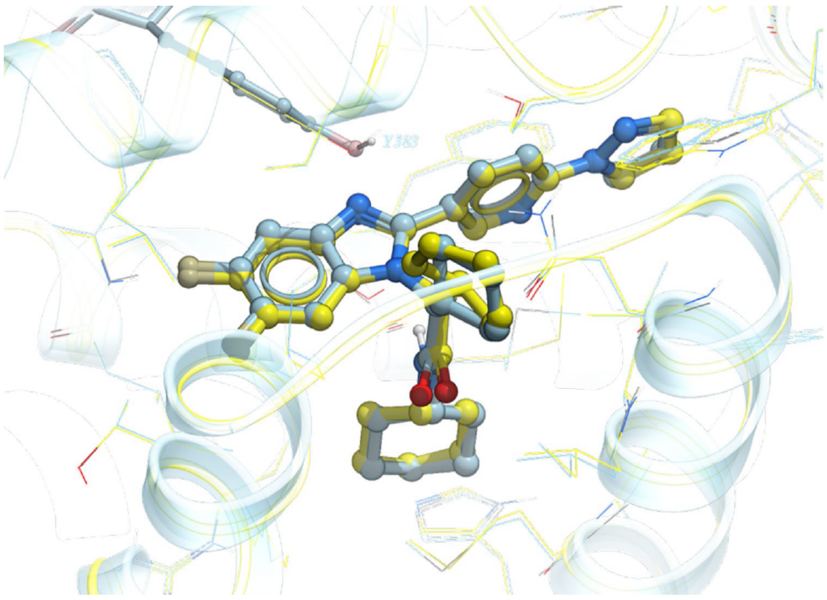


Figure 10: Predicted docking pose of FXR_26 (*blue*) versus the co-crystallized structure released (*yellow*). *Fig. 3 in (Lam et al., 2018).*

Here, each data structure represents a conformation of a ligand and their respective fitness to the protein (receptor) physicho-chemical atomic properties, and *atomic property field* (APF) interaction energy, giving a measure of chemical 3D similarity, that is, pharmacophoric potential, on a grid (Lam et al., 2018). APF can be generated from one or multiple ligands and seven properties are assigned from empiric physico-chemical components (classic pharmacophorics types: hydrogen bond donors, acceptors, Sp2 hybridization, lipophilicity; others: size, electropositive/negative and charge) (Totrov, 2008). An example of a conformation (blue) of a ligand (yellow) for alignment within a protein can be seen in Figure 10.

To summarize, ICM uses an hybrid ligand/receptor structure-based docking and pose selection method, *Ligand-Biased Ensemble Docking* (LigBEnD), by incorporating the APF method, also developed by Molsoft, into structure-based ensemble docking (Lam et al., 2018). This commercially available software ranked in first place for average RMSD docking accuracy in the *Drug Design challenge competition* (D3R) in both 2016-17 and 2018 (Molsoft, 2017, 2018). Although it only considers semi-flexible alignment, this states the accuracy and highly-use of semi-flexible methods for flexible molecular alignment.

3.2 FLEXIBLE MOLECULAR ALIGNMENT

Although, as stated, the software used as reference for this work, ICM, does **not** take flexibility as a direct parameter in the alignment, it is still relevant to review some articles related to the subject.

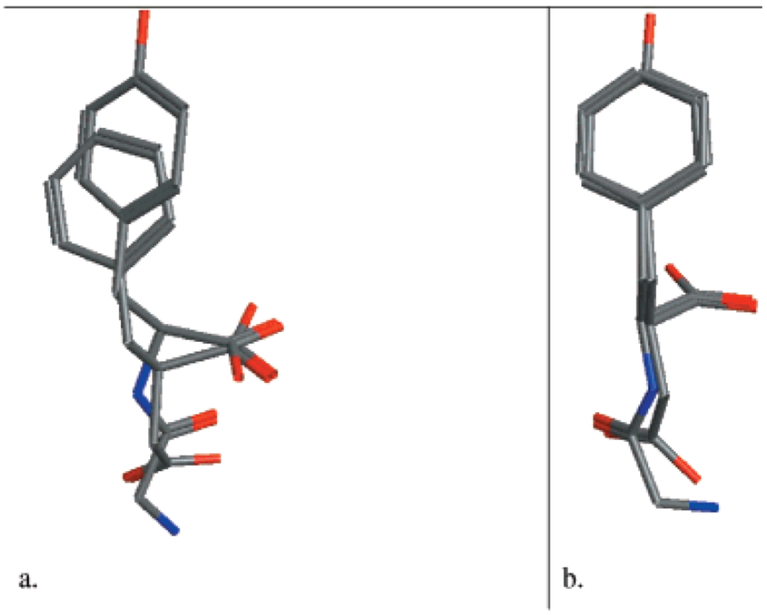


Figure 11: Overlay of *L-benzylsuccinate* and *glycyl-L-tyrosine*: (a) alignment in the crystal; (b) calculated flexible alignment at the final set of parameters optimized for all ligand pairs. *Fig. 3 in (Labute et al., 2001).*

Labute et al. (2001) created a method that produces a collection of alignments along with a score for each alignment based upon the internal energy of the molecules and a similarity score defined by an overlap of Gaussian feature densities. In the former, the volume, aromatic, donor and acceptor feature densities are used as similarities factors. In order to simultaneously search the conformational space of each molecule and the alignment space of the collection for optimal alignments, it resorts to a modified RIPS procedure (originally created by Ferguson and Raber (1989)) that can be summarized in five steps:

1. Set the values of the adjustable parameters;

2. Set all rotable bonds to random dihedral angles. Add a random number, within a certain range, to all atomic coordinates. Randomly orient all molecules by choosing three atoms randomly from each molecules and superpose;

3. Minimize the objective function with respect to the coordinates of all atoms;

4. If the new configuration has not been seen before, then set the variable k to 0, otherwise, $k = k + 1$. If k is greater than some predefined amount, then terminate the search and go to step 5; otherwise return to step 2;

5. Prune the list of configurations by removing all configurations in which the average potential energy (of the alignment) is greater than the minimum observed average potential energy plus some predefined threshold.

As a disadvantage, this method takes into account the X-Ray crystallographic coordinates to infer whether the alignment scoring function assigns the expected scores to experimentally determined alignments, which can be a faulty reference. In Figure 11 it is possible to see a calculated flexible alignment and respective crystallographic alignments.

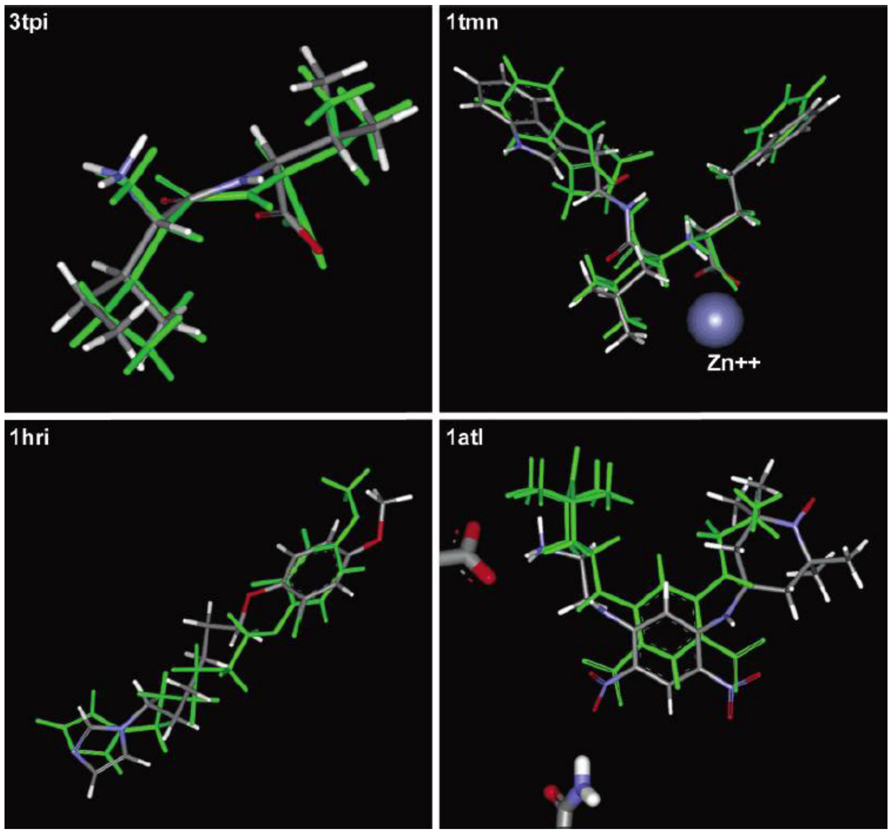


Figure 12: Examples of dockings executed by Surflex with high accuracy (*3tpi*); good accuracy (*1tmn*); acceptable accuracy (*1hri*) and poor accuracy (*1atl*). *Fig. 6 in (Jain, 2003).*

Surflex (Jain, 2003), a fully automatic flexible molecular docking algorithm, combines the scoring function from the *Hammerhead* docking system, also created by Jain (1996), with a search engine that relies on a surface-based molecular similarity method as a mean to rapidly generate suitable putative poses for molecular fragments. The Hammerhead docking system is an empirically derived scoring function based on the binding affinities (such as hydrophobic and polar complementarity) of protein-ligand complexes coupled with their crystallographically determined structures (Jain, 1996). Surflex combines the Hammerhead's function with a molecular similarity method (morphological similarity) to generate putative poses of ligand fragments (Jain, 2003). Figure 12 depicts examples of dockings with different levels of accuracy produced by Surflex.

Some disadvantages of this algorithm rely on the scoring function as it does not explicitly include the training on negative examples (which would reduce false positive rates) and the effect of nonbonded self-interactions within ligands. As in the previous algorithm, Surflex also depends on the closeness to already determined crystallographic structures as a parameter to the scoring function, making it an unreliable algorithm to find new structures that may be possible but have not yet been detected experimentally.

As a different method, Marialke et al. (2007) takes a combined 2D/3D approach for the superposition of flexible chemical structures based on common subgraphs identification and a gradient-based torsion space optimization algorithm. This method neither requires precalculated statistics on the conformations of the molecules nor does it make simplifying assumptions on the topology of the molecules being compared.

This algorithm is a *graph-based molecular alignment* (GMA) and consists of three steps: a preprocessing step, which produces mappings between the query and template, the actual mapping of the dihedral angles, and the torsion space optimization of the flexible degrees of freedom of the query molecule. In the first step, its used a *maximum common subgraph* (MCS) metric (Raymond and Willett, 2002), but because the standard one-to-one mapping between atoms is a poor choice for structurally mapping molecules (due to only taking into account direct connectivity), it resorts to a variant of graph isomorphism, which, in turn, leads to topologically correct mappings. The last two steps obtain the 3D coordinates of the query molecule by optimization of its conformations with respect to the *root-mean-square deviation* (RMSD) of the atoms that are mapped to the template. The optimization itself is obtained with an algorithm that solves the RMSD minimization in torsion space. This method does not require additional constraints or energy terms to keep the conformation of the ligand physically reasonable, that is, with correct bond lengths and angles (Marialke et al., 2007).

Unlike the two previous works reviewed, this algorithm does an unbiased alignment, *i.e.* it makes no use of the X-Ray structure of complexes. As a downfall, it always takes one of the molecules (the template) as rigid, not taking into account the flexibility of both

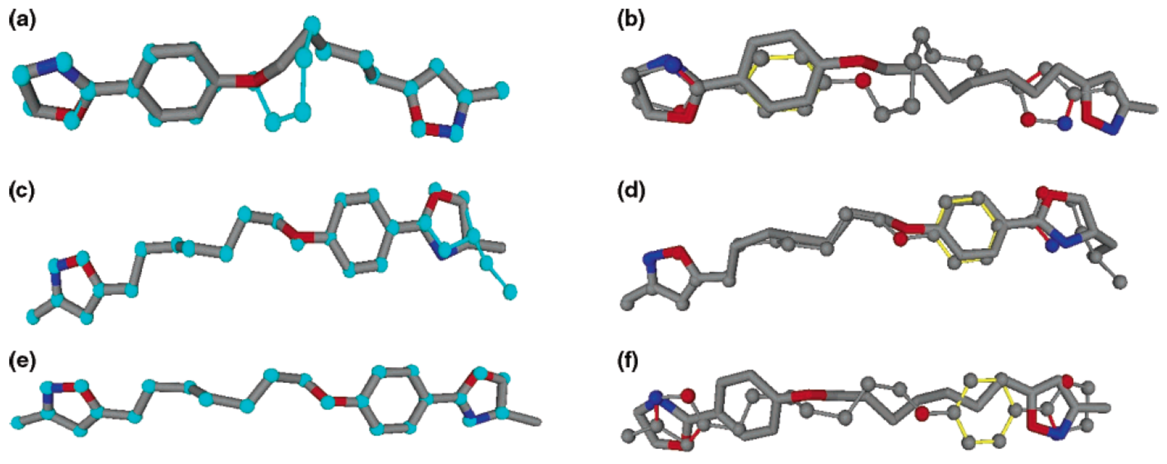


Figure 13: (a) 2r04 ligand (light-blue) aligned on 2r06 (thick sticks). (b) Comparison between aligned pose (ball-and-stick) and crystal structure for ligand 2r04 (thick sticks). (c) 2rs3 ligand (light-blue) aligned on 2rr1 ligand (thick sticks). (d) Comparison between aligned pose (ball and stick) and crystal structure for ligand 2rs3 (thick sticks). (e) 2r04 ligand (light-blue) aligned on 2rr1 ligand (thick sticks). (f) Comparison between aligned pose (ball-and-stick) and crystal structure for ligand 2r04 (thick sticks). *Fig. 3 in (Marialke et al., 2007).*

molecules to align. Figure 13 compares alignments obtained by Marialke et al. (2007) and the respective crystallographic alignment. It is possible to see that the alignments found are really close to the template, but the crystallographic structure did not fit so well to the pose generated. This proves that errors may be introduced when using the experimentally obtained structure as a parameter in the scoring function used to align molecules.

All methods until now, despite their individual pros and cons, have one con in common, the fact that they only consider two molecules at a time in the alignment.

On the contrary, the next approach, proposed by Schneidman-Duhovny et al. (2008), is able to automatically align multiple flexible ligands in a deterministic matter. This method, named *Pharmagist*, aims at pharmacophore detection by aligning (two or more) flexible ligands, with or without prior knowledge of the receptor. The input of this algorithm is a set of ligands, each given by the 3D coordinates of its atoms centers and covalent bonds between them. To avoid explicit conformational search, it is assumed that one of the ligands, called the *pivot*, is considered rigid. The other (*target*) ligands are treated as capable of exhibiting torsional flexibility about their rotational bonds. Additionally, the scoring function measures the similarity between a set of features from the targets and pivot based on a set of definitions discussed in section 2.1 of Schneidman-Duhovny et al. (2008)'s work. The features considered are physico-chemical properties relevant to ligand-receptor binding (aromaticity, charge, hydrogen bonding, or hydrophobicity).

Pharmagist works as depicted in Figure 14. In the first stage, each ligand is partitioned into rigid groups connected by rotatable bonds and is assigned a set of physico-chemical

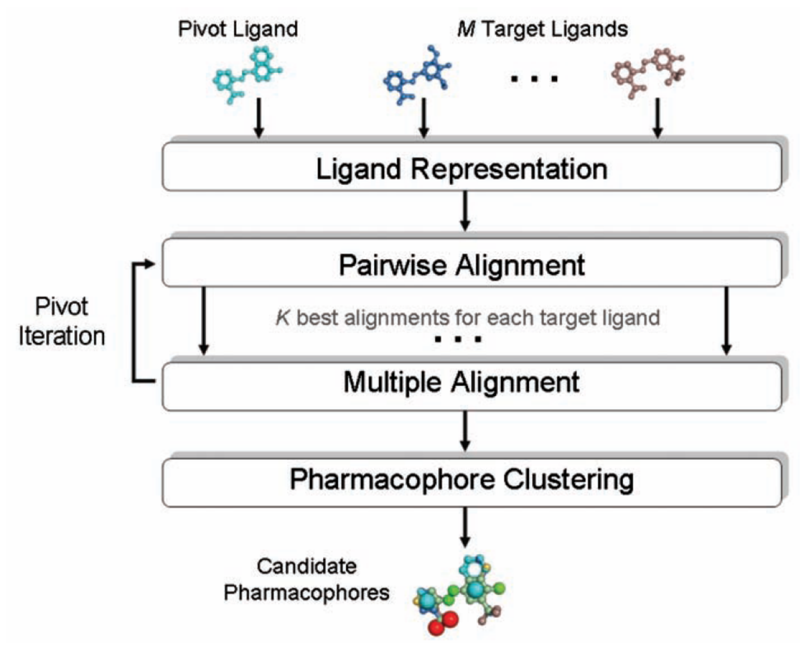


Figure 14: Method flow of *Pharmagist*. *Fig. 1 in (Schneidman-Duhovny et al., 2008).*

features. In the second, pairwise flexible alignments between pivot and each target ligand are computed. In the third, pairwise alignments are combined into multiple alignments between pivot and at least two target ligands. In the last stage, all candidate pharmacophores are clustered to produce a non-redundant set of solutions. The clustering stage is invoked only once to cluster solutions generated by all pivot iterations (Schneidman-Duhovny et al., 2008). In this method, the pivot can be selected by the user. However, by default, the assumption is that the pivot ligand is unknown and thus, the method iteratively selects each one of the input ligands to work as pivot. In the case where the pivot is selected by the user, there is no *Pivot Iteration*, as depicted in Figure 14.

The only negative characteristic of this algorithm is the fact that a pivot is always needed, *i.e.* conformational search is not explicitly applied. A very similar approach, but based on ant colony optimization, was proposed by Korb et al. (2010).

3.3 QUANTUM SIMILARITY SUPERPOSITION ALGORITHM - QSSA

Unlike any other algorithm reviewed above, this algorithm does not mention or consider molecular flexibility. It is only focused on optimization of rigid molecular alignment. This algorithm can be of greater interest in this problem than all the others previously mentioned. This relevance, of course, is not to this case study in particular, but for the design of future quantum algorithms for optimization of molecular alignment, rigid or flexible.

Due to this reason, it will be briefly done a review on the principles of molecular similarity and superposition based on quantum chemistry. In the next section, however, it will be continued the train of thought to the case study presented in this dissertation.

A concept like molecular similarity is not so easily described in a mathematical form based on quantum chemical ideas. The same goes for the alignment of molecules. In a visual approach, the process starts by trying to find spaces within the two molecules where a high similarity becomes apparent. In most of the molecular alignment procedures, this similarity is based on molecular geometries. Consequently, when such topographical similarity is absent or not expressive enough, this constitutes a limiting problem. The work of Bultinck et al. (2003) takes into account the topographic properties along with a general and internally consistent scheme based on quantum chemical ideas.

In quantum chemistry the determining entity is the wave function itself. Although this mathematical object does not carry any physical meaning, it is possible to obtain information with physical meaning, such as the electron density, using the density functional theory. Bultinck et al. (2003) builds an approximate electron density in order to derive similarity and alignment equations and algorithms. The molecular similarity is calculated through the overlap of molecular densities between two molecules, and the alignment is performed in such a way as to maximize the molecular similarity. This is achieved in terms of the relative orientation of the second molecule's coordinate system with respect to that of the first molecule, using three translation parameters and three Euler angles. Due to the large number of local maxima, this optimization involves a Lamarckian genetic algorithm with the simplex method as a local optimizer (Bultinck et al., 2002b,a, 2003).

The method used, *molecular quantum similarity* (MQS), was first defined in 1980 (Carb et al., 1980) and states that the quantum similarity measure of two molecules, A and B, can be simply obtained through the integral measure, Z_{AB}, expressed in Equations (1) and (2). All elements Z_{AB} form a symmetrical $(N \times N)$ matrix $\mathbf{Z}$, called *quantum similarity matrix*.

$$\begin{aligned} Z_{AB} &= \int \int \rho_A(\boldsymbol{r}_1)\Omega(\boldsymbol{r}_1, \boldsymbol{r}_2)\rho_B(\boldsymbol{r}_2)d\boldsymbol{r}_1 d\boldsymbol{r}_2 \\ &= \int \rho_A(\boldsymbol{r}_1)\rho_B(\boldsymbol{r}_1)d\boldsymbol{r}_1 \end{aligned} \tag{1}$$

where $\rho_A(\boldsymbol{r}_1)$ refers to the electron density of molecule A at some point $\boldsymbol{r}_1$ in space; $\Omega(\boldsymbol{r}_1, \boldsymbol{r}_2)$ is a chosen positive definite operator, such as the kinetic energy operator or the coulomb operator. In this case was used the Dirac function $\delta(\boldsymbol{r}_1 - \boldsymbol{r}_2)$.

Due to the very high computational effort, the electron density of a molecule with M_A atoms is approximated as the sum of the electron densities of the atoms composing the molecule. This is called the *Promolecular Atomic Shell Approximation* (PASA), which is later expanded in terms of a basis set of spherical *Gaussian Type Orbitals* (GTO). The GTO's employed are all chosen as spherically symmetric s-type orbitals, which yield rotational invariant density functions. After these implementations and normalization, Equation (1) becomes:

$$\begin{aligned} Z_{AB} &= \int_{-\infty}^{\infty} \Big(\sum_{a}^{M_A} Z_a \sum_{i_a}^{N_G} w_{i_a} |s_{i_a}(\boldsymbol{r})|^2\Big)\Big(\sum_{b}^{M_B} Z_b \sum_{i_b}^{N'_G} w_{i_b} |s_{i_b}(\boldsymbol{r})|^2\Big) d\boldsymbol{r} \\ &= \sum_{a}^{M_A}\sum_{b}^{M_B} Z_a Z_b \sum_{i_a}^{N_G}\sum_{i_b}^{N'_G} w_{i_a} w_{i_b} \int_{-\infty}^{\infty} |s_{i_a}(\boldsymbol{r})|^2 |s_{i_b}(\boldsymbol{r})|^2 d\boldsymbol{r} \end{aligned} \tag{2}$$

where Z_a refers to the atomic number of a single atom a, w_{i_a} is the expansion coefficients and N_G is the number of GTOs of symmetry s used in the expansion.

To see the entire equation development refer to Bultinck et al. (2003).

The results, nonetheless, depend on the alignment of the involved pair of molecular structures. Depending on the relative orientation of both molecules, their electron densities will differ in the same point in space in an external coordinate system. Aligning the molecules by searching the relative orientation of molecule B versus molecule A (where Z_{AB} becomes maximum) is, once again, an approach with a high computational effort.

Although it is a very relevant procedure, due to lack of relevance to the main purpose of this dissertation, this article will not continue to be reviewed. Nevertheless, it is important to emphasize certain aspects and highlight the relevance of this article in the future world of quantum simulations for chemistry or medicinal chemistry and drug-related problems. This method and respective equations only take into account two molecules and until now, there were two times where the computational power required could be too high. This is important because, following the principles of quantum mechanics, these equations can easily include interactions between several molecules. The reason for not implementing lies on the lack of computational power, characteristic that may be solved with the help of quantum simulations on quantum devices.

As stated by Feynman (1982):

"Nature isn't classical, dammit, and if you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly, it's a wonderful problem because it doesn't look so easy."

Following the same principle of quantum simulations based on pure quantum mechanics, as is the case of quantum chemistry, is the simulation of small molecules. As already mentioned in section 1, it was explored a case study on simulation of small molecules using quantum computation, in which is present my personal contribution, during the same timeline of this dissertation.

In Tavares et al. is reviewed all relevant theories, tools, techniques and explored all steps of the case study, from the conception to its actual execution. These include the Hamiltonian modeling of the molecular system, the Hamiltonian translation to a quantum circuit and finally, the actual execution of the circuit in a quantum device. From such Hamiltonians is calculated some molecular properties of interest, such as the ground state, the dissociation energy and the Stark effect.

4

THE APPROACH

The purpose of the case study addressed in this dissertation is to discover the best alignment between sets of molecular conformations, *i.e.* the best alignment between flexible molecules given some corresponding conformations. As discussed in section 2.3 and mentioned in section 3.1, these means that not a totally flexible alignment but rather, a **semiflexible molecular alignment** was adopted. In this case study, each conformation is represented by the Cartesian coordinates of a given set of points. These points are not the atoms, but the midpoints of a specific pharmacophore feature (which is a set of atoms that together have a given property), and each of them can only be "connected" (superposed) to another point with the same feature.

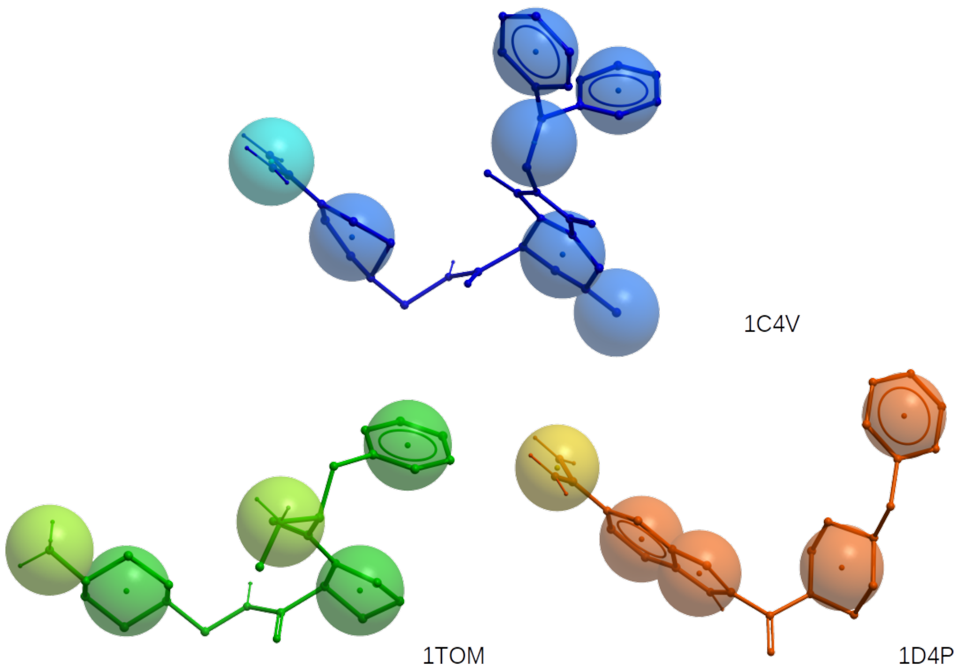


Figure 15: Stereochemical formulae of the conformations of ligand molecules (1*C*4*V*, 1*TOM* and 1*D*4*P*) with respective pharmacophore features surrounded by 2Å spheres. Positive charges represented by the lighter colors and hydrophobicity by the darker colors within the respective molecule's colors (blue, green and orange).

This abstraction of perceiving a conformation as a set of points with specific properties is at some point a good approach to this sort of problems since it does not take into account a specific molecule or geometry, but instead a structure with some properties. This allows the initial data to be any type of molecule, provided that it respects the positions of the groups that have the properties that have been tested. Therefore, as easily concluded from section 2.1, this is an **approach based on the ligand-based pharmacophore model**.

As illustrated in Figure 15, the number of points is variable among different molecules, and so, among different conformations from different molecules. For a better understanding: there can be a conformation with two points with Positive charge properties and one point with Hydrophobic properties and another conformation, from a different molecule, with only one point for each feature.

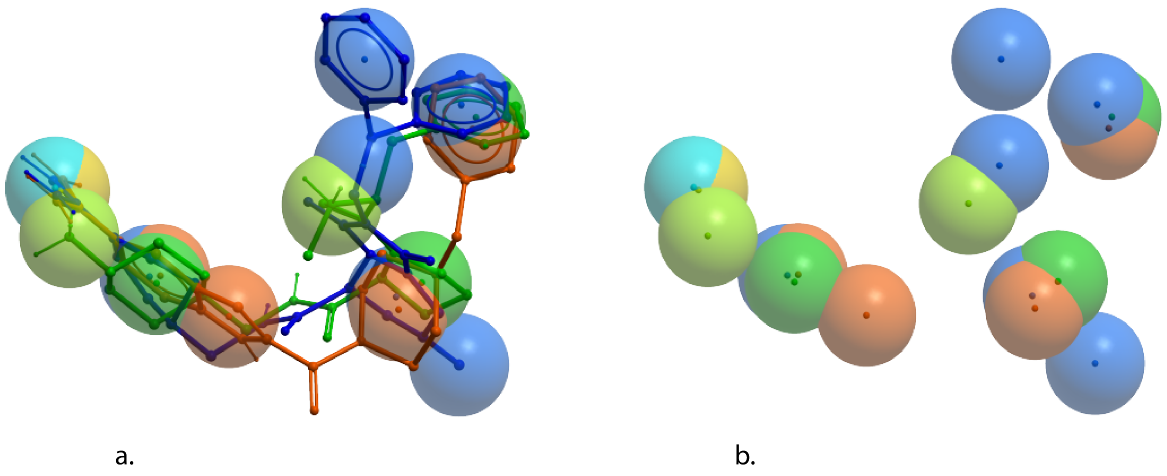


Figure 16: Alignment of conformations of ligand molecules ($1C4V$, $1TOM$ and $1D4P$) with respective pharmacophore features surrounded by 2Å spheres. (a) with the molecule structure, (b) without. Data used as depicted on (b).

As can also be seen from Figure 15 and 16, due to the structure and chemical properties of molecules, each point can only be aligned to another point from each conformation. Furthermore, another relevant restriction is that two points can only be aligned if they are at a maximum distance of 2 Å, also represented in Figure 16. Both these restrictions exist because this alignment method is based on distance and chemical affinity.

To summarize, these are the imposed restrictions to the alignment problem:

1. Two points can only be aligned if they have the same chemical feature.
2. Each point can only be superposed to one and only one point from each conformation.
3. Two points can only be considered aligned if they are at a maximum distance of 2 Å.

In the end, the solution is a set of conformations, one from each different molecule. Additionally, it is important to highlight that there is not a unique solution to this type of problem. There can be several possible sets of conformations that respect all the restrictions, from which, the "best" is the one with the most points aligned at the least distance possible.

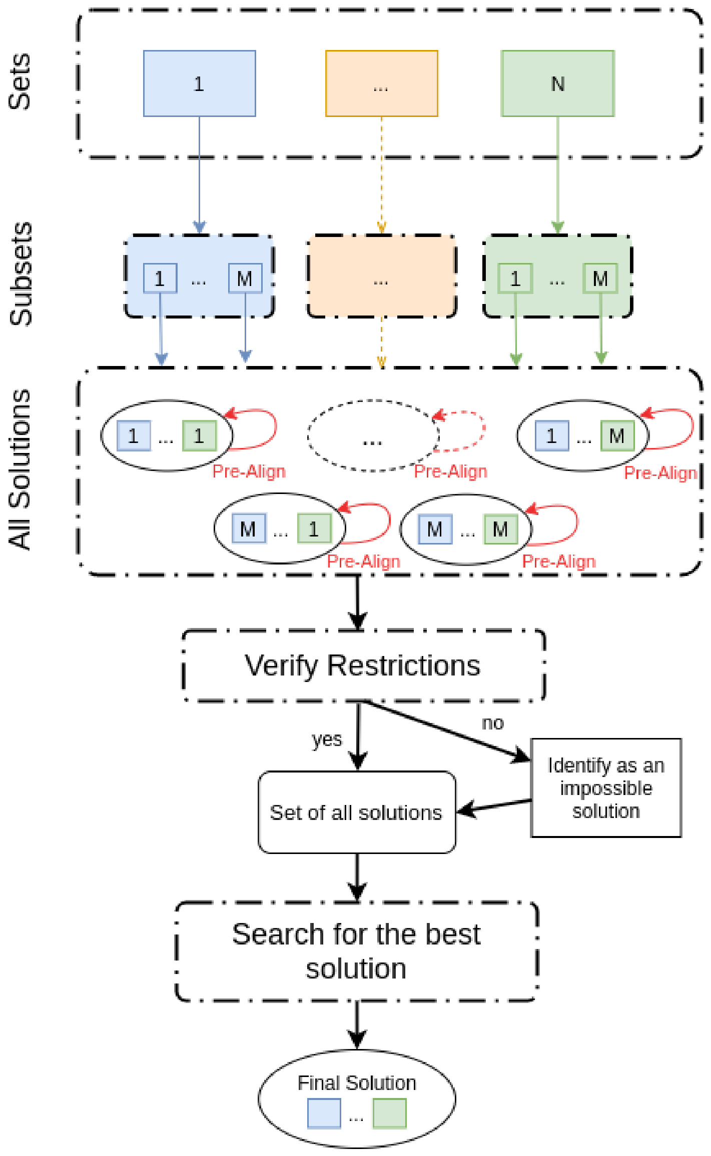


Figure 17: Flow chart of the problem. The order of alignment is indifferent. ©SofiaOliveira, 2020

Therefore, it is possible to write out the problem as illustrated in Figure 17, and as so:

With N sets, each set M subsets and each subset a variable number of points, choose the subset from each set that contributes to the best alignment. This alignment depends on the distance between the points of the same type, from each subset. Each point can only be connected to a maximum of $N-1$ *points, one from each set. The calculation of 'fitness' of a given N subsets takes into account all possibly connected points that are at a maximum of 2 Å distance from each other. The mentioned* *subset of points is a rigid set of points that can undergo translation and rotation to achieve the best position for posterior analysis of alignment.*

Since this is an optimization problem and the goal is to change it to an entirely different paradigm, quantum computation, one should abstract from their non-mathematical parameters. To do so, it is important to formalize the way one refers to the point's properties. This abstraction was made taking into account the actual data but keeping it adaptable.

4.1 THE DATA

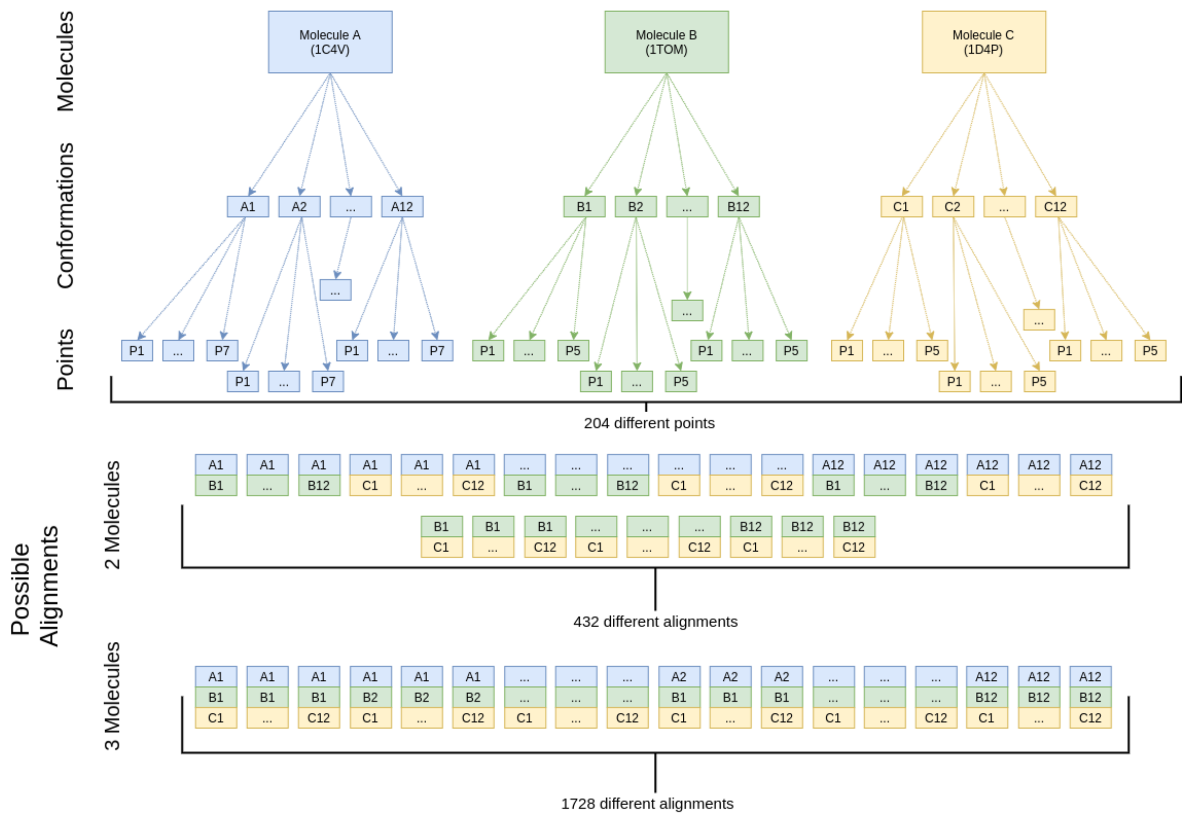


Figure 18: Organizational flow chart of the data and possible alignments. The order in which these alignments are done is indifferent. Each point, referenced as P* (where '*' stand for a given number), has 3 integer values (for the x, y and z coordinates) and a string value, the type (representative of the pharmacophore feature). ©SofiaOliveira, 2020

In the initial stage, on which all assumptions and decisions were based, there are **twelve** **conformations** for each of *three molecules* and there is only **two different pharmacophore properties: Positive charge and Hydrophobic**. Due to the required abstraction from chemical aspects and to simplify, from now on, the Positive charge property of a point will be referred as a point of type P, and the hydrophobic property as the type H. In Figure 18 is possible to consult the organization of the data and possible alignments.

The points from each conformation are given as Cartesian coordinates relative to a fixed axis. This means that the data received can be located far apart and so, the third restriction above, which states that two points can only be considered aligned if they are at a maximum distance of 2 Å, could never be obeyed. To correct and prevent this situation, an additional step is required, the spatial pre-alignment of the conformations. This pre-alignment step is already depicted in Figure 17.

In the end, the solution consists of one set of three conformations (one from each different molecule) when considering all the molecules generating a pharmacophore, or a set of two conformations, when considering only two of them. As a consequence, the number of different molecules chosen to generate a pharmacophore is a degree of freedom and a decision one needs to previously make. This degree exists because there is no fixed number of molecules required to create a pharmacophore. Indeed, the higher the number of different molecules used, the more accurate would be the pharmacophore. However, in the proposed case by BIAL, it was given freedom in this parameter and the number of different molecules used to create a pharmacophore became a choice, which will be further explained later on.

4.2 THE APPROACH

How should one organize the data? Even more important, what should be the approach to the problem? These were questions that immediately emerged and changed this problem to a purely computational problem, not a chemical one. Graphs have been used previously to solve the problem of flexible molecular alignment, and as they are well studied and characterized mathematical structures, it came as obvious that they could provide a good approach. Then came the questions: Should one deal with this problem as a graph problem (graph similarity or subgraph matching problem), *i.e.* treat data as graphs (*Approach 1*)? Or use graphs as a tool to solve the problem (*Approach 2*)? To deliver a reliable answer, each of the approaches needs to be studied. But first, a brief review to the notion of a graph.

Graph theory deals with the way objects are connected. For a mathematician, a graph is the application of a set on itself, *i.e.* a collection of elements of the set and a binary relation between these. There is hardly any concept in the natural sciences which is closer to the notion of graph than the constitutional formulae of a chemical compound because one may directly use a graph to represent a molecule if considering only its internal connectivity (*i.e. chemical bonds between atoms in a molecule*) (Trinajstic, 1992).

Using the notation in Trinajstic (1992), a simple graph is defined as an ordered pair $[V(G), E(G)]$, where $V = V(G)$ is a nonempty set of elements called *vertices* (or points) of G and $E = E(G)$ is a nonempty set of unordered pairs of distinct elements of V(G) called *edges* (or lines). For further information on the subject check Trinajstic (1992), where is given a chemical point of view, or West (2001), which follows a purely mathematical approach.

Next, the idea behind these two approaches, the problem and, in the second case, the envisaged solution, is described.

APPROACH 1

The first approach resorts to two well-known graph problems, *subgraph matching* and *graph similarity*, where the goal is to match a graph to another in the first case, and assess the similarity between two graphs, in the second. Both tasks need to be accomplished, in that order, in this problem.

Based on Koutra et al. (2011), one can state these two problems as follows:

Subgraph matching involves identifying the coherent or well-connected subgraphs that appear in some or all of a set of graphs, that is, it tries to extract a subset of *nodes* that are highly connected (sharing a great number of *edges*).

Graph similarity, by its turn, is the classification of similarity based on the mapping between the graph's *nodes*. The mapping can be established through graph isomorphism, feature extraction or iterative methods.

IDEA: Each subset forms a graph resorting to a subgraph matching algorithm, make an evaluation of each possible solution and, choose the best matching score based on a graph similarity algorithm.

PROBLEM: The points, from each subset, have no known ”connections” between them, *i.e.* there is no set of vertices. Although one may think that these could be easily inferred by geometric proximity, that could possibly lead to biased and unreal data. The original data are *nodes* (points) with no relation between them[1]. This means the data cannot be treated as graphs and thus makes it impossible to solve the problem as a graph problem.

APPROACH 2

The second approach takes advantage of graphs but only as an auxiliary data structure to solve the problem.

1 In a discussion with BIAL, it was attempted to convert these abstracted data into a graph by connecting all points to a *null* new point, creating a center point with all other points connected to the *null*. Although this would indeed turn the data into graphs, it would not bring any greater advantage over the second approach discussed below.

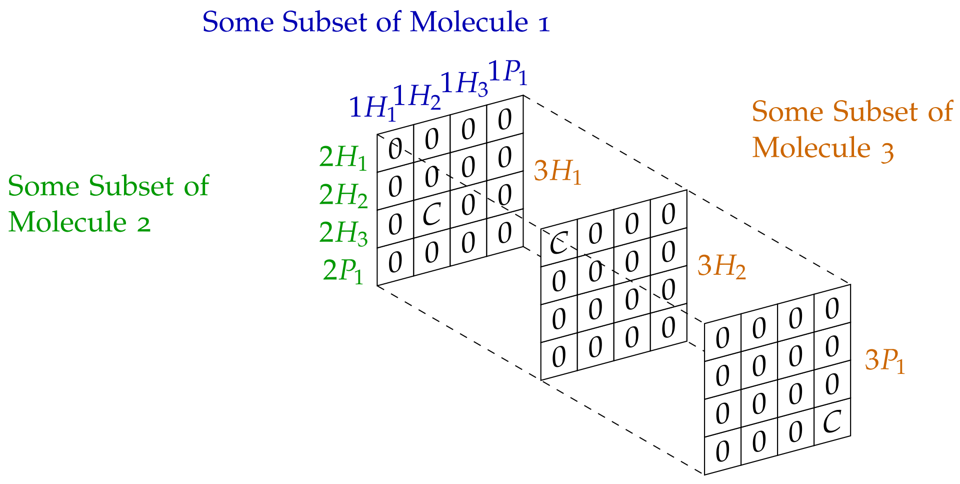

Some Subset of Molecule 1
$1H_1$ $1H_2$ $1H_3$ $1P_1$
Some Subset of Molecule 2
$2H_1$ $2H_2$ $2H_3$ $2P_1$
Some Subset of Molecule 3
$3H_1$ $3H_2$ $3P_1$
C
0

Figure 19: 3-Dimensional Matrix of approach 2 after evaluation of which points are connected. In this example are taken into account conformations (subsets) of 3 different molecules (sets), with different number of points. C stands for connected. ©SofiaOliveira, 2020

IDEA: **Creation of multiples graphs**, each representing a possible solution. Every of these graphs would be represented by a $\prod_i^N K_i$ dimension matrix, where N stands for the number of subsets to be matched at once and K_i the number of points of the subset i. Each matrix dimension is a subset of points.

The data in a given matrix position can either be an integer or 0, where 0 stands for non-connected and otherwise connected. The solution would be the graph with the best score. This score is a relation between the number of non-zero elements of the matrix and the sum of theirs values since these represent the distance between the elements that are connected. The points chosen to be connected will connect so after the verification of restrictions. Figure 19 illustrates this approach.

PROBLEM: When comes to subsets with a different number of points of the same type, there may exist the possibility of some points of at least two subsets not being connected to at least one of the other subsets, within the same graph (possible solution). Having an $\prod_i^N K_i$ dimensional matrix restricts this type of connections since it demands, at every time, N points to be connected to each other.

SOLUTION: This was solved by analyzing not only possible solutions with N subsets (which implies one subset from each different molecule), but also, all possible solutions with size lower than N $(N, N-1, ..., 2)$, which implies that a possible solution may not involve all N initial molecules. This, of course, entails the need for an analysis of different alignments, each considering a different number of molecules (sets).

In conclusion, *approach 2* was the followed approach in this dissertation.

4.3 DIVISION OF THE PROBLEM

Quantum computation still has a long way to go, which means that a complex problem, such as the one to be studied here, needs to be broken into small pieces in order to be solved. With that in mind, and by looking at Figure 17, one can easily see three distinct computational tasks as depicted in Figure 20:

1. Spatial pre-alignment of conformations (*Pre-Alignment*)
2. Verification of restrictions (*SAT Solver*)
3. Search of the best solution (*Solution Search*)

The first one, only by itself, is a complex task that needs to consider rotations and translations of a rigid set of points (as in a conformation) in order to minimize the overall distance between different sets of points (different conformations). After consideration, a quantum algorithm of this segment of the problem was put aside and the idea of an hybrid algorithm, with both quantum and classical computations, emerged. A totally classic algorithm would tackle this first step, as illustrated in Figure 21.

In the next section, such algorithm will be introduced and further explained, but in an overall view, this is a task of registration of 3D point sets, component key in many computer vision problems (Fitzgibbon, 2003; Myronenko and Song, 2010; Wu et al., 2013). The goal of point set registration is to assign a correspondence between two sets of points and to recover the transformation that maps one point set to the other (Myronenko and Song, 2010). In this particular case, the map transformation is not relevant, only the final position and the correspondence error between the two sets of points to be aligned.

On the other hand, **the second step**, verification of restrictions, has a direct resemblance to a very well known problem, *CSP*, short for constraint satisfaction problem. While *CSP* deals with discrete variables, satisfiability *SAT* problems, also known as binary *CSP*, deals with binary variables. As stated by Gu et al. (1996), due to this close relationship, any *CSP* algorithm can always be transformed into a (*SAT*) algorithm. This is important because working with binary variables is a lot less expensive (from a computational perspective) than working with discrete ones. This is relevant when one is dealing with devices with low computational power, as it the case of the nowadays NISQ (*Noise Intermediate-Scale Quantum*, (Preskill, 2018)) devices. Nevertheless, the goal of a *SAT* problem is the same as the *CSP*, determine whether there exists an assignment of truth values to variables that makes formula (3) satisfiable (Gu et al., 1996).

$$C_1 \wedge C_2 \wedge ... \wedge C_n \tag{3}$$

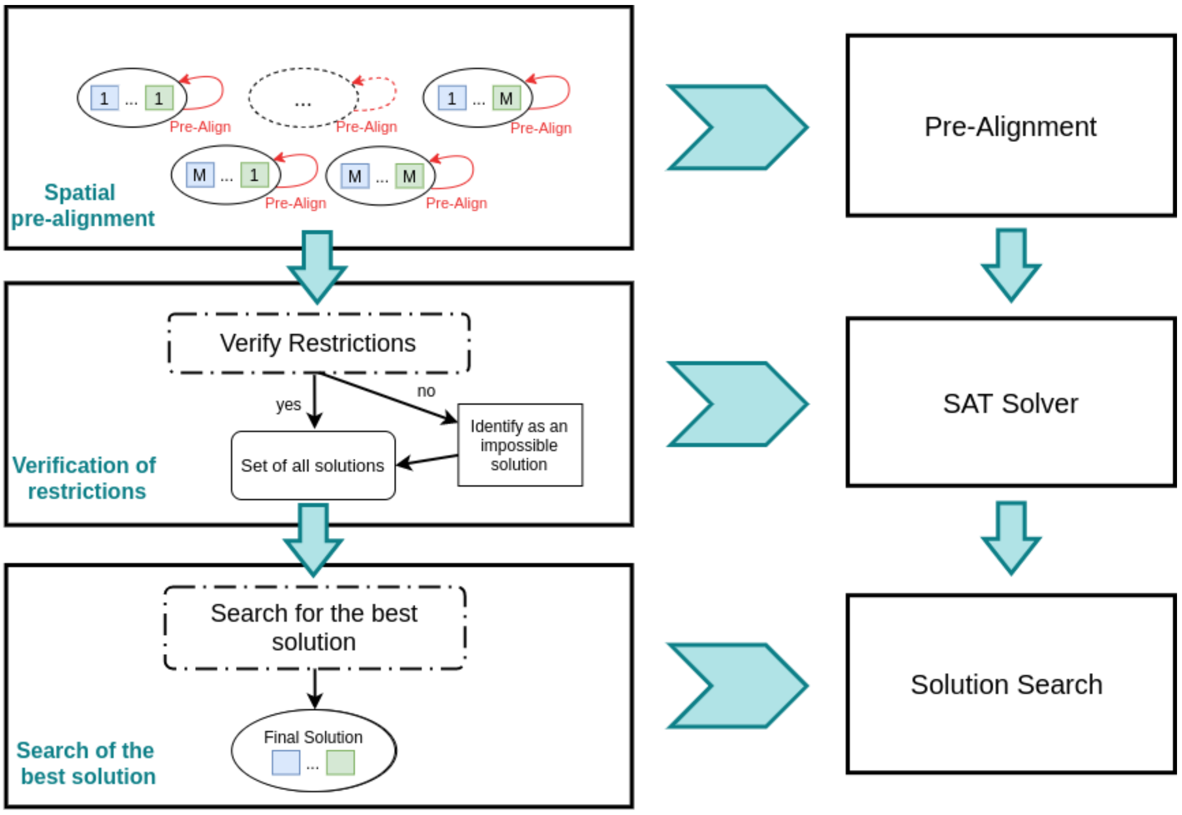


Figure 20: Flow chart of the division of the problem (right) based on Figure 17 (left). ©SofiaOliveira, 2020

Each C_i, for $i \in 1...n$, is a restriction with value *True* of *False* witnessing the verification of a restriction. When this formula is satisfied, all n restrictions imposed are verified, exactly what is intended for this step in the case study. Accordingly, there will be a quantum algorithm that is capable of verify the satisfiability of the restrictions, which, for the sake of coherence, will be named *Quantum SAT* (*qSAT*) solver. This *qSAT* solver will be iteratively applied to each possible set of alignments but every intermediate step will be executed classicaly, as depicted in Figure 21. (In order to better understand or recall what are all possible sets of alignments, please check the bottom of Figure 18.)

The last, but not least, **third step**, is the final search throughout all alignments, of the one that fits the criteria, *i.e.* that respects the restrictions. The search takes into account, not only if the criteria are met, but also what is the alignment that has the highest possible number of points correspondence and the least overall distance between those points. As in the previous step, there will be a quantum algorithm to execute this search but all intermediate steps are classical. The procedure is also illustrated in Figure 21.

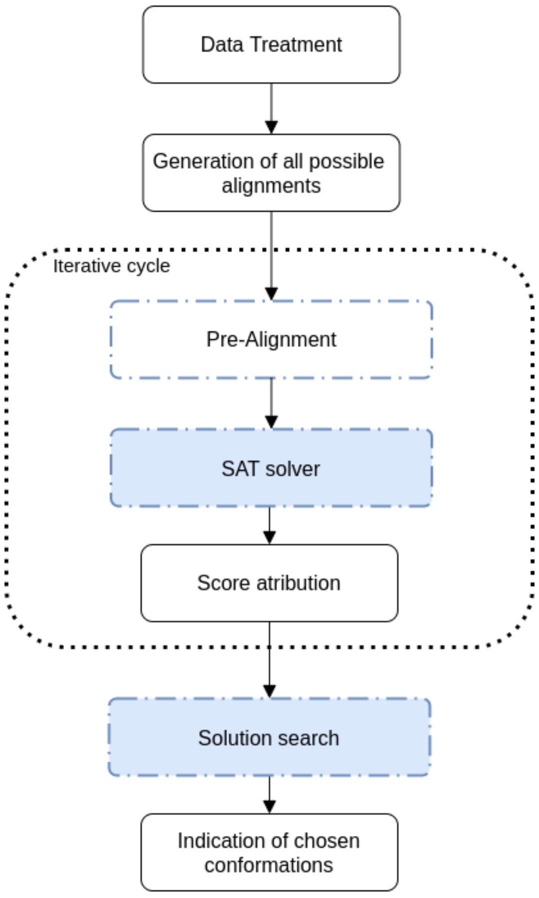


Figure 21: Flow chart of the problem hybrid approach. Filled blue boxes are steps under quantum paradigm and large dotted box stands for an iterative cycle throughout all possible alignments previously generated. ©SofiaOliveira, 2020

4.4 REENGINEERING OF THE PROBLEM

To close this section, one needs to recall that one of the goals of this dissertation is to compare and analyse the viability of a quantum algorithm for this type of problem. Having that in mind, it is relevant to compare similar processes in the quantum versus classic paradigm. Although the present case study is based on a software, ICM, the actual problem solved by ICM and the one solved in this dissertation, are not computationally the same. ICM takes several more steps, such as the initial generation of the conformational space, the energy function minimization as a parameter of alignment, and the consideration of the molecule's bonds to the actual alignment. ICM takes into account much more information than the one used for this case study. Of course, because the problem to tackle is not the same and it will be substantially abstracted and reduced compared to the original[2], it only

2 Here, qualifier original stands for the commercially available optimized software, but not exclusively, ICM.

makes sense to create a classical algorithm that tries to solve the problem in the same steps as the quantum algorithm.

With that in mind, it was done a classical reengineering of the problem executing the same steps as the quantum algorithm. For that, a rather big research was necessary in order to find and understand classical algorithms for this purpose. All of this process is documented in the next section.

Meanwhile, is important to mention that the best comparison regarding viability of an algorithm versus another always comes by comparing both of them with their best abilities in action. Since the quantum devices available do not represent, by far, an optimal quantum computer, it is not possible to make a fair comparison in that matter. Indeed, considering the very noisy quantum computation versus the very well establish classical computation existent nowadays, the classical algorithm would always win. The question remaining, that hopefully could be answered in the future, is: Considering the ideal conditions for both algorithms to run, which would get the best or the fastest results?

5

REENGINEERING OF A CLASSIC SOLUTION

When approaching a new problem, an initial step is evaluating similar problems, if not possibly the same, and respective resolutions.

As stated previously, the case study is not the whole problem of *Flexible Molecular Alignment*, but instead a small section of its components. That is, it will not be considered the whole conformational space of the molecules neither their respective analysis. This dissertation will only focus on a small set of pre-computed conformations per molecule. Nowadays, this small section is never considered alone, *i.e.* the whole problem is considered. Hence, there are not updated methods that could be used for the comparison, analysis, and comprehension of the problem. Consequently, and as concluded in section 4.4, a classical approach to the problem is also necessary to be discussed. With this in mind, classical approaches to the case study are documented in the current section, starting with the pre-alignment of conformations, which will not be implemented by a quantum algorithm; next the *SAT* implementations tested and used, and lastly the score evaluation and solution choice. All respective programs can be found in the Appendix.

For a better understanding of the methods used, check Figure 22 for equivalence of data structures, depicted in Figure 18, in *pyhton* language.

5.1 PRE-ALIGNMENT

The aim of this first step is to move the conformation as a rigid set of points by rotations and translations in order to make it the closest possible to the other conformations to which the first one is to be aligned. At an abstract level this problem can be seen as a *Rigid Point Set Registration*.

Point Set Registration (*PSR*) is, as already mentioned in section 4.3, a common problem in computer vision where the task of *registration* is to place the data into a common reference frame by estimating the transformations between the datasets (Fitzgibbon, 2003). Therefore, *PSR* is a search for the transformations necessary to make correspondences between two

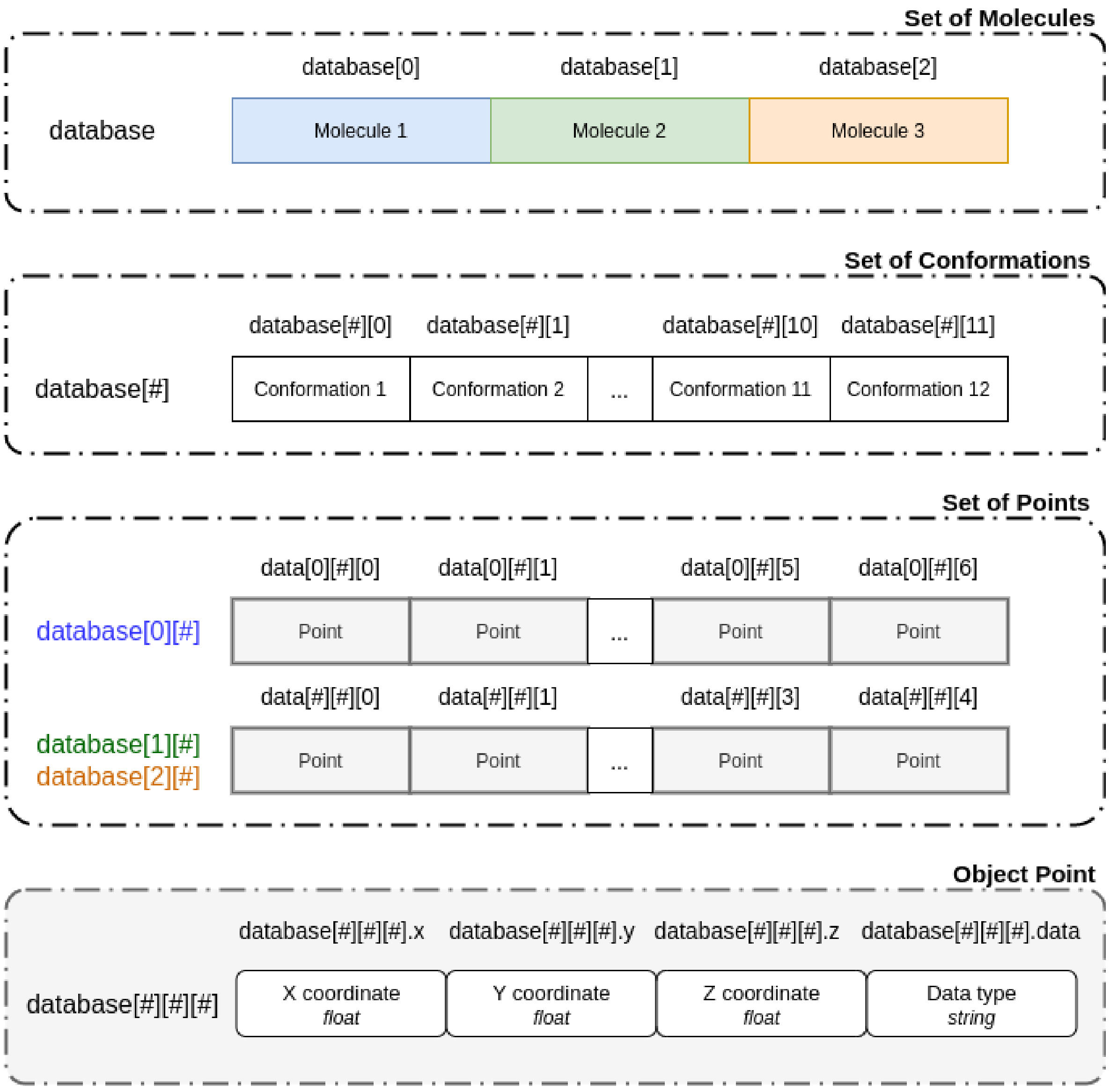


Figure 22: Data organization through *pyhton*. Representation with equivalent chemical meaning, colors added to simplify correlation. Real list size illustrated for each case. (# stands for any valid position of the list.) ©SofiaOliveira, 2020

sets of points (Fitzgibbon, 2003; Myronenko and Song, 2010; Wu et al., 2013). An illustration of this problem can be found in Figure 23. *Rigid Point Set Registration*, on the other hand, implies a specific search for a *rigid transformation*, also known as the *Euclidean transformation*, that makes such correspondences. A *rigid transformation* is a transformation that preserves the distance between points and can be seen as if there were a rigid link between points, as illustrated in Figure 24.

According to Bottema and Roth (1990) and McCarthy (2013), a rigid transformation is only composed by translations, rotations, and possibly reflections, definition that will be

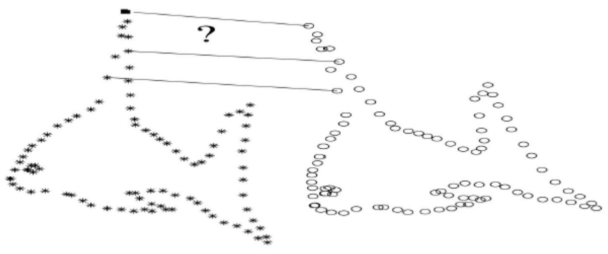


Figure 23: Problem of point set registration: given two sets of points, assign the correspondence and the transformation that maps one point set to the other. *Figure 1. in Myronenko and Song (2010).*

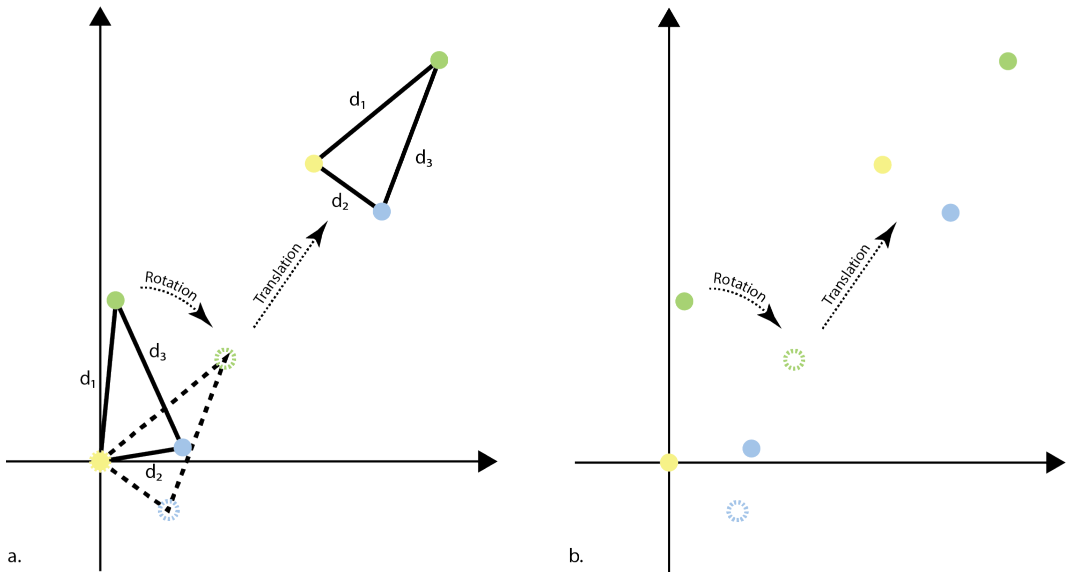


Figure 24: Example of a rigid transformation, including rotation and translation, (a) with visible rigid links between points, and (b) without. d_1, d_2 and d_3 is always maintained during transformation. Note that (a) is merely a representation, there are no actual links between points, as represented in (b). ©SofiaOliveira, 2020

followed in this dissertation when mentioning a *rigid transformation*. However, in Myronenko and Song (2010) is also mentioned another geometric transformation, scaling[1], which, by abuse of language, is also considered as a rigid transformation. Regardless, the latter reference will be used so please take notice when such differences appear (which will be mentioned). Explicitly, formula (4) is the considered formula in Myronenko and Song (2010) while in Bottema and Roth (1990) and McCarthy (2013) is considered the one represented by formula (5).

1 Linear transformation that increases or diminishes structures by a scale factor that is the same in all directions, maintaining the proportion between points but altering the corresponding distances.

$$T(\mathbf{v}) = s\mathbf{R}\mathbf{v} + \mathbf{t} \tag{4}$$

$$T(\mathbf{v}) = \mathbf{R}\mathbf{v} + \mathbf{t} \tag{5}$$

In both equations, $T(\mathbf{v})$ stands for a transformation acting on a vector $\mathbf{v}_{D\times 1}$, where $\mathbf{R}_{D\times D}$ is a rotation matrix, $\mathbf{t}_{D\times 1}$ is a translation vector and, in Equation (4), s is a scaling parameter. D is the dimension of the point sets.

There are a lot of algorithms proposed to perform this task. Among the most widely used are the *Iterated Closest Point* (*ICP*) (Besl and McKay, 1992; Zhang, 1994) and *Coherent Point Drift* (*CPD*) (Myronenko and Song, 2010) algorithms. The former iteratively assigns correspondence based on the closest distance criterion and finds the least-squares rigid transformation relating the two point sets. The algorithm then redetermines the correspondences and continues until it reaches the local minimum. Although this is a very popular method due to its simplicity and low computational complexity, *ICP* requires the initial position of the two point sets to be adequately close. The later, on the other hand, formulates the registration as a probability density estimation problem, where one point set is represented using the *Gaussian Mixture Model* (GMM), and the other point set seen as an observation of that GMM. The correspondence is achieved by maximization of the likelihood. Since this is a more recent approach and, in this case, there is no information regarding initial closeness, *CPD* is expected to be more accurate and thus, it is the algorithm that will be further explained. For additional information on *ICP* refer to Besl and McKay (1992) and Zhang (1994).

COHERENT POINT DRIFT ALGORITHM

Is important to clarify how this algorithm operates but do not forget that the aim of this dissertation is not to reproduce the classic process but rather its quantum version. As this algorithm will not have a "quantum version", as explained in section 4, consider an example extracted from Khallaghi (2017), where all further details can be found in Myronenko and Song (2010)'s work.

Assume there are two point sets X and Y. The unknown transformation is a rotation around the origin (parameterized by θ in a rotation matrix R) followed by a translation (parameterized by t). The first step of this algorithm is assignment of correspondences between points. If the correspondences are known the solution is given by the following, also known as the orthogonal Procrustes problem (Schönemann, 1966):

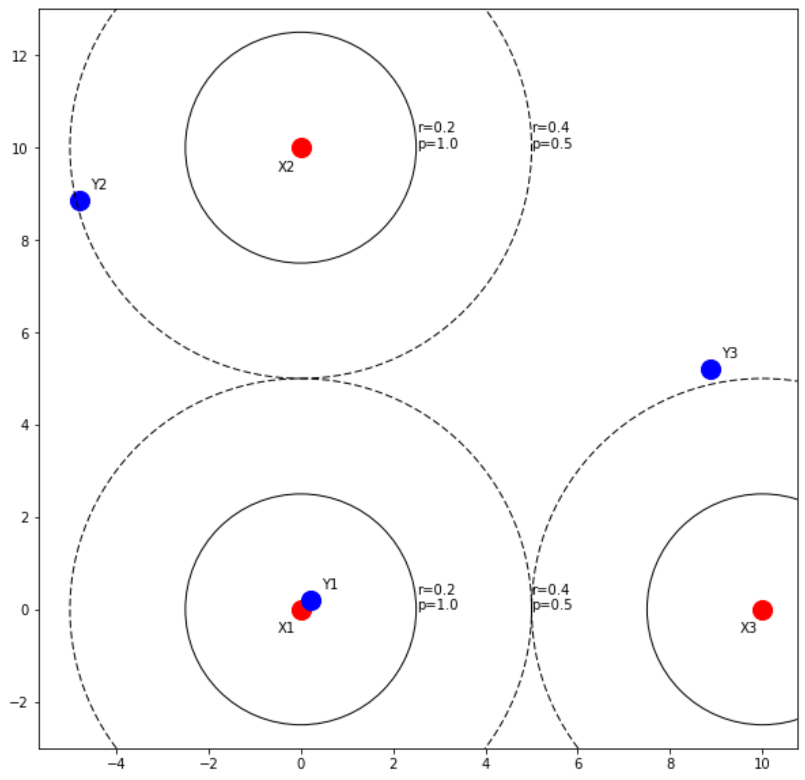


Figure 25: Example of probability distribution based on proximity extracted from Khallaghi (2017). Inner circle has a radius of 0.2 units and an assigned probability of 1. The outer circle has a radius of 4 units and corresponding probability of 0.5.

$$\text{argmin}_{R,t} \|X - RY - t\|^2, \text{ subject to } R^T R = I \tag{6}$$

When the correspondence is not explicitly known, point set registration algorithms assumes that correspondence can be inferred through point proximity. Based on this principle, it is possible to assign arbitrary correspondence probability to point sets based on proximity, as depicted in Figure 25. Even though this approach is quite simple, it is highly used due to two distinct advantages. First, it allows to assign correspondences so that the registration can be solved solely as a Procrustes problem. Furthermore, it also allows to weight the certainty function according to the correspondence probability. CPD comes in handy when the proximity is not adequate to make the correspondences. *How so?*

Instead of dealing with X, Y point sets directly, construct a GMM from the moving point cloud, Y, and treat X as observations from that GMM. Figure 26 depicts a GMM where the three Gaussians have a variance of 0.75 units. Blue points, *i.e.* Gaussian centroids, are the transformed moving points (Y). Isocontours represent the log-likelihood that red points (X point cloud) are sampled from this GMM.

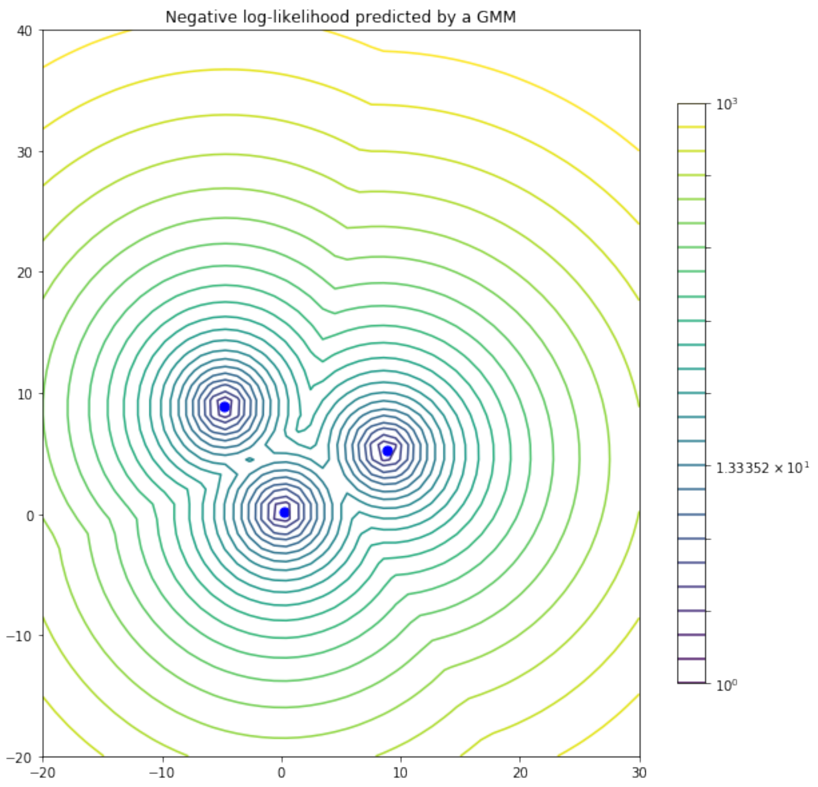


Figure 26: GMM extracted from Khallaghi (2017), based on example depicted in Figure 25, where three Gaussians have a variance of 0.75 units.

In this case, in order to perform registration, the point correspondence and the moving point set transformation problems need to be solved simultaneously. This is done through *expectation-maximization* (EM) optimization, which can be divided in two steps. To solve the correspondence problem, the Gaussian from which the observed point cloud was sampled needs to be found (E-step). This provides the correspondence probability, similar to Figure 25. Once correspondences probabilities are known, the maximization of the negative log-likelihood that the observed points were sampled from the GMM is performed with respect to transformation parameters (M-step).

In Figure 26, if there was only one Gaussian component in the mixture, then the probability that a point x is sampled from this Gaussian is given using the probability density distribution of the multivariate normal distribution. For the 2D case, with isotropic Gaussians, this simplifies to:

$$p(X) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{\|X - RY - t\|^2}{2\sigma^2} \right\}$$

However, since multiple Gaussians are in place, this probability needs to be normalized by the contribution of all Gaussian centroids. This is already accomplished and all taken into account in Khallaghi (2017)'s work, *PyCPD*.

Coming back to the problem, because in this *pre-alignment* there is no need to consider the types of the points to align but just their geometric proximity, the **pre-alignment can be reached with a common computer vision algorithm** that only considers Cartesian coordinates of a set of points in order to reach a greater proximity, such as the one described so far.

With this in mind, it was used a slightly modified version of the *pyhton* code documented in Khallaghi (2017) and available here[2]. The modified version can be found here[3] and in appendix A.4.1. The algorithm per se was not altered, only some minor aspects and parameters were adjusted.

The first obvious change lays on the definition of rigid transformation followed by Myronenko and Song (2010) and subsequently Khallaghi (2017), as already illustrated through formulas (4) and (5). Due to this difference, it was necessary to eliminate the scaling factor from the algorithm, which was accomplished quite easily by setting the scale parameter always to 1 on each iterative update of parameters. Note, moreover, that a scaling factor cannot be considered because molecules do not shrink or grow.

Unfortunately, with this adjustment came an unpredictable error, also incomprehensible by the original author of the code[4]. It generated an error regarding convergence when the tolerance was set lower than 0.5, value found by exhaustive search. The value of tolerance is the interval of error between point sets proximity accepted as minimum in order to reach convergence. In the original version this tolerance is preset to 0.001, which was clearly impossible to reach in this case study.

A possible reason for this is the no similarity between point sets, that is, between molecules. Since the points usually used in this type of problem are expected to be parts of the same image/object, it is expected as well that a near perfect correspondence would be possible, which in this case study is not. The tolerance, therefore, can not be very low. Initially the value tested was 1, by exhaustive search of intervals as low as 0.01, the minimum value allowed to make possible all transformations was 0.5. The change was as well implemented within the code. It was tested several times whether such tolerance was enough to observe any changes within the molecules proximity, and such was always found.

To describe the last adjustment, one needs to understand how the algorithm was used. To do so, look at some of the code (given complete in appendix A.3).

```
def transformation(*confs):
```

2 `https://github.com/siavashk/pycpd`
3 `https://github.com/msofiasoliveira/MasterDissertation/blob/master/pycpd_altered.zip`
4 Contact was established with the author in the name of *Sofia Oliveira* on his blog, `http://disq.us/p/223g8ia`

```
    # X and Y are lists with coordinates of 2 different conformations to "align"
    if (len(confs)==2):
        [c1,t1], [c2,t2] = convertTolist(confs[0]), convertTolist(confs[1])

        reg1 = rigid_registration(**{ 'X': c1, 'Y': c2 })
        TY1, _, err1 = reg1.register() #the new closest coordinates of Y

        reg2 = rigid_registration(**{ 'X': c2, 'Y': c1 })
        TY2, _, err2 = reg2.register() #the new closest coordinates of Y

        if(err1 < err2):
            result = unconvert(c1,t1), unconvert(TY1,t2)
        else:
            result = unconvert(TY2,t1), unconvert(c2,t2)

    if (len(confs)==3):
        [c1,t1], [c2,t2], [c3,t3] = convertTolist(confs[0]),
                                    convertTolist(confs[1]),
                                    convertTolist(confs[2])

        nc1, nc2, nc3 = besterror(c1,c2,c3)

        result = unconvert(nc1,t1), unconvert(nc2,t2), unconvert(nc3,t3)

    return result
```

Please focus on function *transformation*, which receives as input a list of conformations with the corresponding points in the object structure mentioned in Figure 22. In this case study, because there are only three molecules to align, there are two possibilities. To align two molecules at once, or to align three molecules at once. Focus on the first case since the process behind it will mostly apply to the other one.

Initially, it is executed a conversion of the list of points of each conformation into two separate lists, one to store the Cartesian points and the other the corresponding types. This is necessary because the *PyCPD* is expecting a list of only Cartesian points. Afterwards, the rigid registration is executed having in mind one conformation as the reference, that is, as an observation of the GMM, which is always given as the first argument (X), and the other as the moving GMM, always given as the second argument (Y).

Ideally, references should not be considered when making alignments because, as previously stated, they introduce a bias. Since solving this problem was not the focus of this dissertation, it was chosen the best reference but no further enhancements were made.

It is possible to see that two transformations are made, one to each different conformation taken as reference. Ideally, and if one were tackling a problem of computer vision, both would converge to the same error level because it would be dealing with the same image at different spatial locations. Such is not the case. The different conformations from different molecules can, or cannot, have actual spatial resemblances. Remember this algorithm is just an approximation and not the actual alignment. On that account, these two transformations are performed and the one with the lowest error, that is, with the less overall distance between points is chosen to be the best transformation. The new coordinates of the Y conformation on such transformation will be taken into consideration from now on.

The last and only "visible" aspect changed in the original *PyCPD* is the return of such error. This error is not originally given as output although it is always calculated until convergence.

Although all three adjustments were already mentioned, let us continue with the code analysis. After the error check to see which reference corresponded to the best transformation, it is reversed the initial data conversion to the initial structure that gathers data into one set (with both Cartesian coordinates and string types). The object of Class Point, depicted in Figure 22, is used throughout the code except in this case. After this conversion, data is sent back in its original structure but with the transformation applied.

If there were three sets to align, one would expected the process to be a lot more complex. In appendix A.3 it is possible to see all necessary iterations to check for the best error. Nine different transformations were done in order to find the best (minor) error. Still, the process documented for the two-set case is similar to the one used in the three-set case and this scales to higher numbers of sets. This method is not very efficient, its execution time grows exponentially with the number of sets to align. Nevertheless, it is never too much to remind that the aim of this dissertation is not this algorithm development but rather the quantum algorithms that will still be addressed. The goal of this (classical) component is simply to obtain the data necessary to solve the problem via a quantum process, which indirectly required this extra development.

5.2 SAT SOLVER

The next phase, implementation of restrictions, resorts to the application of an algorithm to every graph (matrices representing each possible solution) in order to choose the right connections between every point. Since these are binary restrictions that allow, or not, an alignment to occur, they can easily be implemented with a common SAT solver (or such is expected to). Further on, it will be possible to see that, because there is a weight associated with some restrictions, not all SAT solvers can be applied. Notwithstanding, all events will be document as they occur.

$1H_1$	database[0][#][0]	$2H_1$	database[1][#][0]	$3H_1$	database[2][#][0]
$1H_2$	database[0][#][1]	$2H_2$	database[1][#][1]	$3H_2$	database[3][#][1]
$1H_3$	database[0][#][2]	$2H_3$	database[1][#][2]	$3H_3$	database[3][#][2]
$1H_4$	database[0][#][3]	$2P_1$	database[1][#][3]	$3H_4$	database[3][#][3]
$1H_5$	database[0][#][4]	$2P_2$	database[1][#][4]	$1P_1$	database[3][#][4]
$1H_6$	database[0][#][5]				
$1P_1$	database[0][#][6]				

Table 1: Relations between notation used in Figures 19 and 28 and *pyhton* code.

Recall, through Figure 22 and Table 1, how the data structure documented in Figure 19 is implemented in the pyhton code.

As already stated in section 4.2, each possible solution is a graph, represented by a N-dimensional matrix and each matrix position, as easily observed in Figure 19, represents the interaction between N points, one from each conformation (set of points). Each interaction can be existent (points aligned) or non-existent (points not aligned), depending on the verification of the restriction, which, as previously stated in section 4, is:

1. Two points can only be aligned if they have the same chemical feature.
2. Each point can only be superposed to one and only one point from each conformation.
3. Two points can only be considered aligned if they are at a maximum distance of 2 Å.

In order to implement these in a SAT solver, the restrictions need to be elaborated, as will explained in the sequel.

SAME CHEMICAL FEATURE AND MAXIMUM DISTANCE

The process is straight forward: One only needs to, first, compare the string variable from each N points to align and, second, decide on the distance calculation method to be used.

The first operation is done by a Boolean comparison between all string variables. This was implemented with function *sameType(*Point)* which receives as input a variable number of *Point* objects. It was also implemented an error exception regarding the number of points to be compared, where in case those are less then two, sends an error message.

```
def sameType (*points):
    result = True
    type_data = ""

    if(len(points)<2):
        raise Exception('Number of points to be compared should be at least 2.
                         The number of points was: {}'.format(len(points)) )

    for i in points:
        if (type_data == ""):
            type_data = i.data

        result = result and (type_data==i.data)

    return result
```

The second operation results to the definition of Euclidean distance in the distance calculation.

Definition 5.1 *Given two points in an Euclidean 3-space,* $\mathbf{p} = (x_1, y_1, z_1)$ *and* $\mathbf{q} = (x_2, y_2, z_2)$*, in Cartesian coordinates, the distance between* $\mathbf{p}$ *and* $\mathbf{q}$ *is given by the formula:*

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

In *python*, given two *Point* objects, the distance calculation is done by function *dist (Point, Point)* as follows:

```
def dist ( PointA, PointB ):
    xa = PointA.x
    xb = PointB.x
    ya = PointA.y
    yb = PointB.y
    za = PointA.z
    zb = PointB.z

    final = m.sqrt( (xa-xb)**2 + (ya-yb)**2 + (za-zb)**2 )

return final
```

Since the distance between three points is also relevant, is was created a function, *distances(*Points)*, that joins both previous functions. Given a set of points, it checks whether all are of the same type, and if so is computes the distance between all individual non-repetitive pairs of points. In case of all the pair-distance being inferior or equal to 2Å, it returns the distance as the sum of all their pair-distance. Otherwise returns 0.

```
def distances (*points):
    value = 0
    pairs = [] #list with the distances between each different pairs

    if sameType(*points):
        for pair in comb(points,2):
            pairs.append(dist(pair[0],pair[1]))

    if (all(dists<=2 for dists in pairs )):
        value = sum(pairs)

    return value
```

(All codes are available in the link present in appendix A.3.)

These functions were created as auxiliary functions to the used in the SAT solver, because they produce relevant data to the restriction itself.

alignment between one point from each conformation

The implementation of this restriction can be more complex. Using the same example of Figure 19, this restriction demands that given all, but one, elements in every straight "direction" viewed in a determined position needs to be zero. As an example, Figure 27 depicts in red such elements viewed from the position $(1H_2, 2H_3, 3H_1)$.

Thus, look at Figure 27, and imagine that instead of a possible zero or integer in every position, it is zero or one, that is, instead of C being an integer, it is a 1. The restriction, is given by the following calculation, where $N_{moleculex}$ is the number of points of conformation x:

For a fixed:

$$i \in [0, N_{molecule1}[, \qquad \sum_{j}^{N_{molecule2}} \sum_{k}^{N_{molecule3}} elems[i][j][k] <= 1 \tag{7}$$

$$j \in [0, N_{molecule2}[, \qquad \sum_{i}^{N_{molecule1}} \sum_{k}^{N_{molecule3}} elems[i][j][k] <= 1 \tag{8}$$

$$k \in [0, N_{molecule3}[, \qquad \sum_{i}^{N_{molecule1}} \sum_{j}^{N_{molecule2}} elems[i][j][k] <= 1 \tag{9}$$

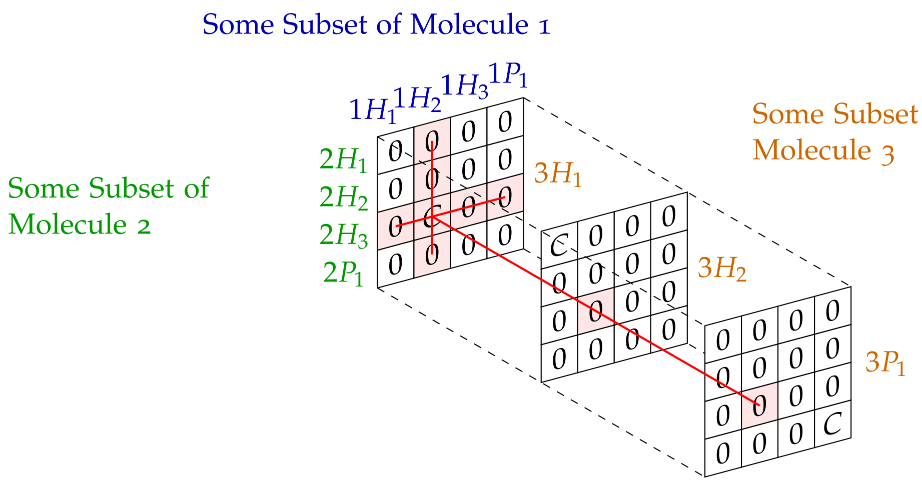

Some Subset of Molecule 1
$1H_1$
$1H_2$
$1H_3$
$1P_1$
Some Subset of Molecule 2
$2H_1$
$2H_2$
$2H_3$
$2P_1$
Some Subset of Molecule 3
$3H_1$
$3H_2$
$3P_1$

Figure 27: 3-Dimensional Matrix of a possible solution after evaluation of which points are connected. Same example as depicted in Figure 19. Here, the red lines and red filled squares represent the elements one needs to check to see whether only one point from each conformation is simultaneously aligned or not. ©SofiaOliveira, 2020

Since the goal is to create binary restrictions, take two matrix as decision variables. One, *elems*, is based on the assumption that each position is zero or one, either the points are non-align or align, respectively. The other, *dists*, follows the initial approach depicted in section 4.2. The key in this SAT implementation is, therefore, to implement all restrictions on the *elems* variable, and force *dists* to only have a value where *elems* has a 1. Such was necessary in order to force *elems* to be maximized and *dists* to be minimized. Firstly, take a look at the sketch of the SAT solver.

SCIP

```
def solve(conf1,conf2,conf3):

aviso = 0

cmol1 = conf1 # molecule 1 conformation
cmol2 = conf2 # molecule 2 conformation
cmol3 = conf3 # molecule 3 conformation

size = pmol1*pmol2*pmol3  #number of elements of matrix solution

aloc  = VarArray(size,2) # array with 1 or 0
alocs = Variable(0,size+1) # sum of elements !=0
dists = VarArray(size,0,601) # array with elements (distances)
distss= Variable(0,size*601) # sum of elements (distances)
```

```
def X(p1,p2,p3):
    return aloc[p1 + pmol1*( p2 + pmol2*p3)]

def D(p1,p2,p3):
    return dists[p1 + pmol1*( p2 + pmol2*p3)]

solucao = Model()

for id1, p1 in enumerate(cmol1):
    for id2, p2 in enumerate(cmol2):
        for id3, p3 in enumerate(cmol3):
            value = distances(p1,p2,p3)

            if ( not(sameType(p1,p2,p3)) or value==0 ):
                solucao.add(( (D(id1,id2,id3)==0) & (X(id1,id2,id3)==0) ))
            else:
                solucao.add(( ( (D(id1,id2,id3)==0) & (X(id1,id2,id3)==0) ) |
                              ( (X(id1,id2,id3)==1)
                               & (D(id1,id2,id3) == 100*(m.sqrt( (p1.x-p2.x)**2
+ (p1.y-p2.y)**2 + (p1.z-p2.z)**2 ) + m.sqrt( (p1.x-p3.x)**2 + (p1.y-p3.y)**2 +
(p1.z-p3.z)**2 ) +  m.sqrt( (p2.x-p3.x)**2 + (p2.y-p3.y)**2 + (p2.z-p3.z)**2) ) )
                                 )
                           ))

for p1 in range(pmol1):
    solucao.add( Sum([ X(p1,p2,p3) for p2 in range(pmol2)
                                   for p3 in range(pmol3)]) <=1 )

for p2 in range(pmol2):
    solucao.add( Sum([ X(p1,p2,p3) for p1 in range(pmol1)
                                   for p3 in range(pmol3)]) <=1)

for p3 in range(pmol3):
    solucao.add( Sum([ X(p1,p2,p3) for p1 in range(pmol1)
                                   for p2 in range(pmol2)]) <=1)

solucao.add( alocs == Sum([ X(p1,p2,p3) for p1 in range(pmol1)
                                            for p2 in range(pmol2)
                                            for p3 in range(pmol3)]) )

#relevant pharmacophore has to have at least 3 common groups
solucao.add( alocs>=3 )

solucao.add( Sum([ D(p1,p2,p3) for p1 in range(pmol1)
                               for p2 in range(pmol2)
                               for p3 in range(pmol3)]) == distss )

solucao.add( distss>0 )

solucao.add(Minimize(distss))
solucao.add(Maximize(alocs))


if solucao.load('SCIP').solve():
    aviso=0
else:
    aviso = 1
    print("No solution")

return [aviso,dists]
```

In a first attempt, the SAT model was implement using *SCIP*, where all variables need to be in list (array) format. Here, *elems* is given by the name *aloc* and *alocs* is its respective sum. *dists* is given by the same name and *distss* is the respective sum. In this SAT solver such "duplicated" variables where necessary because it was not possible to optimize the sum of a decision variable, it demanded the optimization to be directly applied to a decision variable. To simplify, it was defined functions $X(p1, p2, p3)$ and $D(p1, p2, p3)$, that give direct access to lists *alocs* and *dists*, respectively, given the matrix position.

A relevant aspect of this implementation is the use of integer lists. This is important because the distance between points returns a float variable, but such variable is not accepted and unnecessarily complicates the SAT solver. In order to avoid such excess of information, it was used the multiplication of every distance calculations by 100 in order to obtain an *integer* with three digits with minimum value of 0 and maximum of 600 (as easily calculated, because each pair-distance between 3 points can only be 2Å in order to respect the restriction, and so, the maximum distance between 3 points can only be 6).

Another limitation of this model was the impossibility of assigning an external integer to a decision variable. That is, when using the previously defined function *distance (*Point)*, despite all attempts[5], it was not possible the give the result as the value of an imposed restriction. The alternative was to calculate the distance directly in the restriction:

```
solucao.add(( ( (D(id1,id2,id3)==0) & (X(id1,id2,id3)==0) ) |
       ( (X(id1,id2,id3)==1) & (D(id1,id2,id3) == 100*(m.sqrt( (p1.x-p2.x)**2
+ (p1.y-p2.y)**2 + (p1.z-p2.z)**2 ) + m.sqrt( (p1.x-p3.x)**2 + (p1.y-p3.y)**2 +
(p1.z-p3.z)**2 ) +  m.sqrt( (p2.x-p3.x)**2 + (p2.y-p3.y)**2 + (p2.z-p3.z)**2) ))
       )
           ))
```

Here is also perceptible the implication of *elems* (*alocs*) on *dists*. That is, given a position, if $X(i, j, k) = 0$, then $D(i, j, k) = 0$, and whenever $X(i, j, k) = 1$, $D(i, j, k)$ becomes an integer which is the calculated distance between such points.

The last aspect of the *SCIP* solver, and obviously common to any other solvers, was the impossibility to simultaneously minimize a decision variable and maximize another that depended on the first, as it happens in this case. A solution was attempted by not forcing a maximization of the sum of *elems* (*alocss*) but instead, force it to be a value equal or greater than 3. Such value, 3, was chosen after discussion with BIAL, which stated that a relevant pharmacophore should always have more than 3 groups in common with the ligands used.

The aspect that forced the change of the SAT solver was the impossibility to run, even for only one time, the required number of calculations and restrictions[6] in acceptable time. The computer crashed every time real data was used. Due to that reason, another solver was used, *Gurobi*, which, luckily, could perform all this restrictions in a non-relevant time.

GUROBI

As a non-expected side effect, all bad aspects previously stated were cleared with the use of *Gurobi*. In order to not extend too much on this topic, check the following implementation of the solver.

```
def solve3(conf1,conf2,conf3,alignN):

aviso = 0

pmol1 = len(conf1)
```

5 Which included: use directly the result as the value of an upper restriction; attribute the result to a variable and give the variable as the value of an upper restriction; separate the type and distance restrictions and try to implement them externally; based on the last, simply use function *dist(Point, Point)* instead of *distance (*Point)* and retry the first two attempts.

6 more than 500, according to *SCIP*.

```
pmol2 = len(conf2)
pmol3 = len(conf3)

cmol1 = conf1 # molecule 1 conformation
cmol2 = conf2 # molecule 2 conformation
cmol3 = conf3 # molecule 3 conformation

size = pmol1*pmol2*pmol3  #number of elements of matrix solution
toReturn = np.zeros(size)

try:
    m = Model("trial")
    m.Params.OutputFlag = 0
    elems = m.addVars(pmol1,pmol2,pmol3,vtype=GRB.BINARY) #array with 1 or 0
    dists = m.addVars(pmol1,pmol2,pmol3,lb=0.0,ub=60001.0,vtype=GRB.INTEGER)
    # array with elements (distances)

    for id1, p1 in enumerate(cmol1):
        for id2, p2 in enumerate(cmol2):
            for id3, p3 in enumerate(cmol3):

                value = round(10000*distances(p1,p2,p3),0)

                if ( not(sameType(p1,p2,p3)) or value == 0 ):
                    m.addConstr( elems[id1,id2,id3] == 0 )

                m.addConstr( (elems[id1,id2,id3] == 0) >>
                                             (dists[id1,id2,id3] == 0.0) )
                m.addConstr( (elems[id1,id2,id3] == 1) >>
                                             (dists[id1,id2,id3] == value) )

    for p1 in range(pmol1):
        m.addConstr( sum( elems[p1,p2,p3] for p2 in range(pmol2)
                                          for p3 in range(pmol3)) <=1 )

    for p2 in range(pmol2):
        m.addConstr( sum( elems[p1,p2,p3] for p1 in range(pmol1)
                                          for p3 in range(pmol3)) <=1 )
    for p3 in range(pmol3):
        m.addConstr( sum( elems[p1,p2,p3] for p1 in range(pmol1)
                                            for p2 in range(pmol2)) <=1 )

    #relevant pharmacophore has to have at least 3 commun groups
    m.addConstr( elems.sum() >= alignN )

    m.setObjective( elems.sum(), GRB.MAXIMIZE)
    m.setObjective( dists.sum(), GRB.MINIMIZE)

    m.optimize()

    aviso = 0

    if (m.status == GRB.Status.OPTIMAL):
        solD = m.getAttr('X', dists)
        for i in range(pmol1):
            for j in range(pmol2):
                for k in range(pmol3):
                    toReturn[i + pmol1*( j + pmol2*k)] = solD[i,j,k]
    elif (m.status == GRB.Status.INFEASIBLE):
        aviso=1

except GurobiError as e:
    print('Error code ' + str(e.errno) + ": " + str(e))
    aviso = -1

except AttributeError:
    aviso = -1
    print('Encountered an attribute error')

return [aviso,toReturn]
```

The only additional procedure, that was not previously required in SCIP, was the execution of the model several times. This was necessary, because, as previously stated, is impossible to minimize a decision variable and maximize another that depend on each other. The proposed solution with *SCIP* was to force the sum of *elems* to be a value equal or greater than 3. As the distance was being minimized, no value greater than 3 would ever be found, and so, the brute force solution found was to run the model several times, each time for a value greater than the previous until it reaches a value of impossibility to find a solution.

Another aspect taken into consideration for this solver, but not for the previous one, was the conversion of the matrix used as decision variable (which was possible in this model) to a list. That was done in order to make easier to perform several calculations, necessary in the iterative step of solution choice. With the same goal, and in order to achieve better precision in calculations, integer values of up to five digits were used (multiplication by 10.000 in every distance calculation, instead of the previous 100).

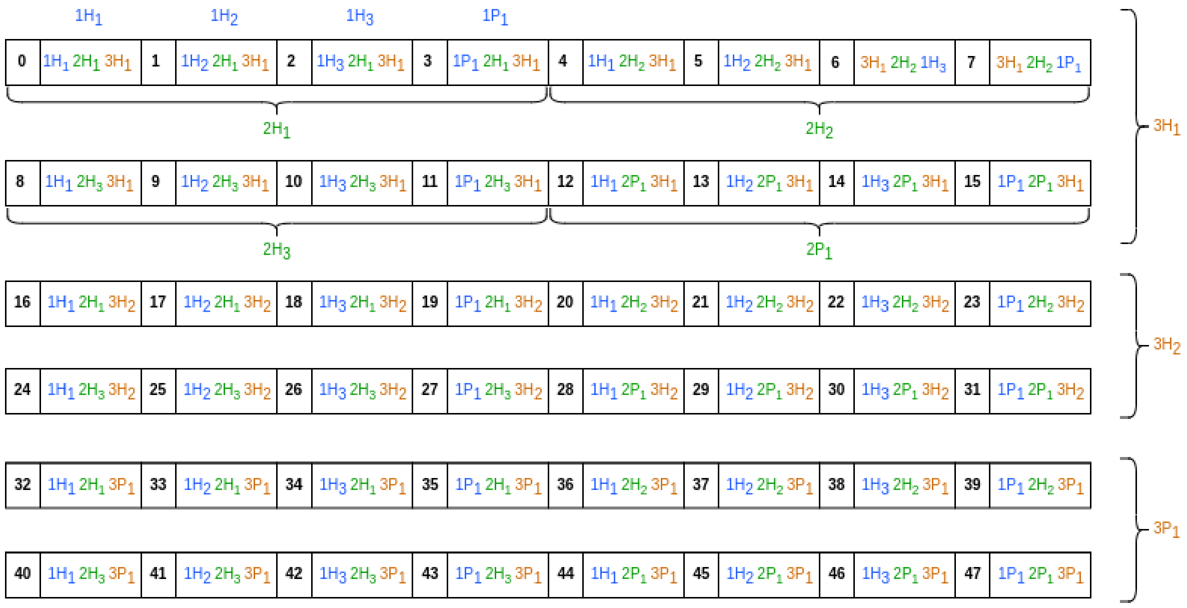
5.3 SCORES EVALUATION AND SOLUTION CHOICE

For this last step, in order to simplify some pre-search steps and because the future quantum approach also requires it, each of the solution matrices in the code is converted to a list of size $\sum_i^N K_i$ where K_i is the number of points of a conformation from molecule i. In this particular case and because there are 3 molecules ($N = 3$), one with conformations of seven points and the other two with conformations of 5 points, this means that when considering the alignment between these three molecules, there will be a list of size 175 ($7 \times 5 \times 5$) that represents each solution. Since there are twelve conformations from each molecule, in total, there are 1728 different solutions, each represented by a 175-length list. Figure 28 illustrates the list format for each solution using the same example data and notation used in Figure 19 and the respective correlations between the *pyhton* code, presented in Table 1.

The use of a list came with the need to simplify and prepare the data to be treated in the search algorithm and also later in the quantum approaches. Is then important to see how the correlation between matrix and list is done since both notations are used. To do that, one simply needs to use the following relationship to find the correspondence between matrix positions and list positions:

$$\text{matrix}[i][j][k] = \text{list}[i + P_1 \times (j + P_2 \times k)]$$

Indexes i, j, k specify the position of the point from molecule 1, 2 and 3 to align, respectively, P_1 stands for the total number of points from the molecule in the first position and P_2 for the number of points from the molecule in the second position. In this case, the order matters, molecule 1 is the first molecule to appear and 2 the second. When considering the alignment between only two molecules, the relationship simplifies to:



$1H_1$ $1H_2$ $1H_3$ $1P_1$
0 $1H_1\ 2H_1\ 3H_1$ 1 $1H_2\ 2H_1\ 3H_1$ 2 $1H_3\ 2H_1\ 3H_1$ 3 $1P_1\ 2H_1\ 3H_1$ 4 $1H_1\ 2H_2\ 3H_1$ 5 $1H_2\ 2H_2\ 3H_1$ 6 $3H_1\ 2H_2\ 1H_3$ 7 $3H_1\ 2H_2\ 1P_1$
$2H_1$ $2H_2$
8 $1H_1\ 2H_3\ 3H_1$ 9 $1H_2\ 2H_3\ 3H_1$ 10 $1H_3\ 2H_3\ 3H_1$ 11 $1P_1\ 2H_3\ 3H_1$ 12 $1H_1\ 2P_1\ 3H_1$ 13 $1H_2\ 2P_1\ 3H_1$ 14 $1H_3\ 2P_1\ 3H_1$ 15 $1P_1\ 2P_1\ 3H_1$
$2H_3$ $2P_1$
$3H_1$
16 $1H_1\ 2H_1\ 3H_2$ 17 $1H_2\ 2H_1\ 3H_2$ 18 $1H_3\ 2H_1\ 3H_2$ 19 $1P_1\ 2H_1\ 3H_2$ 20 $1H_1\ 2H_2\ 3H_2$ 21 $1H_2\ 2H_2\ 3H_2$ 22 $1H_3\ 2H_2\ 3H_2$ 23 $1P_1\ 2H_2\ 3H_2$
24 $1H_1\ 2H_3\ 3H_2$ 25 $1H_2\ 2H_3\ 3H_2$ 26 $1H_3\ 2H_3\ 3H_2$ 27 $1P_1\ 2H_3\ 3H_2$ 28 $1H_1\ 2P_1\ 3H_2$ 29 $1H_2\ 2P_1\ 3H_2$ 30 $1H_3\ 2P_1\ 3H_2$ 31 $1P_1\ 2P_1\ 3H_2$
$3H_2$
32 $1H_1\ 2H_1\ 3P_1$ 33 $1H_2\ 2H_1\ 3P_1$ 34 $1H_3\ 2H_1\ 3P_1$ 35 $1P_1\ 2H_1\ 3P_1$ 36 $1H_1\ 2H_2\ 3P_1$ 37 $1H_2\ 2H_2\ 3P_1$ 38 $1H_3\ 2H_2\ 3P_1$ 39 $1P_1\ 2H_2\ 3P_1$
40 $1H_1\ 2H_3\ 3P_1$ 41 $1H_2\ 2H_3\ 3P_1$ 42 $1H_3\ 2H_3\ 3P_1$ 43 $1P_1\ 2H_3\ 3P_1$ 44 $1H_1\ 2P_1\ 3P_1$ 45 $1H_2\ 2P_1\ 3P_1$ 46 $1H_3\ 2P_1\ 3P_1$ 47 $1P_1\ 2P_1\ 3P_1$
$3P_1$

Figure 28: List organization of a solution based on the same notation and example of Figure 19. ©SofiaOliveira, 2020

$$\text{matrix}[i][j] = \text{list}[i + P_1 \times j]$$

Now that the conversion from matrices to lists is clear, it is important to show its usefulness. As integer variables were used, the values of distance are returned multiplied by 10.000, as already stated. In order to see if such calculations are correct, ones needs to compare them with real values obtained visually with other reliable software. To do such comparison, and because it is easier to apply an operation to every element of a structure through lists, instead of matrices, due to several pre-existent mechanisms, such as lambda operations, the return of the SAT solver is done in list format so that one can easily revert the 10.000$\times$ multiplication. After that being done, all results are reconverted to matrix structure. Now, back to this more intuitive structure, let us dive in the score process.

In a first attempt, because there are two important parameters to consider in a possible solution, according to all restrictions implied, the approach was to count the number of non-zero elements of the solutions (nz), as well as their respective sum, and divide by nz, s, as the following code illustrates.

```
def NonZeroAndSum(matrix):
    nz=0
    s=0
    for idC1, c1 in enumerate(matrix):
        for idC2, c2 in enumerate(c1):
            if isinstance(c2, (np.ndarray)): #tests if this is an array or not
                for idC3, c3 in enumerate(c2):
                    if (c3!=0):
                        s += c3
                        nz += 1
            else:
                if (c2!=0):
                    s += c2
                    nz += 1

    if (nz!=0): #sum is divided by the number of points aligned
        s = s/nz

    return [nz,s]
```

After this calculation, the solution is chosen by means of the algorithm below, which, having as the most important factor the number of points aligned (*nz*), chooses the solution with the smallest distance with that particular highest value of *nz* found.

```
def answer (listmatrix):
    bestNZ = 0
    bestSum = 100000 # highest number
    solution = 0

    for idS, pSolution in enumerate(listmatrix):
        #(number of non-zero elements)&(sum of this elements divided by previous)
        [nz,s] = NonZeroAndSum(pSolution)

        #parameter more relevant is the number of 'connections'
        if (nz!=0):
            if(nz==bestNZ and s<bestSum):
                bestSum = s
                solution = idS
                print(nz, s, idS)

            if(nz>bestNZ):
                bestNZ= nz
                bestSum = s
                solution =idS
                print(nz, s, idS)

    #last printed bestNZ and bestSum are the ones from the solution
    print("-------------------")
    print("Position of solution:",solution) #position in array listmatrix
    print("-------------------")

    return solution, bestSum
```

Unfortunately, as explained further on, in quantum search one cannot have two variables as the score. There can only be one, and so, an alternative needs to be found. After several attempts to find the best function capable of joining both variables in an appropriate matter, the following approach was made.

```
def answer (scores):
    bestSum = max(scores)+1 # highest number
    solution = -1

    sols = []
    vals = []

    for idS, s in enumerate(scores):
        if( s < bestSum):
            bestSum = s
            solution = idS
            print(s, idS)

    print("-------------------")
    print("Position of solution:",solution) #position in array listmatrix/scores
    print("-------------------")

    return solution, bestSum #position of solution, value of solution
```

This depends on the next score method, where the function *NonZeroAndSum*, instead of returning *s* as the sum of all pair-distances divided by *nz*, returns the strict sum of all pair-distances. The final answer is found by search of the lowest score.

```
def Scores(listMatrix,M):
```

```
scores = []

for matrix in listMatrix:
    naligned, d = NonZeroAndSum(matrix)
    if (naligned!=0):
        scores.append( int(round( (d*200/(naligned*comb2(M,2))) +
                                  (5-naligned)*400 )) )
    else:
        scores.append(int(2000))

return scores
```

Although the code is based in the formula (11), this corresponds to the final attempt. The first attempt was actually with the formula (10) which, although appearing to be the most obvious function, induced error in the search algorithm because it allowed a solution to be found based on the distance solely, and not primarily on the number of points aligned.

$$\text{score}(n_{aligned}, d) = round\left(\frac{1000 \times d}{n_{aligned}}\right) \tag{10}$$

$$\text{score}(n_{aligned}, d) = round\left(\frac{200 \times d}{\binom{M}{2} \times n_{aligned}} + 400 \times (5 - n_{aligned})\right) \tag{11}$$

Although more complex than the first attempt, with this type of score function, it is impossible for the algorithm to choose a solution based only on the distance sum, preventing errors such as the ones that were found in the former. Both functions are depicted in Figures 29 and 30 respectively, in which one can see the errors that could happen when one considers the formula (10). An example: given two distinct solutions, one with three points aligned and $d = 1.3$ and another with four points aligned but with $d = 3$, according to the Figure 29, the solution with the smallest score, and hence, the chosen score would be the one with only three points aligned. However, because the number of points aligned is always more relevant than the distance, the correct smallest score should be the one with four points aligned. This is clearly obtained when using formula (11), as depicted in Figure 30.

There is, as expected, a reasoning behind the formula (11). If one considers $[0, 2000]$ as the codomain of the score function, and 5 to be the maximum set of points aligned within a given solution, one knows that, given the formula (11), there is a 400 value interval witnessing a linear growth according to the distance, d, when considering a specific number

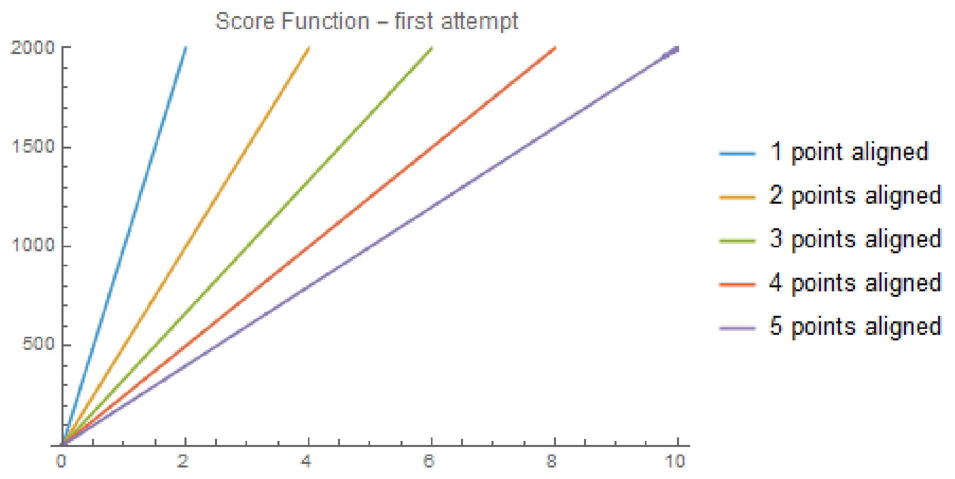


Figure 29: Graphic display of Equation (10). ©SofiaOliveira, 2020

of points aligned, as depicted in Figure 30. Another value that can be found is the 200 that is initially multiplied by the d. This value is computed having in mind the 2000 maximum value in the codomain, and hence, the 400 maximum interval for each linear dependency of points aligned. According to the data, $d/(\binom{M}{2} \times n_{aligned})$ can only, at maximum, acquire the value of 2, and so, in order to obtain a maximum of 400 with $d/(\binom{M}{2} \times n_{aligned})$, it need to be multiplied by 200. Here, $\binom{M}{2}$ represents the number of different pairs that can be generated with M elements. Since the distance between several points is calculated as the sum of all pair-distance, as stated in the beginning of this section, it is necessary to divide the distance by that number in order to obtain a maximum of 2 in the variable of interest.

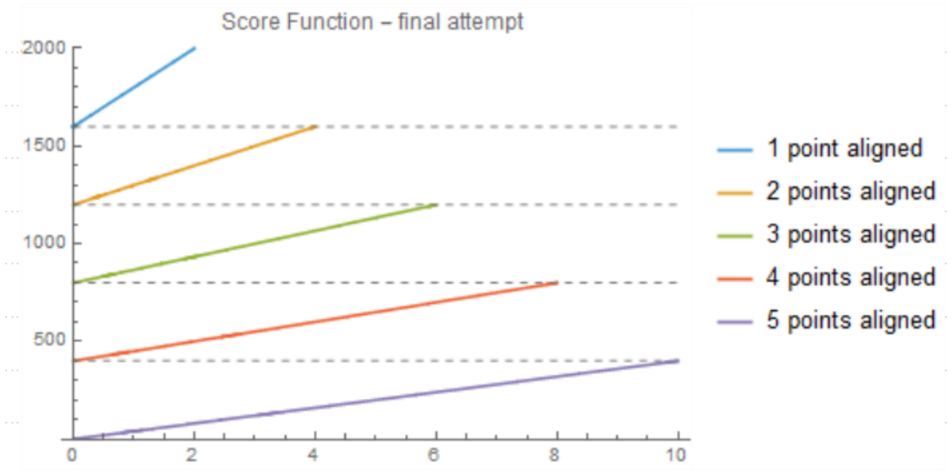


Figure 30: Graphic display of Equation (11) for $M = 2$, *i.e.*, only considering two different conformations in the alignment. ©SofiaOliveira, 2020

It is interesting to state that this last function is far for perfect since it has different precision when considering different numbers of points and has a strict codomain. This could not be a relevant aspect if one considers a bigger codomain, but that is not possible in order to be able to perform, in a quantum simulator, the corresponding quantum search. This, obviously, depends on the same score function as the classic version and the codomain could not surpass a certain maximum, dictated by the maximum number of qubits available. This explanation may sound vague, but hopefully, it will all come clear by the end of the next section, where all quantum dependencies will be explained. Another dependency based on the quantum version, is the observed *round* function present in both score functions. This is needed because floats require more space and hence, more qubits, reason for which integers will always be used in the quantum algorithms discussed in the sequel.

One last step was necessary to know exactly to which molecule's conformations does the position, returned by the *answer* function, correspond. Such is given by the following code, which depends on considering two or three structures in the alignments.

```
def which2(number,a):
    if(number>=12):
        a += 1
        return which2(number-12,a)
    else:
        print("Conformation from molecule given as input 1: ",a,
              "\nConformation from molecule given as input 2: ",number)
        return [a, number]

def which3(number,a,b):
    if (number>=144):
        a += 1
        return which3(number-144,a,b)
    else:
        if(number>=12):
            b += 1
            return which3(number-12,a,b)
        else:
            print("Conformation from molecule given as input 1: ",a,
                  "\nConformation from molecule given as input 2: ",b,
                  "\nConformation from molecule given as input 3: ",number)
            return [a, b, number]

def whichConfs (number,mol):
    #insert position of listmatrix and returns to each conformations that corresponds
    if(mol==3):
        numbers = which3(number,0,0)
    if(mol==2):
        numbers = which2(number,0)
    return numbers
```

The same process is also used in the quantum version.

6 THE QUANTUM APPROACH

This section will introduce some theoretic notions and pre-existent algorithms that can fulfill the problem needs. Afterwards, it will be explained the particular approach followed in this dissertation.

As already stated, the pre-alignment step will not have a quantum version and hence, only the SAT solver and solution search will be approached in section 6.1 and 6.2, respectively. In the former, will be revised two different techniques for implementation of restricting on quantum devices and in the latter will introduce a quantum search algorithm. Before, on both cases, the particularities of each algorithm used will be further explained.

6.1 QUANTUM SAT SOLVER

6.1.1 *Related work*

The first algorithm that seemed to qualify as a quantum algorithm for a satisfiability problem was an algorithm reported by Farhi et al. (2000), based on **adiabatic evolution**, (Refer to Albash and Lidar (2018) for more information on the subject). In this approach, the evolution of a quantum state is governed by a time-dependent Hamiltonian that interpolates between an initial Hamiltonian, whose ground state is easy to construct, H_0, and a final Hamiltonian, whose ground state encodes the satisfying assignment, H_P. To ensure that the system evolves to the desired final ground state, one must assure that the evolution time is big enough, a key aspect to make such an algorithm reach the desired solution.

As an example, imagine a 3-SAT problem with clauses C, each involving bits i, j and k. After replacing the bits by qubits, the Hamiltonian associated with all of the clauses, H_P, takes the form:

$$H_P = \sum_C H_{P,C} \tag{12}$$

where $H_{P,C}$ is a Hamiltonian associated with clause C. On the other hand, H_0, can simply be, for i, j, k:

$$H_0 = \sum_C H_{0,C} \tag{13}$$

where

$$\begin{gathered} H_{0,C} = H_0^{(i_C)} + H_0^{(j_C)} + H_0^{(k_C)} \\ H_0^{(i)} = \frac{1}{2}(1 - \sigma_x^{(i)}) \qquad \text{with} \qquad \sigma_x^{(i)} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{gathered}$$

Taking time into account, the solution can be found, by adiabatic evolution, considering Equations (13) and (12) in:

$$H(t) = (1 - \frac{t}{T})H_0 + (\frac{t}{T})H_P \tag{14}$$

See Farhi et al. (2000) for full example.

However, the real difficulty in this type of algorithm lies in the construction of a device that produces an assignment that satisfies all clauses (restrictions). Unfortunally, if one strictly follows the same type of construction made by Farhi et al. (2000), which is the direct implementation of a Boolean formula, another problem arises.

Despite being abstract, the problem still has a lot of information to be dealt with and, once again, it become an issue. Since each matrix solution has 175 ($7 \times 5 \times 5$) positions, and these need to be binary variables with a chosen value by the end of the restriction evaluation, it implies the use of 175 qubits solely for data representation. But these are not all the qubits required by this type of algorithm!

As already stated in section 5.2, in a three-molecule alignment, there are more than 500 individual restrictions to be applied in each solution. Consider that each restriction deals with, at most, 7 variables at a time (considering 7 is the highest dimension of the solution matrix). This makes the problem into a 7-SAT problem. According to Farhi et al. (2000), in a 3-SAT, each Boolean clause involves, at most, 3 bits. In the case study, each clause involves, at most, 7 bits. This number is not actually relevant because these bits are only referent to the oracle used, that is, the number of inputs the oracle will have. The problem comes with the number of auxiliary qubits required for each restriction. Considering restrictions in an almost "best-case scenario", such as having the oracle to be a toffoli gate that only requires 1 auxiliary qubit, one still requires in total, more than 500 auxiliary qubits solely for clause evaluation.

In total, by using the algorithm suggested by Farhi et al. (2000), one would need more than 675 (500 + 175) qubits, which is, of course, impossible to obtained even in the most advanced quantum device simulator currently available.

Even if one would consider the alignment of the two smallest molecules, only for the data representation, 25 (5×5) qubits would be required. Of course, the actual total number is higher considering all the auxiliary qubits one would need for each restriction implementation. Such number is much smaller than 500, but still, more than 50[1], individually greater than the number of qubits that are currently available for simulations.

The **Ising model** is another possible restriction construction for an adiabatic-based quantum algorithm that might be considered. But what is precisely an Ising model?

ISING MODEL

An Ising model is a mathematical model that can represent a variety of physico-chemical systems with nearest-neighbor interactions. This model relies on the assumption that the system can be represented by a regular lattice arrangement of particles in space where each particle can be represented by any two-valued variable (Brush, 1967). In this case study, it represents the system of qubits, a spin system, where each variable (qubit) can be represented by its spin, up or down.

According to Lucas (2014), an Ising model can be written as a quadratic function on a set of N spins $s_i = \pm 1$:

$$H(s_1, ..., s_N) = -\sum_{i<j} J_{ij} s_i s_j - \sum_{i=1}^{N} h_i s_i \tag{15}$$

The quantum version of this Hamiltonian using adiabatic evolution and thus, Equation (14), is simply

$$H_P = H(\sigma_1^z, ..., \sigma_N^z) \tag{16}$$

1 This number was not confirmed since the SCIP solver, which gave such information, was only tested in an initial stage. After some thought, it was concluded that this number would be smaller than 500 but, in any case, much greater than 50.

where σ_1^z is a Pauli matrix acting on the ith qubit in a Hilbert space of N qubits $\{|+\rangle, |-\rangle\}^{\otimes N}$, and J_{ij} and h_i are real numbers. H_0 can be

$$H_0 = -h_0 \sum_{i=1}^{N} \sigma_i^x, \tag{17}$$

so that the ground state of H_0 is an equal superposition of all possible states on the eigenbasis of H_P. For a better understanding and applications, the reader may want to check Lucas (2014), where several Ising formulations for a variety of NP problems are illustrated.

Reference Cervera-Lierta (2018) provides the documentation of an exact Ising model simulation on a quantum computer, *i.e.* an efficient quantum circuit that diagonalizes the Ising Hamiltonian and allows to obtain all eigenstates of the model by just preparing the computational basis states. This is a highly relevant and promising work that could be further explored in our case study, but because it involves complex approaches and there is an already implemented version, made available by IBM, such was not done in this work.

The IBM approach can be found in the Aqua segment of their public, commercially available material. Indeed, two distinct problem implementations are discussed in their advanced tutorials, IBM (a) and IBM (b).

The technique follows Farhi et al. (2014), Farhi et al. (2017) and Wecker et al. (2016), for both the conception and implementation of the algorithm. But the foundation of this technique is essencially set by Farhi et al. (2014). Introducing a bit of context, an algorithm based on adiabatic evolution (*Quantum Adiabatic Algorithm*, QAA) produces an optimal solution (if the time is long enough). On the contrary, Farhi et al. (2014) introduced a quantum algorithm that produces approximate solutions for combinatorial optimization problems (*Quantum Approximate Optimization Algorithm*, QAOA).

QAOA executes a Trotterized approximation to adiabatic evolution by alternation of two operators, $\mathrm{U}(H_P, \gamma)$ and $\mathrm{U}(H_0, \beta)$, where the sum of the angles (γ and β) is the total run time. Making the total time long enough, it is guaranteed that the solution can always be found as close to the ideal as desired. By using an QAA one would be forced to reach the ideal solution, if existent. In Farhi et al. (2014) is even documented a particular case where QAA fails and QAOA succeeds. In that work is also documented a variant of the original QAOA algorithm suited for cases where the search space is a complicated subset of the n bit strings, for example, finding a large independent set in a given graph.

Throughout the years optimizations to such algorithm were suggested. Wecker et al. (2016) introduces a slightly different parametrization and a different objective. Through the help of machine learning, rather than looking for a state which approximately solves an optimization problem, the goal is to find an algorithm modification that, given an instance of the maximum 2-SAT problem, will produce a state with high overlap with the optimal state.

Farhi et al. (2017) presents the strategy for programming with this algorithm, but suffices to say that one advantage of using this method compared to adiabatic approaches is that the target Ising Hamiltonian does not have to be implemented directly on the hardware, allowing the algorithm not to be limited to the connectivity of the device.

Certainly IBM has made their own optimizations but nevertheless, the base algorithm is still the QAOA. Therefore, it will not be made a tecnhical explanation but rather a deeper understanding of their final implementation. Although in IBM (a) is depicted a weighted Max-Cut problem, a simpler and perhaps easier example, IBM (b) documents the Vehicle Routing problem, which has more restrictions and hence may better explain how complex problems can be implemented.

Mathematically speaking, the vehicle routing problem (VRP) is a combinatorial problem, wherein the best routes from a depot to a number of clients and back to the depot are sought, given a number of available vehicles. (There are a number of formulations possible, the one followed is known as MTZ (Miller et al., 1960).)

Let n be the number of clients (indexed as $1, ..., n$), and K be the number of available vehicles. Let $x_{ij} = 0, 1$ be the binary decision variable which, if it is 1, activates the segment from node i to node j. The node index runs from 0 to n , where 0 is (by convention) the depot. If two nodes i and j have a link from i to j, it is said $i \sim j$. $\delta(i)^+$ denotes the set of nodes to which i has a link, *i.e.* $j \in \delta(i)^+$ if and only if $i \sim j$. Similarly, $\delta(i)^-$ denotes the set of nodes which are connected to i, in the sense that $j \in \delta(i)^-$ if and only if $j \sim i$. In addition, there is also considered continuous variables, for all nodes $i = 1, ..., n$, denoted u_i.

The VRP can be formulated as:

$$(VRP) \quad f = \min_{\{x_{ij}\}_{i \sim j} \in \{0,1\}, \{u_i\}_{i=1,...,n} \in \mathbb{R}} \quad \sum_{i \sim j} w_{ij} x_{ij} \tag{18}$$

subject to the node-visiting constraint:

$$\sum_{j \in \delta(i)^+} x_{ij} = 1, \quad \sum_{j \in \delta(i)^-} x_{ji} = 1, \quad \forall i \in \{1, ..., n\}, \tag{19}$$

the depot-visiting constraints:

$$\sum_{i \in \delta(0)^+} x_{0i} = K, \quad \sum_{j \in \delta(0)^+} x_{j0} = K, \tag{20}$$

and the sub-tour elimination constraints:

$$u_i - u_j + Qx_{ij} \leq Q - q_j, \forall i \sim j, i, j \neq 0, \quad q_i \leq u_i \leq Q, \forall i, i \neq 0.$$

In particular,

- The cost function is linear in the cost functions and weighs the different arches based on a positive weight $w_{ij} > 0$ (typically the distance between node i and node j);
- The first set of constraints enforces that from and to every client, only one link is allowed;
- The second set of constraints enforces that from and to the depot, exactly K links are allowed;
- The third set of constraints enforces the sub-tour elimination constraints and are bounded by u_i, with $Q > q_j > 0$, and $Q, q_i \in \mathbb{R}$

As already mentioned above, IBM followed the QAOA algorithm. In particular in this case, it is used the Variational Quantum Eigensolver (VQE, Peruzzo et al. (2014)). Due to the quantum circuits employed (variational forms) and inherent limited depth, the solution obtained is heuristic by nature. Now, take a look at the steps followed by IBM.

– Preparation steps:

- Transform the combinatorial problem into a binary polynomial optimization problem with equality constraints only;
- Map the resulting problem into an Ising Hamiltonian (H) for variables **z** and basis Z, via penalty methods if necessary;
- Choose the depth of the quantum circuit m. Note that the depth can be modified adaptively.
- Choose a set of controls θ and make a trial function $|\psi(\boldsymbol{\theta})\rangle$, built using a quantum circuit made of C-Phase gates and single-qubit Y rotations, parameterized by the components of $\boldsymbol{\theta}$.

– Algorithm steps:

- Evaluate $C(\boldsymbol{\theta}) = \langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle$ by sampling the outcome of the circuit in the Z-basis and adding the expectation values of the individual Ising terms together. In general, different control points around $\boldsymbol{\theta}$ have to be estimated, depending on the classical optimizer chosen.

- Use a classical optimizer to choose a new set of controls.
- Continue until $C(\boldsymbol{\theta})$ reaches a minimum, close enough to the solution $\boldsymbol{\theta}^*$.
- Use the last $\boldsymbol{\theta}$ to generate a final set of samples from the distribution $|\langle z_i|\psi(\boldsymbol{\theta})\rangle|^2$ to obtain the answer.

Although there are many parameters throughout the steps, according to IBM the difficulty of finding good heuristic algorithms bails down to the choice of an appropriate trial wavefunction. Here, and latter on the qSAT implementation, it will be considered a simple trial function of the form

$$|\psi(\theta)\rangle = [\mathrm{U}_{single}(\boldsymbol{\theta})\mathrm{U}_{entangler}]^m|+\rangle \tag{21}$$

where $\mathrm{U}_{entangler}$ is a collection of C-Phase gates (fully entangling gates), and $\mathrm{U}_{single}(\theta) = \prod_{i=1}^n Y(\theta_i)$, where n is the number of qubits and m is the depth of the quantum circuit. The motivation is that for these classical problems, this choice allows a search over the space of quantum states that have only real coefficients, still exploiting entanglement to potentially converge faster to the solution.

Now that one is familiar with both the problem definition and the algorithm, it is clear that in order to build this algorithm, one needs to construct a binary polynomial optimization with equality constraints from Equation (18). This is done, is this case, by considering cases in which $K = n-1$. In these cases the sub-tour elimination constraints are not necessary and the problem is only on the variable z. In particular, the Ising Hamiltonian can be written as:

$$\begin{aligned}(IH)\quad H = \sum_{i\sim j} w_{ij}x_{ij} + A\sum_{i\in\{1,\dots,n\}}\Big(\sum_{j\in\delta(i)^+} x_{ij} - 1\Big)^2 + A\sum_{i\in\{1,\dots,n\}}\Big(\sum_{j\in\delta(i)^-} x_{ji} - 1\Big)^2 \\ + A\Big(\sum_{i\in\delta(0)^+} x_{0i} - K\Big)^2 + A\Big(\sum_{j\in\delta(0)^+} x_{j0} - K\Big)^2\end{aligned} \tag{22}$$

where A is a big enough parameter. Each component of the formula is associated to a restriction. Explicitly, the first term refers to Equation (18), the second and third to Equation (19) and the last two terms to Equation (20).

The QP formulation of the Ising Hamiltonian needs to be deduced form this point in order to be used in the VQE. This can be done with the assignment $x_i \to (1-Z_i)/2$, where Z_i is the Pauli Z operator that has eigenvalues ±1. In the vector $\mathbf{z}$, and for a complete graph ($\delta(i)^+ = \delta(i)^- = \{0, 1, ..., i-1, i+1, ..., n\}$), the Hamiltonian can be written in matricial form as follows.

$$\min_{\mathbf{z}\in\{0,1\}^{n(n+1)}} \mathbf{w}^T\mathbf{z} + A\sum_{i\in\{1,\dots,n\}}\left(\mathbf{e}_i\otimes\mathbf{1}_n^T\mathbf{z}-1\right)^2 + A\sum_{i\in\{1,\dots,n\}}\left(\mathbf{v}_i^T\mathbf{z}-1\right)^2$$
$$+A\Big((\mathbf{e}_0\otimes\mathbf{1}_n)^T\mathbf{z}-K\Big)^2 + A\left(\mathbf{v}_0^T\mathbf{z}-K\right)^2 \tag{23}$$

that is

$$\min_{\mathbf{z}\in\{0,1\}^{n(n+1)}} \mathbf{z}^T\mathbf{Q}\mathbf{z} + \mathbf{g}^T\mathbf{z} + c, \tag{24}$$

where

$$\mathbf{Q} = A\sum_{i\in\{0,1,\dots,n\}}\Big[(\mathbf{e}_i\otimes\mathbf{1}_n)(\mathbf{e}_i\otimes\mathbf{1}_n)^T + \mathbf{v}_i\mathbf{v}_i^T\Big],$$
$$\mathbf{g} = \mathbf{w} - 2A\sum_{i\in\{1,\dots,n\}}\Big[(\mathbf{e}_i\otimes\mathbf{1}_n)+\mathbf{v}_i\Big] - 2AK\Big[(\mathbf{e}_0\otimes\mathbf{1}_n)+\mathbf{v}_0\Big]$$
$$\text{and,}\qquad c = 2An + 2AK^2.$$

The solution is found by solving Equation (23).

The problem is instantiated and solved using a class *QuantumOptimizer* that encodes the quantum approach. The same approach was followed in this dissertation so it is important to check in this example the methods that are inside the class:

`binary_representation` : encodes the problem into the Ising Hamiltonian QP;

`construct_hamiltonian` : constructs the Ising Hamiltonian in terms of the Z basis;

`check_hamiltonian` : makes sure that the Ising Hamiltonian is correctly encoded in the Z basis: to do this, it solves an eigenvalue-eigenvector problem for a symmetric matrix of dimension $2^N \times 2^N$, with $N = n(n+1)$;

`vqe_solution` : solves the problem via VQE by using the SPSA solver (with default parameters);

`_q_solution` : internal routine to represent the solution in a usable format.

Then, the steps encoding the problem as a binary formulation (IH-QP) and respective version in the Z basis are accomplished and the problem is solved via VQE using the Qiskit Aqua package.

Additionally, it is important do denote that depending on the number of qubits, the simulation can take a while; for example with 12 qubits, it takes more than 12 hours.

Coming back to the problem of this dissertation, there are two ways to approach it using Ising Hamiltonians. The first requires both SAT implementation as well as a solution search. The latter, adopted in this dissertation, only handles the SAT solver. Both options will be introduced next.

6.1.2 *Approaches*

FIRST APPROACH

The first approach would take into consideration each possible solution from all possible alignments at the same time. That is, the restriction implementation and solution search would be imposed simultaneously. To check if such is viable, one must calculate how many qubits these would require.

To facilitate, assume each alignment between two points to be an edge. As restrictions, there can only be chosen edges between points from different molecules, and all points of such edges need to be from different conformations (one from each molecule). In total, there are 20706 different edges, possible and impossible. This is calculated as the number of different combinations of two points from all points from all molecules.

$$\binom{12 \times (7+5+5)}{2} = {}^{204}C_2 = \frac{204!}{2!(204-2)!} = 20706$$

In order to find out which points from each conformation and from each molecule, make up the solution, one needs to consider one qubit for each edge. Hence, just for data representation, this requires more than twenty thousand qubits. To find the solution, one needs to find alignments between the three molecules, that is, each alignment of points is a set of three edges that fits the required criteria. In the end, the solution is a set of triples of edges.

$$\binom{20706}{3} = {}^{20706}C_3 = 1479361980320$$

There are, in total, more than 1 billion sets of 3 edges. Luckily, each of these sets does not need to be represented by its own qubits, but rather its own states. With 41 qubits there are more than 2 billion states ($2^{41} = 2199023255552$), which is more than enough for this problem!

Nevertheless, it is still a lot more qubits that one has available, and hence, this approach cannot be taken into consideration.

Of course, compared to classical computation, which would require more than 2 billion bits for this last step, this quantum approach has a clear advantage, but do not forget the previously calculated number of 20706 qubits.

SECOND APPROACH

The second approach, as previously stated, only executes restriction implementations. *But how is this done?* As before, each alignment between two points is an edge. But contrary to the first approach, there is only one possible solution considered at a time. That is, the input corresponds to one conformation from each different molecule and the output is a graph of the possible and correct connections between them. Taking that in mind, the qSAT is executed as many times as the number of different possible solutions, *i.e.* number of distinct sets of conformations (each conformation from each molecule).

Here, each edge is a qubit and the number of edges, and hence, qubits is given by:

$$\binom{7+5+5}{2} = {}^{17}C_2 = \frac{17!}{2!(17-2)!} = 136$$

	$1H_1$	$1H_2$	$1H_3$	$1H_4$	$1H_5$	$1H_6$	$1P_1$	$2H_1$	$2H_2$	$2H_3$	$2P_1$	$2P_2$	$3H_1$	$3H_2$	$3H_3$	$3H_4$	$3P_1$
$1H_1$	▨							★	★	★			★	★	★		
$1H_2$	▨	▨						★	★	★			★	★	★		
$1H_3$	▨	▨	▨					★	★	★			★	★	★		
$1H_4$	▨	▨	▨	▨				★	★	★			★	★	★		
$1H_5$	▨	▨	▨	▨	▨			★	★	★			★	★	★		
$1H_6$	▨	▨	▨	▨	▨	▨		★	★	★			★	★	★		
$1P_1$	▨	▨	▨	▨	▨	▨	▨				★	★				★	★
$2H_1$	▨	▨	▨	▨	▨	▨	▨	▨					★	★	★		
$2H_2$	▨	▨	▨	▨	▨	▨	▨	▨	▨				★	★	★		
$2H_3$	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨			★	★	★		
$2P_1$	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨					★	★
$2P_2$	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				★	★
$3H_1$	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				
$3H_2$	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨			
$3H_3$	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		
$3H_4$	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
$3P_1$	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨

Figure 31: Scheme of the edges between three molecules. As in section 5, the points are identified by the number of the molecule, type of the point and an index. The edges checked with the symbol ★ are the ones already respecting some restrictions. White squares represent all possible different edges, and grey squares are not considered, since they are the same as the ones in the upper diagonal of the scheme. ©SofiaOliveira, 2020

Although a lot smaller than the previous number of qubits necessary for data representation, it is still a very high number. Since it became obvious that considering all possible edges was too much and in any case unnecessary (because edges within the same conformation will not align), it was opted to implement that restriction along with the first restriction mentioned in section 4.2, about the type of the points. This was made because the aim of this dissertation was to obtain pratical and implementable results, which was impossible with 136 required qubits only for data representation.

In any case, instead of requiring 136, this restriction implementation reduced the number of qubits to 53, as depicted in Figure 31. Unfortunately, due to the available number of qubits, this number is still too high. Having that in mind, it was also opted to make the simulation of the alignment of the two smallest molecules, instead of three. But until discussing the simulation itself, consider the three molecule's case.

Moving on and following the steps mentioned when introduced the VRP, each restriction needs to be written algebraically and then translated to its *quadratic programming* (QP) formulation. In every restriction there will be a vector binary variable **x** that represents the edges (each taking the value 1 if that edge is aligned or 0 if not) and a vector integer variable **d** that takes the value of the distance between the points represented by each edge. Additionally, there will also be a binary variable **p** (called *possibility*) associated to every edge, which represents some restrictions and takes the following form:

$$p_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are:} \\ & \quad \text{- of the same type} \\ & \quad \text{- at a maximum distance of } 2\mathring{A} \\ & \quad \text{- from different molecules} \\ 0 & \text{otherwise} \end{cases}$$

p was created to avoid using more complex restrictions in the quantum algorithm. As an example, if one would check if every edge links different molecules in the quantum algorithm, that would involve several cycles and more complex algebraic calculations. Moreover, that would require more qubits and in the end, the classic algorithm would be unequivocally better. As will be further explained, the ideal approach to computation is the best of the two worlds, both quantum and classic. There is no need to over complicate the quantum approach that would, in the end, have no advantage over the classic one.

With that in mind, the first and third restrictions, *two points can only be aligned if they have the same chemical feature* and *if they are at a maximum distance of 2 Å*, respectively, are implemented as

$$x_{ij} \leq p_{ij} \quad \Leftrightarrow \quad p_{ij} - x_{ij} \geq 0, \qquad \forall i, j \in \text{Points} \tag{25}$$

stating that each final edge can only take a value less or equal to p. As an example, if the edge is not possible, *i.e.* $p_{ij} = 0$, the final edge cannot be aligned and hence, x_{ij} is forced to be 0. In the case where the edge is possible, *i.e.* $p_{ij} = 1$, x_{ij} can either take the value 0 or 1, according to the other restrictions and optimizations. In a vector formulation, Equation (25) becomes Equation (26).

$$\mathbf{x} \leq \mathbf{p} \quad \Leftrightarrow \quad \mathbf{p} - \mathbf{x} \geq \mathbf{0} \tag{26}$$

The other restriction, second in section 4.2, which states that *each point can only be superposed to one and only one point from each conformation*, was adapted to state that, *each point can only be superposed to* 0 *or* $M - 1$ *points where* M *is the number of different molecules*. This extrapolation was possible because one already assumes that every possible edge needs to be from different molecules. By stating that one point can only be connected to 0 or $M - 1$ points, it is already taken into account that when M molecules are being aligned this results in an alignment between all molecules, and hence, one point from each molecule needs to be connected to $M - 1$ other points.

$$\sum_{j \sim i}^{N} x_{ij} \leq M - 1 \quad \Leftrightarrow \quad M - 1 - \sum_{j \sim i}^{N} x_{ij} \geq 0, \qquad \forall i \in \text{Points} \tag{27}$$

where N is the total number of points.

With this slight modification to the restriction, comes another problem because this restriction do not ensure that a given point is aligned to $M - 1$ points, each from different molecules! As an example, imagine a point i, from molecule A, two other points from molecule B, and a final point from molecule C. One knows that point i needs to be aligned with two points, one from each of the other molecules, because there are three molecules to align. With the above formulation, it allows i to be connected to the two points from the same molecule. In order to solve this problem, another restriction was added, which states that if point A is aligned with point B and C, it implies that point B and point C also needs to be aligned, and hence, respect all the other restrictions. This is assured by Equation (28).

$$\frac{1}{3}(x_{ij} + x_{jk} + x_{ik}) \leq 1 \quad \Leftrightarrow \quad 1 - \frac{1}{3}(x_{ij} + x_{jk} + x_{ik}) \geq 0, \qquad \forall i, j, k \in \text{Points} \tag{28}$$

The next step formulates the problem itself and, therefore, the Hamiltonian. To do so, one needs to first specify what is the objective function. In this problem, the objective function is to maximize the number of points aligned and minimize the total distance between such alignments. This is stated in Equation (29),

$$\min \sum_{i \sim j}^{N} d_{ij} x_{ij} - \sum_{i \sim j}^{N} x_{ij} \tag{29}$$

where, by minimizing the negation of the sum of final aligned edges, one achieves its maximization. The reason to formulate this optimization as a minimization is because Ising formulations are suitable to find the ground state, the lowest level of the function, *i.e.* the minimization. In vector formulation, Equation (29) becomes

$$\min \mathbf{d}^T \mathbf{x} - \mathbf{x}. \tag{30}$$

Combining Equations (25), (27), (28) into Equation (29), one gets the augmented Lagrangian:

$$\begin{aligned} H = \sum_{i \sim j}^{N} d_{ij} x_{ij} - \sum_{i \sim j}^{N} x_{ij} + A \sum_{i \sim j}^{N} \left(p_{ij} - x_{ij} \right)^2 \\ + B \sum_{i}^{N} \left(M - 1 - \sum_{j \sim i}^{N} x_{ij} \right)^2 \\ + C \sum_{i}^{N} \sum_{j}^{N} \sum_{k}^{N} \left(1 - \frac{1}{3} (x_{ij} + x_{jk} + x_{ik}) \right)^2 \end{aligned} \tag{31}$$

Because it is required the absolute value of the expressions inside the parentesis, they were raised to the power of two to inforce non-negative values. Additionally, A, B and C are energy penalties associated to each restriction. According to the relevance/weight one wants to assign to each restriction, the values are altered. The search for the ideal constant for each value is a challenge and must always be tested because hardware noises also introduce an energy penalty to the solution. Reference Lucas (2014) documents how to make a rough estimation for such values based on the maximum and minimum values of the constraint. Since this constants depend on the simulation itself, an approach was to first test the objective function without any restrictions and see its behavior, and only later introduce the restriction and respective energy penalties.

As already stated, due to the number of qubits required, a qSAT for only two molecules was implemented. Having that in mind, the last restriction does not apply and was not implemented. Above that, because in this restriction one needs to go through the list of points several times within the same restriction, which is somehow equivalent to finding a subgraph within a graph, it is required a more complex approach when taking this into a QP formulation. In Lucas (2014) several problems are documented which require the same approach, named cycles, but here, this was not considered.

Now comes the practical implementation.

6.1.3 *Implementation*

Since the quantum device is going to implement Equation (29), one needs first to write it in the **Z** basis, and so Equation (29) becomes Equation (32), using $x_{ij} \to \frac{1-z_{ij}}{2}$.

$$\mathbf{H} = \frac{1}{2}\sum_{i\sim j}^{N} d_{ij} - \frac{1}{2}\sum_{i\sim j}^{N} d_{ij}z_{ij} - \frac{1}{2}\sum_{i\sim j}^{N} 1 + \frac{1}{2}\sum_{i\sim j}^{N} z_{ij} \tag{32}$$

Using vector id representation and isolating the expression that defines the contribution of the individual variables, Equation (32) becomes:

$$\mathbf{H} = \left(\frac{1}{2}(\mathbf{1}-\mathbf{d})\right)^{T}\mathbf{z} + \frac{1}{2}(\mathbf{d}-\mathbf{1}) = \mathbf{g}^{T}\mathbf{z} + \mathbf{c} \tag{33}$$

Differently from VRP, there is no interaction between variables (**Q** expression).

Now that all theory has been discussed, we arrive at another stage: the implementation of the qSAT using Qiskit. Notice that before the Qiskit implementation, some data preparations was performed with respect to the *pre-alignment* of the molecules. It suffices to say that the data is available in two vectors, one regarding the distances, D and another the possibilities P of each edge.

As in VRP, a *QuantumOptimizer* class with the methods `binary_representation`, `construct_hamiltonian`, `check_hamiltonian`, `vqe_solution` and `_q_solution` and of course, the initialization method was created. Although at this point only the D vector is required, this method is already prepared to receive the P vector as well.

To begin with, look at the `binary_representation` method.

```
def binary_representation(self,x_sol=[]):

        p = self.p
        d = self.d
        size = self.size
        ones = np.ones(size)
        g = [x / 2 for x in np.subtract(ones,d)]

        c = np.sum([x / 2 for x in np.subtract(d,ones)])

        try:
            fun = lambda x: np.dot(d,x) - np.dot(ones,x)
            cost = fun(x_sol)
        except:
            cost = 0

        return g, c, cost
```

This implements the QP formulation of the problem already in the **z** basis, following Equation (33). Besides the D and P vectors, it also creates a vector *ones* with similar size. Next the cost of the objective function is evaluated, given a solution as input, in order to check its validity.

In the following method, `construct_hamiltonian`, the Hamiltonian is generated in the form of Pauli terms:

```
def construct_hamiltonian(self):

        p = self.p
        d = self.d
        size = self.size

        N = size

        gz,cz, _ = self.binary_representation()

        pauli_list = []
        for i in range(N):
            if gz[i] != 0:
                wp = np.zeros(N)
                vp = np.zeros(N)
                vp[i] = 1
                pauli_list.append((gz[i], Pauli(vp, wp)))

        pauli_list.append((cz, Pauli(np.zeros(N), np.zeros(N))))
        return cz, pauli_list
```

The next two methods, `check_hamiltonian` and `vqe_solution`, use the full form Hamiltonian resorting to an exact (classic) Eigensolver, in the first case, and a quantum simulator/device, in the second, to compute a solution to the problem. The first method only computes the lowest Eigenvalue and Eigenvector (the offset is set to 0), while in the second all information is used and obtained a solution to the objective function.

`vqe_solution` uses VQE as the chosen Eigensolver, SPSA as the optimizer and RY as the variational form. It is important to notice that these choices were made based on the VRP problem already mentioned.

```
    def check_hamiltonian(self):

        cz, op = self.construct_hamiltonian()
        Op = WeightedPauliOperator(paulis=op)

        qubitOp, offset = Op, 0

        result0 = NumPyEigensolver(operator=qubitOp).run()
        result = result0['eigenstates'].to_matrix(massive=True)

        quantum_solution = self._q_solution(np.real(result[0]).tolist(),self.size)

        ground_level = np.real(result0['eigenvalues'][0]) + offset

        return quantum_solution, ground_level

    def vqe_solution(self):

        cz, op = self.construct_hamiltonian()
        Op = WeightedPauliOperator(paulis=op)

        qubitOp, offset = Op, cz

        aqua_globals.random_seed = 10598

        num_qubits = qubitOp.num_qubits
        var_form = RY(qubitOp.num_qubits, depth=5, entanglement='linear')
        optimizer = SPSA(max_trials=self.max_trials)
```

```
algo = VQE(qubitOp, var_form, optimizer)

backend = provider.get_backend('ibmq_qasm_simulator')

quantum_instance = QuantumInstance(backend,
                                   seed_simulator=aqua_globals.random_seed,
                                   seed_transpiler=aqua_globals.random_seed,
                                   skip_qobj_validation=False)
result = algo.run(quantum_instance)

quantum_solution_dict = result['eigenstate']

q_s = max(quantum_solution_dict.items(), key=operator.itemgetter(1))[0]
quantum_solution= [int(chars) for chars in q_s]
quantum_solution = np.flip(quantum_solution, axis=0)

_,_,level = self.binary_representation(x_sol=quantum_solution)
return quantum_solution_dict, quantum_solution, level
```

Finally, consider the `_q_solution` method, which solely converts the output of the Eigensolver to an usable format, and the initialization method. It can be consulted in appendix B.1, the entire class is definition.

Due to some problems detected along the simulations, no further discussion is made here regarding the QP formulations of the restrictions. Please refer to section 7.2 to read about such problems and respective results, expected and unexpected.

6.2 SOLUTION SEARCH

This section concerns the solution search. A generalization of the Grover's algorithm (Grover, 1996) leads to the requested solution. This generalization takes inspiration in several works but mainly Alves (2019); Boyer et al. (1996); Dürr and Høyer (1996).

The following subsections describe the algorithm and how it was adapted and implemented, respectively.

6.2.1 *Original Grover's algorithm and its generalizations*

$C(S)$	0	0		1		0
S	S_1	S_2	...	S_w	...	S_N

Table 2: Grover's example database

Grover's algorithm is a fast algorithm for database search problems published in 1996 (Grover, 1996). Since then its relevance has been growing due to the possibility of using it as a subroutine that enables a quadratic runtime enhancement, in several algorithms (Neri and Rodrigues, 2018).

Consider a database with N items. Each item has a label as its description and a state for its representation $S_1, S_2, ..., S_N$. Let S_w be the state that satisfies condition $C(S_w) = 1$. To identify S_w, one has to encode an oracle function like (34).

$$\begin{aligned} f(w) &= 1 \\ f(s) &= 0 \end{aligned} \tag{34}$$

In other words, there is an array of N elements, designated S. The output of the oracle function is 1 for the envisaged object and 0 otherwise (see table 2).

Any classical algorithm, deterministic or probabilistic, takes $O(N)$ steps since on average it has to examine a large fraction of the N items: in the worst case all N items but on average $N/2$ items. The Grover's algorithm shows an evident advantage by finding the envisaged object in $\sqrt{N}$ steps. Additionally, this algorithm is also generic because it does not use the internal structure of the list. In Figure 32 is a schematic diagram of the Grover's algorithm.

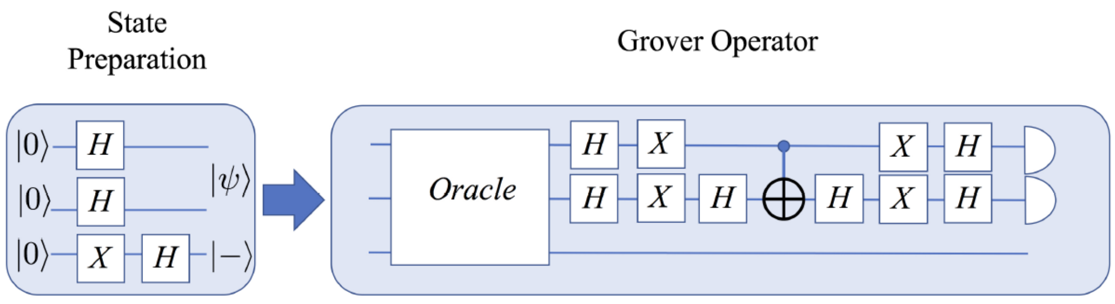


Figure 32: Grover's algorithm. In this case, only one application of the Grover operator is perfomed. The *oracle box* represents the oracle operator U_f and the remaing circuit on the right, except for the external Hadamard gates, the diffusion operator U_D. *Fig.6 in J. et al. (2020).*

The subsequent steps express the algorithm implementation, as depicted in Neri and Rodrigues (2018), IBM (c) and J. et al. (2020), among others.

1. Initialize the system with a uniform superposition of states, *i.e.* with the same amplitude in each N state (check Figure 33). This superposition is obtained through application of Hadamard gates. The system is now the superposition $|s\rangle = |0\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$.

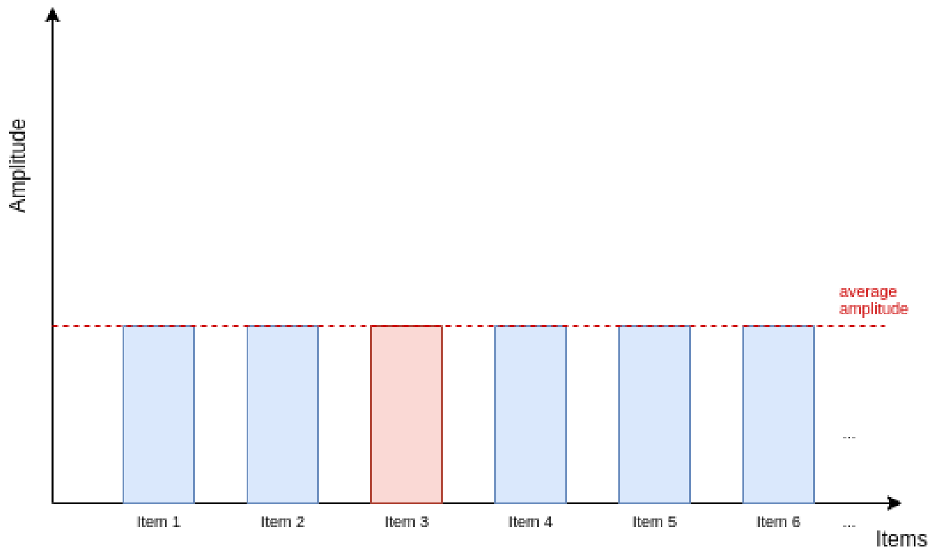


Figure 33: System distribution after first step. The average amplitude is $\frac{1}{\sqrt{N}}$. ©SofiaOliveira, 2020

2. Apply $O(\sqrt{N})$ times the following operators which together form the Grover's operator, as depicted in Figure 32.

   a) Quantum operator U_f. This operator is responsible for identifying the solution, by application of the oracle function (34), as the target.

   $$U_f|x\rangle = (-1)^{f(x)}|x\rangle \tag{35}$$

   This operator is also called *oracle reflection* since, geometrically, this corresponds to a reflection of the target state while the other states remain unchanged, as depicted in Figure 34.

   b) Quantum operator U_D, also called the diffusion operator. This operator implementation can be achieved by $U_D = WRW$, where W is the Walsh-Hadamard transform matrix, and R is a rotation matrix. In another notation, this operator can also be written as $U_D = 2|s\rangle\langle s| - 1$. Similarly to U_f, U_D is also considered a reflection operator but not only it flips the desired input but also increases its amplitude and lowers the remaining ones. (see Figure 35).

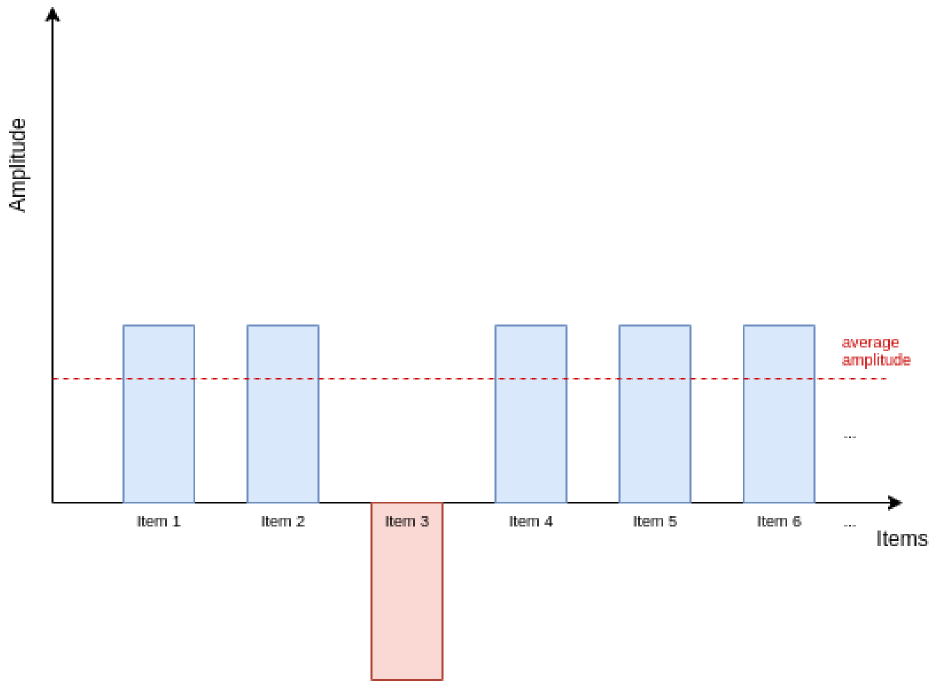


Figure 34: System Distribution after the first application of operator $U_f|\psi_t\rangle = \psi_{t'}$. Because the amplitude of the desired state becomes negative, the average amplitude is lowered and hence, becomes $< \frac{1}{\sqrt{N}}$. ©SofiaOliveira, 2020

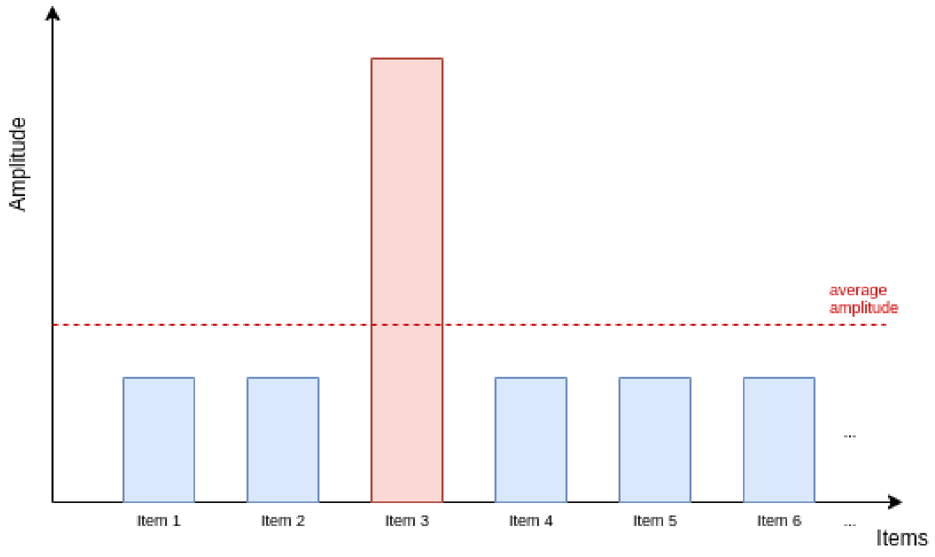


Figure 35: System Distribution after the diffusion operator $U_D|\psi_{t'}\rangle$. The action of this operator can be seen as a reflection about the average amplitude. Since the average amplitude has been lowered, this transformation boosts the negative amplitude of the desired state while it decreases the other amplitudes. ©SofiaOliveira, 2020

Both steps have to be repeated roughly $\sqrt{N}$ times to get close to the optimal measure ($|w\rangle$).

3. Finally the qubits are measured. After t rounds the state transformation is:

$$\psi_t = (U_D U_f)^t |\psi_0\rangle \tag{36}$$

At this point, the probability of finding the wanted object is at least $\frac{1}{2}$.

Despite the enormous popularity of the original article, the scientific community raise some relevant questions: *What if there is more than one solution, i.e. the desired element appears more than once in the table?*, and hence *What if the number of solutions is not known?*.

First in Boyer et al. (1996) and latter in the corresponding published version (Boyer et al., 1998), these issues were investigated and some attempts to solve them were presented. These papers generalize the Grover algorithm by approximately calculating the number of solutions with an approach inspired by Shor's quantum factorization algorithm (Shor, 1994)

Additionally, they also give a simple closed-form formula for the probability of success after any given number of iterations. This allows to determine the number of iterations necessary to achieve almost certainty of finding the answer, as well as an upper bound on the probability of failure.

According to both Grover (1996) and Boyer et al. (1996), if there is a unique solution that suffices the problem, the algorithm must take about $\sqrt{N}$ iterations.

It is essential to observe that in this project, the envisaged element is the smallest number, or even the localization of such solution, in the list. Some months after the release of the draft mentioned above, Dürr and Høyer (1996) presented a quantum algorithm for finding the minimum, in which the main subroutine is the Shor and Grover's inspired exponential searching algorithm of Boyer et al. (1996).

In order to better understand such algorithms, consider the simple case of a list of N unsorted and distinct values. If one consider $T[0..N-1]$ to be the list, the problem is to find the index y such that $T[y]$ is the minimum. The suggested algorithm solves the problem after $O(\sqrt{N})$ iterations of the circuit.

Summarizing, the algorithm calls the previously mentioned algorithm as a subroutine in order to find the index of an element smaller than the value determined by a particular *threshold index*. The result is then chosen as the new threshold. This process is repeated until the probability that the threshold index selects the minimum is sufficiently large. If there are $t > 1$ marked table entries, the exponential searching algorithm will return one of them with equal probability after an expected number of $O(\sqrt{N/t})$ iterations.

This is done by the following steps.

log stands for the binary logarithm. It is also important to notice that there is no "time-out", *i.e.* it is expected that the algorithm always runs long enough to find the minimum. This makes the algorithm an *infinite algorithm*. Due to the lack of a "time-out", there is also

```
1. Choose the threshold index $0 \leq y \leq N - 1$ uniformly at random.
2. Repeat the following actions and interrupt it when the total running time is more
   than $22.5\sqrt{N} + 1.4\log^2 N$. Then go to stage 2(2c).
   a) Initialize the memory as $\sum_j \frac{1}{\sqrt{N}}|j\rangle|y\rangle$. Mark every item $j$ for which $T[j] < T[y]$.
   b) Apply the quantum exponential searching algorithm.
   c) Observe the first register: let $y'$ be the outcome. If $T[y'] < T[y]$, then set
      threshold index $y$ to $y'$.
3. Return $y$.
```

Algorithm 1: Quantum Exponential Searching Algorithm

the possibility that the algorithm could never find the solution, even with infinite time, if no precautions are made.

According to Dürr and Høyer (1996), the expected total time used by the infinite algorithm before y holds the index of the minimum is at most

$$m_0 = \frac{45}{4}\sqrt{N} + \frac{7}{10}\log^2 N \tag{37}$$

After at most $2m_0$ iterations, $T[y]$ holds the minimum value with probability at least $\frac{1}{2}$. The proof can be found in the original paper.

Now that the algorithm is clear, let us discuss the implementation in Qiskit.

6.2.2 *Implementation*

When comes to the implementation, there are some data treatments that need to be done, such as the calculation of the number of qubits required to the circuit and the binary conversion of the data.

This circuit requires two sets of qubits, the ones that represent the index of the list and the ones that represent the values. To calculate the qubits required for the index, it is enough to calculate the length of the list, subtract one (because indexes start at 0) and convert that final number, N, to binary. The number of qubits is the length of this binary number, named `qy`. The number of qubits to represent the value, named `maxq`, is, ideally, the bit length of the maximum value on the list, which requires such value to be known previously. This is, by itself, biased since once is calculating the maximum of a list in order to find its minimum. Another way to find such a value is through some knowledge of the list. If one knows that the values cannot, at any point, surpass a certain limit, that limit can become the reference

decimal		Primitives	binary						
index	value		index			value			
			i_0	i_1	i_2	v_0	v_1	v_2	v_3
0	6	P0	0	0	0	0	1	1	0
1	1	P1	0	0	1	0	0	0	1
2	9	P2	0	1	0	1	0	0	1
3	12	P3	0	1	1	1	1	0	0
4	0	P4	1	0	0	0	0	0	0
5	10	P5	1	0	1	1	0	1	0
6	4	P6	1	1	0	0	1	0	0

Table 3: Example, relations between indexes and values.

number for the qubit calculation. Given that the number of qubits available in nowadays quantum devices is rather small, it was used the exact maximum value of the list as the reference maximum. As mentioned, this was necessary to obtain practical results, although theoretically not the best solution.

Then it is necessary to establish the correspondence between the index, values and their binary representation. To do that, consider the list $[6, 1, 9, 12, 0, 10, 4]$ and look at Table 3. It is required 3 qubits for the index ($2^3 = 8$ indexes) and 4 qubits for the value ($2^4 = 16$, maximum number is 12).

The connection between values and primitives is

$$\begin{aligned} v_0 &= P_2 + P_3 + P_5 \\ v_1 &= P_0 + P_3 + P_6 \\ v_2 &= P_0 + P_5 \\ v_3 &= P_1 + P_2 \end{aligned}$$

and is stored in `data_t`, as such, `{'v0': ['010', '011', '101'], 'v1': ['000', '011', '110'], 'v2': ['000', '101'], 'v3': ['001', '010']}`.

The association between primitives and indexes is established by function `symp_aux`, where the inputs are the number of qubits in the index, `qy`, and the list of data `data_t`. In detail, `symp_aux`:

1. verifies if the number of qubits is ok.
2. by usage of `func_aux`, run through the list of primitives.
3. jumps to `getand` (where the output, taking as example primitive P_5, is "`i0 & i2 & ~i1`") and adds all primitives with the logic 'or', `|`.

4. and resorting to function `simplify_logic` from the library sympy, takes care of the ensuing simplification.

The code corresponding to the functions can be found in appendix B.2.

Let us turn now to the implementation of the generalized Grover algorithm itself, which will generate the circuit. `qaux` is the number of auxiliary qubits necessary to construct the circuit.

```
def grover(qy, maxq, qaux, data_simp,v_threshold,J):
    qr, cr = create_register(qy,maxq,qaux)
    qc_grover= QuantumCircuit(qr,cr)
    qc_grover_0 = grover_init(qr, cr, qy)
    qc_grover_IV=grover_indexValues(qr,cr,qy,maxq,data_simp)
    qc_grover_O1=grover_minor(qr, cr, qy, maxq, v_threshold)
    qc_grover_O2=grover_mark_j(qr,cr,qy,maxq)
    qc_grover_D1=grover_D(qr,cr,qy,maxq)
    qc_grover_M = grover_measure(qr, cr, qy)
    qc_grover = qc_grover_0 + qc_grover_IV
    for i in range(J):
        qc_grover = qc_grover + qc_grover_O1 + qc_grover_O2 + qc_grover_D1
    qc_grover = qc_grover + qc_grover_M
    return qc_grover

y = randrange(N-1)
v_threshold = data[y]

m0= int((9/2)*(m.sqrt(N)))

#'Grover'
J = methods(N,M,'Grover')

backend = Aer.get_backend("qasm_simulator")
shots = 1

for i in range(m0):
    grover_circuit=grover(qy, maxq, qaux, data_simp, v_threshold,J)
    job=execute(grover_circuit, backend, shots=shots)

    result_S = job.result()
    counts_sim = result_S.get_counts(grover_circuit)
    y_temp=int(list(counts_sim.keys())[0], 2)

    if y_temp < len(data):
        if data[y_temp] <= v_threshold:
            v_threshold = data[y_temp]
            y = y_temp
        print(y,v_threshold)

print('RESULT:', '\n          index',y,'\n          value', v_threshold)
```

Taking into account Algorithm 1, step (2b) is used to set the number of required iterations, J, of the Grover algorithm, as shown in the previous code.

By running the code above one gets the circuit but before, one also needs to initialize the qubit registers and other peculiarities. To perform the registers initialization one uses the function `create_register` which, resorting to the Qiskit library initializes both the quantum registers and the classic registers as seen next.

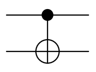
```
def create_register(qubits_index,qubits_values,qubits_aux):
    qrn= qubits_index + qubits_values + qubits_aux
    qr = QuantumRegister(qrn, 'qr')
    cr = ClassicalRegister(qubits_index,'cr')
    return qr, cr
```

The initialization of the memory, step (2a) in Algorithm 1, is accomplished by adding an Hadamard gate to each qubit of the index.

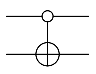
```
def grover_init(qubits, bits, n):
    qc_grover_I = QuantumCircuit(qubits,bits)
    for h in range(n):
        qc_grover_I.h(qubits[h])
    return qc_grover_I
```

Recalling that indexes and values are associated trough logic relations of primitives, it is important to know how to represent them in a circuit. Therefore, there are four associations that can occur and are implemented in the circuit as follows:

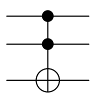
1. The value has a direct relation to the primitive:



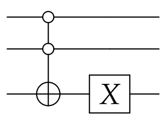
2. The value has a direct relation to the negation of the primitive:



3. The value is the result of a logic AND between two primitives:



4. The value is the result of a logic OR between two primitives:



Now one only needs to write the circuit as above, as illustrated next. It now comes to translating the relations mentioned in the beginning of this implementation to a quantum circuit. As mentioned above, there is a possibility that the primitive is an expression. If the value is only one primitive and such primitive is an index or the negation of an index, it is only required a `cx` gate. On the other hand, when such primitives are expressions of indexes, one needs to divide the implementation into two steps, `truth_table` and `function_aux`.

```
def grover_indexValues(qubits, bits, n, m, data):
    # initialize this section of the circuit
    qc_grover_IV = QuantumCircuit(qubits,bits)
    # get the list of values - keys
    keys= list(data.keys())
    #qt is the position of qubit value
    qt=n
    # run the list of values
    for k in keys:
        simple=0
        value=data[k]
        # if it is a symbol:
        if type(value)!=And and type(value)!=Or:
            gate_cx(value,qt,qc_grover_IV,qubits)
        #if it is an expression (And / Or)
        else:
            table = truth_table(data_simp,k)
            function_aux(table, qt, n+m, qc_grover_IV,qubits)
        qt=qt+1
    return(qc_grover_IV)
```

(Please refer to Appendix B.2 to see the definition of `gate_cx`.)

```
def explore(expr):
    variables = expr.free_symbols
    list_values = []
    for truth_values in cartes([False, True], repeat=len(variables)):
        values = dict(zip(variables, truth_values))
        values['target']=expr.subs(values)
        list_values.append(values)
    return(list_values)

def truth_table(data,v):
    expr = sympify(data[v])
    data_truth=explore(expr)
    return(data_truth)
```

The `truth_table` receives the data expressions that correlates the indexes and values with resource of the primitives and translates it to a truth table. With the help of the function `explore`, it runs over the list of the possible outcomes and returns a list of all possible (classical) inputs and the respective target output. The result is stored in `data_truth`.

If there is the need to use the function `function_aux`, it means the primitive is an expression of indexes. Accordingly, the expression may have two controls or be the result of other expressions. If the controls are symbols, a simple toffoli gate works, but recall that even symbols can be a *not statement*. (Please refer to Appendix B.2 to see the definition of `gate_toffoli`.)

```
def function_aux(t, qt, qa, qc,qr):
    for x in t:
        control_number = len(x)-1
        if x['target']:
            if control_number==2:
                gate_toffoli(x, qt, qc,qr)
            else:
                decomposed_gate(control_number, qt, qa,qc,x,qr)
```

In the possibility of the expression being composed by other expressions, since there is a truth table, it is possible to focus in the situation where the target changes. In that situation, a C^nNOT is required, but since this gate doesn't exist in Qiskit, a decomposition with various toffoli gates is used (Bib, 2020).

```
def decomposed_gate(qcontrol,qtarget,qaux,qc,data,qr):
    # di is the situations where the target is true
    di = list(data.items())
    new_control = toffoli_init(di[0], di[1], qaux, qc,qr)
    for x in di[2:-1]:
        qaux=qaux+1
        new_control = toffoli_mid(new_control, x, qaux, qc,qr)
    qc.cx(qr[new_control],qr[qtarget])
    for x in reversed(di[2:-1]):
        qaux=qaux-1
        new_control = toffoli_mid_rev(qaux, x, new_control, qc,qr)-1
    new_control = toffoli_init(di[0], di[1], qaux, qc,qr)
```

(Please refer to Appendix B.2 to see the definition of `toffoli_init`, `toffoli_mid` and `toffoli_mid_rev`.)

From the decomposition it is possible to guess the number of ancillary qubits, *i.e.* auxiliary qubits, needed in the circuit.

At this point, the only missing piece is the oracle. To do that one needs to find values smaller than the threshold value. But first, in order to simplify, ignore 0 every time, in any value, it is the most significant bit. As one is dealing only with positive natural numbers, if the value is $T[i] = 0$, there is no j for which $T[j] < T[i]$. Similarly, every time there is a 0, there is no need to add any gate to the circuit.

```
def grover_minor(qubits, bits, n, m, v_threshold):
    qc_grover_m = QuantumCircuit(qubits,bits)
    #v is v_threshold in binary
    v= "{0:b}".format(v_threshold).zfill(maxq)
    target = m+n
    zero=False
    for x in "{0:b}".format(v_threshold):
        if x == '0':
            zero = True
    if zero==False:
        mark_or_zero(v, n, m, qc_grover_m,qubits)
    else:
        toconsider = False
        count=n
        lcount= []
        for i in v:
            if i=='1':
                lcount.append(count)
                if not(toconsider):
                    mark_zero(count, target, qc_grover_m,qubits )
                else:
                    mark_other(lcount, target, qc_grover_m,qubits)
                toconsider = True
            count=count+1
    return(qc_grover_m)
```

Auxiliary functions, such as mark_or_zero, mark_zero and mark_other, are used to decompose several cases of threshold values in order to obtain the list of possible values inferior to the threshold. This list is not a list per se, but instead, a circuit that identifies the values that are inferior to the threshold. (Please refer to Appendix B.2 to see the definition of these auxiliary functions.)

After knowing how such inferior values are found, it is time to implement how to mark the indexes of such values. First, the information is entangled in the auxiliary bits with the index qubits. Then the information is marked with a *CZ* gate. If there are only 2 qubits holding index information then only one gate is required, otherwise, the gate needs to be decomposed.

```
def grover_mark_j(qubits, bits, n, m):
    qc = QuantumCircuit(qubits,bits)
    target = m+n
    final_t = n
    for x in range(n):
        qc.cx(qubits[target], qubits[x])
    #only 2 qubits in for index
    if n<= 2:
        qc.cz(qubits[0], qubits[1])
    #need for decomposition
    else:
        t=n+m
        qc.ccx(qubits[0],qubits[1],qubits[t])
        ic_1=t
        for x in range(2,n-1):
            ic_2=x
            t=t+1
            qc.ccx(qubits[ic_1],qubits[ic_2],qubits[t])
            ic_1=t
        qc.cz(qubits[ic_1],qubits[final_t])

    #reverse
        for x in range(n,3,-1):
            ic_1=ic_1-1
            qc.ccx(qubits[ic_1],qubits[ic_2],qubits[t])
            ic_1=t
            t=t-1
        t = t-1
        qc.ccx(qubits[0],qubits[1],qubits[t])
    for x in range(n):
        qc.cx(qubits[target], qubits[x])
    return(qc)
```

The final step is the application of the diffusion operator, U_D, and the measurement.

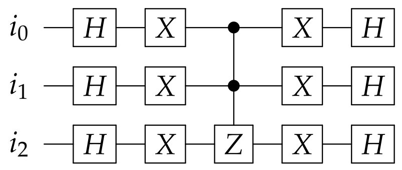


Figure 36: U_D with $qy = 3$.

This operator has Hadamard and X gates in all the index qubits. After, there is a Z or C^nZ gate given by the function ncz. And finally, the diffusion ends with X and Hadamard gates in the index qubits. (See Figure 36)

```
def grover_D(qubits, bits, n, m):
    qc_grover_D = QuantumCircuit(qubits,bits)
    for i in range(n):
        qc_grover_D.h(qubits[i])
    for i in range(n):
        qc_grover_D.x(qubits[i])
    ncz(n, m, qc_grover_D,qubits)
    for i in range(n):
        qc_grover_D.x(qubits[i])
    for i in range(n):
        qc_grover_D.h(qubits[i])
    return(qc_grover_D)

def ncz(n,m,qc_grover_D,qubits):
    t=n-1
    # only one qubit
    if n == 1:
        qc_grover_D.z(qubits[t])
    elif n ==2 :
    # only two qubits
        qc_grover_D.cz(qubits[0],qubits[t])
    # more than 2 qubits
    else:
        decomposed_gate_Z(n,m,qc_grover_D,qubits)
```

As before, if there is more than 2 qubits, a decomposition is required.

```
def decomposed_gate_Z(index,m, qc_grover_D,qubits):
    target= index+m
    qc_grover_D.ccx(qubits[0],qubits[1],qubits[target])
    i=index
    for i in range(2,index-1):
        t=target+1
        qc_grover_D.ccx(qubits[target],qubits[i],qubits[t])
        target=t
    qc_grover_D.cz(qubits[target],qubits[i-1])
    for i in range(index-1,2,-1):
        target=target-1
        t=target+1
        qc_grover_D.ccx(qubits[target],qubits[i-1],qubits[t])
        t=t-1
        target=t
    qc_grover_D.ccx(qubits[0],qubits[1],qubits[target])
```

By last, the measure, only required for the index qubits is given by the function grover_measure.

```
def grover_measure(qubits, bits, n):
    grover_m= QuantumCircuit(qubits,bits)
    for h in range(n):
        grover_m.measure(qubits[h],bits[h])
    return grover_m
```

To run the circuit using specific data, one uses the function `findsolution` which receives the case study data as input. Such function can be consulted in appendix B.2. In the next section, all results and tests executed will be discussed.

7

RESULTS AND COMPARISONS

7.1 PRE-ALIGNMENT

Although not relevant in a comparison, it still important to show the results of this step, even if it was completely classical. As previously explained in section 5, it resorted to an algorithm called CPD with slight modifications. Next, in table 4 and 5, are displayed some results of this pre-alignment regarding two and three conformations, respectively, with 4 decimal values. It is quite easy to deduce which molecule has been used as a reference in each case since the coordinates are the same before and after the transformation.

	Old Coordinates			New Coordinates		
Molecules	x	y	z	x	y	z
1TOM	21.2667	−14.8849	24.9931	21.4404	−15.1142	24.5667
	19.5525	−13.9286	20.8894	19.3836	−14.6355	20.5375
	12.7832	−12.2659	21.5865	12.9837	−11.9752	21.5558
	10.4754	−12.7687	23.0764	10.8223	−11.9027	23.3227
	16.783	−16.8314	23.3519	16.5988	−16.6806	23.7348
1D4P	21.1589	−15.1013	24.7647	21.1589	−15.1013	24.7647
	18.7243	−14.1965	20.3818	18.7243	−14.1965	20.3818
	12.9148	−11.7571	21.6896	12.9148	−11.7571	21.6896
	14.4792	−12.7914	20.5876	14.4792	−12.7914	20.5876
	10.3365	−11.7564	24.1149	10.3365	−11.7564	24.1149

Table 4: Coordinates of a two-molecule set before and after pre-alignment. First conformation from each of the molecules.

Since there are about 144 and 1728 different combinations regarding two and three conformations at a time, respectively, only one from each will be displayed in this document. The entire data can be consulted in this link[1].

1 https://github.com/msofiasoliveira/MasterDissertation/blob/master/pre-alignment.ods

	Old Coordinates			New Coordinates		
Molecules	x	y	z	x	y	z
1C4V	20.843	−15.2005	25.3773	21.3172	−15.1882	25.0506
	18.8752	−13.6715	20.5892	19.1857	−14.3468	20.1633
	12.7282	−11.6792	21.741	13.1105	−12.1026	21.2294
	17.7388	−18.4445	27.0457	18.2418	−18.0983	27.2904
	17.739	−16.7215	24.222	18.1540	−16.8096	24.2451
	20.554	−12.778	18.604	20.7977	−13.7837	18.0087
	10.41	−11.8565	24.2975	10.8897	−11.8625	23.8660
1TOM	21.2667	−14.8849	24.9931	21.4404	−15.1142	24.5667
	19.5525	−13.9286	20.8894	19.3836	−14.6355	20.5375
	12.7832	−12.2659	21.5865	12.9837	−11.9752	21.5558
	10.4754	−12.7687	23.0764	10.8223	−11.9027	23.3227
	16.783	−16.8314	23.3519	16.5989	−16.6806	23.7348
1D4P	21.1589	−15.1013	24.7647	21.1589	−15.1013	24.7647
	18.7243	−14.1965	20.3818	18.7243	−14.1965	20.3818
	12.9148	−11.7571	21.6896	12.9148	−11.7571	21.6896
	14.4792	−12.7914	20.5876	14.4792	−12.7914	20.5876
	10.3365	−11.7564	24.1149	10.3365	−11.7564	24.1149

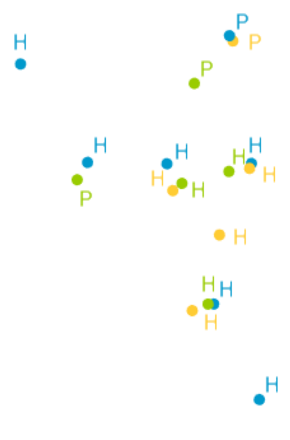
Table 5: Coordinates of a three-molecule set before and after pre-alignment. First conformation from each of the molecules.

Figures 37 and 38 display the data of Table 5. Figure 37 depicts the coordinates alignment of the three molecules while Figure 38 illustrates the coordinates of each individual molecule, before and after the transformation. Please notice that in this particular case the alignment was made with reference to the molecule 1D4P. According to the molecules to be aligned, the reference may change and hence, the alignment can vary.

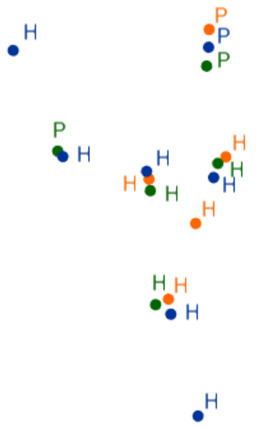
Just so one can better understand how much of a difference this pre-alignment makes: check below, in matrices (38) and (39), the differences obtained when running the classical SAT solver with both coordinates, before (left) and after (right) the transformation. The following examples deal with the first conformation of each of the two and three molecules, respectively. The results below were obtained with the same data displayed above. Due to their irrelevance, all zero internal matrices were removed for better display.

7.2 SAT SOLVER

For the SAT solver, one has to consider data and algorithms regarding both the classic and quantum approaches. In order to make this data presentation more intuitive, we continue to consider the same example as above. As mentioned, matrix (38) and (39) display the

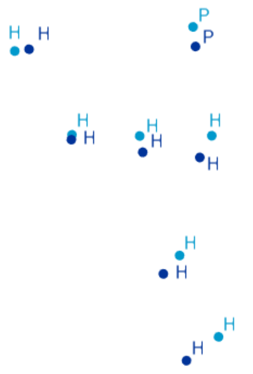


(a)

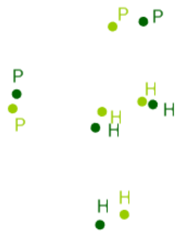


(b)

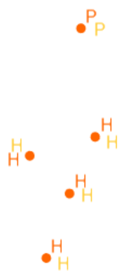
Figure 37: Alignment of three molecules (a) before, in light colors, and (b) after, in darker colors, pre-alignment. *Illustration made with GeoGebra.* ©SofiaOliveira, 2020



(a) 1C4V



(b) 1TOM



(c) 1D4P

Figure 38: Coordinates of each molecule before (in light colors) and after (in darker colors) pre-alignment. *Illustration made with GeoGebra.* ©SofiaOliveira, 2020

results of the classical SAT solver implementation with coordinates before (left) and after (right) pre-alignment.

$$\begin{bmatrix} 0.3326 & 0 & 0 & 0 & 0 \\ 0 & 1.0076 & 0 & 0 & 0 \\ 0 & 0 & 0.5356 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.4569 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0.3444 & 0 & 0 & 0 & 0 \\ 0 & 0.8073 & 0 & 0 & 0 \\ 0 & 0 & 0.2649 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.9407 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (38)$$

At this point, because the data is already expected to be pre-aligned, only the matrices in the right are relevant. Here, is possible to see that every restrictions is obeyed. Do not forget that each position of the matrix is the connection between every two and three points, in matrix (38) and (39), respectively. Each dimension of the matrix represents a conformation from a different molecule.

As previously, there are a lot of data to be documented but because the work is explanatory with only one example, the remaining data (all SAT implementation matrices of every possible solution) can be checked in this link[2].

$$\begin{bmatrix} \begin{bmatrix} 1.6822 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 2.3761 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.3534 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \ldots \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3.2033 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \begin{bmatrix} 1.1873 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1.8519 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.2455 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \ldots \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2.1055 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{bmatrix} \tag{39}$$

The qSAT requires 25 qubits (the number of the edges) and it was run on both a quantum simulator (*ibmq_qasm_simulator*) and a real device (*ibmq_cambridge*).

Unfortunately, even with all the abstractions made and using only the objective function and two molecules, neither the quantum device nor simulator were able to give an answer within viable time in the first version of this code[3]. More than 72 hours after, no progress

2 `https://github.com/msofiasoliveira/MasterDissertation/tree/master/Classic_SAT_%20solver_data`

3 Same code but using version 0.18.x of Qiskit.

was obtained and so, no results were retrieved. Nevertheless, the classic simulator of the *quantum code* worked and no errors were raised, indicating that there were no problems in what concerns the implementation and hence, the Hamiltonian.

Unforeseenly, within one month to finish this dissertation, there was an update on Qiskit that produced two unexpected events. The first was the deprecation of the *ExactEigensolver* class, the classic simulator used, which was substituted by the class *NumPyEigensolver*. This generated some problems regarding memory allocation[4] and no results were obtained with the method `check_hamiltonian`. The second event entailed the quick computation (less than one hour and an half) of the qSAT using real data on the quantum simulator. In the real quantum device it took around 40 hours. A long simulation but that got results. It is important to state that a big percentage of the execution time were spent in waiting queues. The simulation itself took only about 3 hours. Nevertheless, the number 40 will be taken as the reference time since it was the total time required for collecting the results.

Initially, because no errors were detected during the waiting period of the first trial code, it was believed that all steps taken and reproduced until this point of the simulation were correct. Nevertheless, the example tested had only two molecules and unfortunately, due to the late update, not all data was tested. Additionally, for any three molecule example, requiring a lot more time, data and qubits, it remained impossible to obtain results.

If one wishes to test the program with other data note that the altered version of CPD is required. In any case, some data regarding two molecules was obtained. First focus on matrix (41), which is the list D, given as input to the qSAT, that contains all distances, times 10, associated with every point connection between conformations. The number 10 is a weight associated with the distance, turning expression (29) to become,

$$\min \quad A \sum_{i\sim j}^{N} d_{ij} x_{ij} - \sum_{i\sim j}^{N} x_{ij} \tag{40}$$

with $A = 10$. Initially, it was thought that no weight was necessary in the objective function since the distance varied within the edges. The problem was that, with no weight, the function always tended to set the solution to 0 since it was never *beneficial* to set any edge to 1.

4 Please see documentation of class *NumPyEigensolver* in https://qiskit.org/documentation/stubs/qiskit.aqua.algorithms.NumPyEigensolver.html.

$$\begin{bmatrix} 3.4438 & 50.7271 & 96.0379 & 83.4782 & 116.0924 \\ 46.0841 & 8.0730 & 71.7346 & 52.3994 & 101.4583 \\ 93.2221 & 62.6636 & 2.6493 & 19.5967 & 36.8839 \\ 109.1584 & 87.3796 & 26.5832 & 46.5225 & 9.4074 \\ 49.3446 & 46.8307 & 64.8046 & 54.3361 & 79.7557 \end{bmatrix} \tag{41}$$

Is interesting to notice that the distances in the final solution in the right of matrix (38) are the same, divided by 10, as the input.

Although it may seem a weird data input compared to the previous data since the classic SAT receives the data as conformations with the corresponding points, the reason behind such input was the classical data preparation. That is, the preparation of the data would be made, in any case, classical, and thus previous to the qSAT. But, one may ask, why using the distance of each *edge*? This time, the answer comes from the problem since, as illustrated in Equation (40), the objective function is a relation between the number of points aligned and their distance. The output of the qSAT used in the simulator is the presented in matrix (42). The results form the real quantum device are displayed in matrix (43).

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{42}$$

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{43}$$

The equivalent solution of matrix (38) in the qSAT is matrix (42) or (43) multiplied by the transverse of matrix (41), with each element divided by 10, that is, matrices (44) and (45).

$$\begin{bmatrix} 0 & 5.0727 & 0 & 0 & 0 \\ 4.6084 & 0.8073 & 0 & 0 & 0 \\ 0 & 0 & 0.2649 & 1.9597 & 0 \\ 0 & 0 & 2.6583 & 0 & 0 \\ 4.9345 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{44}$$

$$\begin{bmatrix} 0.3444 & 0 & 9.6038 & 8.3478 & 11.6092 \\ 0 & 0.8073 & 0 & 0 & 10.1458 \\ 9.3222 & 0 & 0.2649 & 1.9597 & 0 \\ 0 & 8.7380 & 2.6583 & 4.6522 & 0.9407 \\ 0 & 4.6831 & 0 & 0 & 7.9756 \end{bmatrix} \tag{45}$$

As one can observe, this final matrices did not selected the overall edges with the lowest values. Some possible explanations are the lack of time to reach a better solution (variable `max_trials`), the *quantum noise*, which was very disturbing, the weight given to D was not enough, or too much... There are a lot of possible reasons to justify such a result but in any case, the only way one could verify if the behavior of the qSAT was correct would be by implementation of the restrictions. Of course, this solution has no real value since it does not implement the restrictions, but nevertheless proves the functionality of the algorithm. The fact that this result was achieved at a later stage, no restriction was translated into the QP formulation and, unfortunately, no real data relevance was obtained. Nevertheless, such result opens the door for further investigation and implementations.

Comparing both methods, classic and quantum, it is clear that no advantage was obtained with the quantum approach. Time relating comparisons come up to, at most, 5 minutes in the classic solver, and, without the restrictions being implemented, not much less than 1 hour and an half in the quantum simulator and around 40 hours in the real device. Of course, it is important to notice that there are few qubits available in any quantum device but billions of bits is any commercially available classical computer. In the future, when such a difference becomes smaller, maybe the advantage will be proved to be in the quantum algorithm.

7.3 SOLUTION SEARCH

In this step, as in the previous, there are classic and quantum approaches and results. Recalling from section 5.3, Equation (46) states the score each solution has. By *solution*, we mean here a structure of the same sort of matrices (38) or (44). To each such solution a score value is associated, as already explained in more detail in section 5.3. In the case of all edges being zero, the score is set to a maximum of 2000.

$$\text{score}(n_{aligned}, d) = round\left(\frac{200 \times d}{\binom{M}{2} \times n_{aligned}} + 400 \times (5 - n_{aligned})\right) \quad (46)$$

At this point it is expected a list of scores for each solution. The data obtained in the classical SAT solver of two molecules alignment was used as the input list:

1. for molecule $1C4V$ and $1TOM$:

```
[518, 567, 2000, 555, 2000, 527, 616, 2000, 532, 547, 564, 516, 965, 999,
2000, 993, 2000, 925, 967, 2000, 921, 951, 916, 965, 1068, 2000, 2000, 1026,
2000, 1062, 957, 2000, 640, 1050, 944, 994, 599, 597, 2000, 591, 1096, 585,
632, 1064, 574, 562, 604, 594, 1037, 1091, 2000, 2000, 2000, 1002, 949, 2000,
1001, 1039, 1088, 1060, 1080, 1027, 2000, 2000, 2000, 1080, 934, 2000, 1030,
967, 928, 1092, 934, 936, 1032, 920, 2000, 951, 1049, 1038, 937, 927, 940,
926, 1017, 978, 2000, 2000, 2000, 1032, 2000, 1063, 915, 939, 935, 2000, 958,
993, 576, 2000, 588, 928, 672, 594, 2000, 2000, 1105, 986, 979, 999, 2000,
993, 2000, 951, 993, 2000, 1026, 953, 920, 979, 1168, 2000, 2000, 2000, 2000,
1066, 999, 2000, 1058, 1075, 1045, 1108, 621, 601, 2000, 598, 1097, 632, 1074,
1115, 654, 630, 620, 607]
```

2. for molecule $1C4V$ and $1D4P$:

```
[505, 509, 698, 502, 509, 986, 698, 710, 711, 595, 567, 517, 666, 683, 2000,
2000, 682, 685, 2000, 2000, 2000, 2000, 1160, 1123, 667, 2000, 2000, 2000,
2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 524, 524, 2000, 530, 524,
531, 2000, 2000, 2000, 592, 609, 544, 1063, 1069, 1098, 1056, 2000, 2000,
1098, 1011, 2000, 1076, 2000, 1072, 1103, 1111, 2000, 1102, 1110, 1103, 2000,
2000, 2000, 2000, 1129, 1102, 690, 687, 1077, 1070, 687, 1070, 1077, 1075,
1084, 1031, 1134, 703, 1071, 1051, 1068, 1070, 1051, 1070, 1068, 662, 662,
1051, 2000, 1083, 1056, 1069, 685, 1063, 1069, 954, 686, 2000, 2000, 2000,
582, 2000, 2000, 1065, 2000, 2000, 1065, 2000, 2000, 2000, 2000, 2000, 2000,
2000, 2000, 1084, 2000, 1085, 1084, 1085, 2000, 2000, 2000, 2000, 2000, 2000,
585, 550, 2000, 561, 550, 561, 2000, 1009, 2000, 561, 619, 587]
```

3. for molecule $1TOM$ and $1D4P$:

```
[518, 537, 1035, 527, 537, 649, 1035, 690, 690, 2000, 942, 513, 575, 547,
977, 555, 547, 554, 976, 662, 662, 524, 964, 567, 2000, 2000, 581, 2000, 2000,
2000, 582, 930, 610, 2000, 960, 992, 565, 536, 986, 544, 537, 544, 986, 668,
668, 664, 571, 559, 2000, 2000, 679, 2000, 2000, 2000, 679, 991, 991, 2000,
634, 1070, 522, 547, 1033, 531, 547, 531, 1033, 692, 693, 2000, 619, 516,
564, 600, 2000, 583, 600, 583, 2000, 681, 682, 640, 520, 551, 2000, 2000,
596, 2000, 2000, 2000, 596, 615, 614, 2000, 973, 2000, 633, 665, 2000, 656,
657, 656, 2000, 1119, 1119, 629, 650, 633, 992, 669, 1055, 664, 669, 664,
1055, 663, 663, 653, 993, 533, 543, 533, 654, 531, 534, 530, 654, 627, 627,
645, 538, 535, 526, 529, 1003, 520, 530, 520, 1003, 713, 714, 687, 936, 520]
```

In a classical approach, such best score value is easily obtained with the `min()` function of python, but we resorted to the following cycle:

```
for idS, s in enumerate(scores):
        if( s < bestScore):
            bestScore = s
            solution = idS
```

In the end of the cycle the best score is stored in `bestScore` and the corresponding position in `solution`. In the first case, the minimum value is 516, in position 11. In the second case, 502 in position 3. And by last, 513 in position 11. (Position values start at 0.)

When comes to the quantum solution search, it was used a quantum simulator (*qasm_simulator*) instead of a real device since it requires 30 qubits: 8 qubits for solution representation, 11 qubits for value representation and another 11 auxiliary qubits for circuit construction. The quantum devices available at this point count to a maximum of 28 qubits. There is a quantum device, *ibmq_rochester*, that provides 53 qubits but it works by reservation and only allows for simulations in a total of 12 hours per month. Since the algorithm takes more than 2 days per simulation, it was impossible to run it in this backend (more information on time issues later in this section).

In total, six simulations were made, two for each set of data. For the first set of data it was used the same hardware (A) in both, while for the other two sets was used different hardware (A and B) on each simulation. (Please check appendix C for more information.) Tables 6, 7 and 8 describe the results and respective time taken for each of the simulation's attempts. Each attempt was executed with two distinct versions of Qiskit, a possible reason for why the time taken per iteration was changed so dramatically within the same device (A). (All results obtained in the second attempt can be consulted here[5].)

A possible reason for the correct answer not being found can be related to the first value the circuit finds, and hence, also related to the random number used as the initial threshold.

5 https://github.com/msofiasoliveira/MasterDissertation/tree/master/Quantum%20Search/Output_2attempt

Molecule 1C4V and 1TOM

First Attempt	Device used	A
	Iterations	135
	Time per iteration (seconds)	~ 1700
	Total time (hours)	~ 64
	Value and position	518; 0
Second Attempt	Device used	A
	Iterations	170
	Time per iteration (seconds)	~ 1400
	Total time (hours)	66
	Initial threshold	2000
	First Value and position	2000; 105
	Final value and position	518; 0

Table 6: Results of the quantum solution search and time taken for molecule $1C4V$ and $1TOM$.

Molecule 1C4V and 1D4P

First Attempt	Device used	A
	Iterations	135
	Time per iteration (seconds)	~ 1700
	Total time (hours)	~ 64
	Final value and position	502; 3
Second Attempt	Device used	B
	Iterations	170
	Time per iteration (seconds)	~ 1200
	Total time (hours)	~ 57
	Initial threshold	1110
	First Value and position	1056; 96
	Final value and position	509; 4

Table 7: Results of the quantum solution search and time taken for molecule $1C4V$ and $1D4P$.

In the first attempt, such values were not retrieved, but in the second attempt, it is possible to see that when a smaller value is found in the first iteration, the solution was successfully found. The opposite happens with higher values. One could only confirm such suspicions based on more attempts, which were impossible to obtain due to the time required.

Another possible reason is the proximity of the minimum values. To evaluate such a possibility, one needs to check the states throughout the circuit. The only way to do this is with

Molecule 1TOM and 1D4P

First Attempt	Device used	A
	Iterations	135
	Time per iteration (seconds)	~ 1700
	Total time (hours)	~ 64
	Final value and position	522; 60
Second Attempt	Device used	B
	Iterations	170
	Time per iteration (seconds)	~ 1070
	Total time (hours)	~ 51
	Initial threshold	2000
	First Value	640; 81
	Final Value and position	513; 11

Table 8: Results of the quantum solution search and time taken for molecule $1TOM$ and $1D4P$.

the *statevector_simulator*, backend of the Qiskit library. Unfortunately, with the hardware A[6] used in the first stage of the simulations, such was impossible. In a second attempt, another hardware, B[7], was used and the code started to run straight with 10Gb of RAM memory. A possible explanation to the first attempt of simulation not being able to run is the lack of RAM. After five minutes the usage was of 80% of total RAM memory, 13Gb. Due to the quick exhaustion of the RAM memory, it was impossible to run the simulation until the end.

The statevector operates by plotting the states of the qubits after each gate in the circuit. Since in this algorithm there are hundreds of gates and 8 qubits (regarding the different solutions), this can be a possible reason for this big usage of memory. Remember that although there are only 8 qubits relative to the different solutions, in total the circuit uses 30 qubits.

It is now important to clarify some time related data. In the classic search, the time taken can be considered irrelevant. On the other hand, with the quantum search algorithm used, the circuit of each case needs to be run 170 times, and each circuit takes around $20 - 28$ minutes[8]. That sums up to 63 hours, around two days and an half, for each case. Of course, given that it was only used the lists of two molecules alignment, such time quickly becomes impracticable for bigger molecules. The number of iterations required in each case is calculated by Equation (37) with $N = 144$ since this is the number of solutions in each of the lists.

6 Check table 9 in appendix C.
7 Check table 10 in appendix C.
8 Such number depends on the hardware of the device used

When comes to comparison between the two approaches, is clear that no advantage is achieved with a quantum algorithm, at least, at this point in time. Maybe in the future, with more qubits, such difference can be reversed.

8 CONCLUSIONS

When comes to creating a new algorithm, classic or quantum, it is always a challenge to find the best path to follow. Each decision along the way defines a little of the road that was taken. That being said, this work was an unsteady and adventurous road, full of new obstacles and dilemmas that could never be expected. Additionally, came a lot of interesting problems and subjects one could never imagine would learn. The learning curve has done nothing but to grow during this dissertation. Beyond that learning, despite the efforts, there were problems that could not be overcome and had a great impact in the results of this dissertation. The next section makes an overall analysis of what technical issues have been encountered and how they contributed so negatively to the success of the algorithms created. Finally, we will discuss alternative approaches and work that can be done in the future regarding the problem of flexible molecular alignment.

8.1 TECHNICAL ISSUES

The most important and most mentioned technical problem in this dissertation is the number of available qubits. Despite the knowledge, beforehand, that one would need to abstract the problem to a maximum, it was not anticipated how much abstraction that would actually imply. After all, the problem proposed was never solved itself in any quantum perspective. At least, not by beginning to end. By a difference of only 3 qubits, there was possible to run the qSAT algorithm with real data on a real device. When comes to the solution search quantum algorithm, there was only the possibility to use real data in the algorithm when resorting to a quantum simulator.

It is also important to mentioned, although this is not a technical issue, that the unfamiliarity with medicinal chemistry, and hence, molecular alignment, also had a great impact on the initial approach to the problem. It must be said as well that this unfamiliarity entailed the need to create a whole classical algorithm in order to better understand the problem and which approaches and abstractions could be proposed while still respecting the problem itself.

Continuing with the technical issues, the lack of memory in the local devices used, was also a problem. Or, in another perspective, the excessive need of memory by the qiskit software. Regardless, this was an issue that influenced the number of trials and tests one was able to execute since it required a computer device to be solely dedicated to the simulation, taking several days in some cases. A possible solution to such a problem could be provided by the use of online backends or simulators. Unfortunately, that generates data that could only be used for further computations if there was a steady internet connection. Although it is possible to retrieve the information about the circuit run online, this does not include data in treatable format.

Despite that point, online simulators and backends were used in all qSAT solver simulations. Nevertheless, due to queue waiting time or maybe other connection related reasons, it also took several hours to generate results. In the end, all solution regarding quantum simulations, on real devices or quantum simulators, took a lot more time than any classic algorithm. As mentioned previously, there is no fair comparison when comes to evaluate algorithms in systems with such resource disparity, but it is the only viable approach in a near future.

In the end and to sum up, the biggest problem encountered in this dissertation was the time and resources required for each simulation. It made it impossible to create or verify all algorithms created in a practical manner with all the real data provided by Bial.

8.2 FUTURE WORK

The future work regarding this project has a variety branches. Indeed, there are a lot of different paths entailing different degrees of research. Some more complex than others but in the end, they would all greatly contributed to a better success of this case study.

As an example of future work not too far from the approaches taken in this dissertation, all algorithms could be improved. For the qSAT, this involves the development and implementation of the QP formulations of the restrictions. For the solution search, on the other hand, the weight factor could be further refined to see its implications in the optimization of the search.

Additionally, if all algorithms could be run several times with different input parameters, (*i.e.* different data, number of circuit iterations, number of shots, number of `max_trials`), it would be possible to study the impact of each of these parameters in the simulation.

A promising future work is the development of a quantum algorithm that would be able to make the spatial alignment of the molecules, *i.e.* the pre-alignment. Such could be inspired on the QSSA algorithm, presented in section 3.3. Although not classically efficient, since elaborated quantum calculations are required, with a quantum approach that could be greatly changed. This idea came from the essence of quantum devices, that this, quantum mechanics. The calculations could never be required if one executed the simulation as a system of molecules that need to aligned according to the laws of physics and chemistry. In such a small scale, once again, reality is quantum. In such an algorithm, qubits would not be edges or points but atoms and all its respective peculiarities, which would improved the accuracy and reliability of the data produced.

Since the spatial alignment of molecules is such an important step in molecular alignment, the sole achievement of such could revolutionize the field of medicinal chemistry. Stepping back a little further in the problem, although extremely ambitious, the development of such an algorithm would create the possibility to treat the molecular flexibility as a direct parameter, instead of the semiflexible approach used during this dissertation. Even the promises of integrating the flexibility of molecules directly in the alignment with an efficient quantum algorithm would lead to a breakthrough within the science community and even industries that investigate this subject.

In any case, all the mentioned suggestions require time and resources allied with great knowledge of a variety of fields, including Physics, Chemistry, Mathematics and Computer Science. It is probably a theme for a broad scope research project.

BIBLIOGRAPHY

Metastable state. Encyclopdia Britannica website, 2018. URL https://www.britannica.com/science/metastable-state. Acessed: 03-07-2020.

Automatic compilation of quantum circuits, Apr 2020. URL https://quantumcomputing.stackexchange.com/questions/4086/automatic-compilation-of-quantum-circuits. Acessed: 16-06-2020.

Pierre-Gilles de Gennes A. Y. Grosberg, A. R. Khokhlov. *Flexibility Mechanisms*. World Scientific, 5 Toh Tuck Link, Singapore 596224, 2nd edition, 2011.

Ruben Abagyan and Maxim Totrov. Biased probability monte carlo conformational searches and electrostatic calculations for peptides and proteins. *Journal of Molecular Biology*, 235 (3):983 – 1002, 1994. ISSN 0022-2836. doi: https://doi.org/10.1006/jmbi.1994.1052. URL http://www.sciencedirect.com/science/article/pii/S0022283684710527.

Tameem Albash and Daniel A. Lidar. Adiabatic quantum computation. *Reviews of Modern Physics*, 90(1), Jan 2018. ISSN 1539-0756. doi: 10.1103/revmodphys.90.015002. URL http://dx.doi.org/10.1103/RevModPhys.90.015002.

Carolina Alves. A Quantum Algorithm for Ray Casting using Orthographic Camera. Master's thesis, University of Minho, Portugal, 2019.

Daniel Baum. Multiple semi-flexible 3d superposition of drug-sized molecules. In Michael R. Berthold, Robert C. Glen, Kay Diederichs, Oliver Kohlbacher, and Ingrid Fischer, editors, *Computational Life Sciences*, pages 198–207, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31726-5.

Paul J. Besl and Neil D. McKay. Method for registration of 3-D shapes. In Paul S. Schenker, editor, *Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 586–606. International Society for Optics and Photonics, SPIE, 1992. doi: 10.1117/12.57955.

Denise D. Beusen and Garland R. Marshall. Pharmacophore definition using the active analog approach. In Osman F. Güner, editor, *Pharmacophore Perception, Development, and Use In Drug Design*, chapter 3, pages 17–45. International University Line, La Jolla, California, 2000.

Gold Book. Compendium of chemical terminology. *International Union of Pure and Applied Chemistry*, 2014. Acessed: 24-06-2019.

O. Bottema and B. Roth. *Theoretical kinematics*. Dover Publications, New York, 1 edition, 1990. ISBN 0-486-66346-9.

Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *arXiv*, May 1996. doi: 10.1002/(SICI)1521-3978(199806)46:4/5⟨493::AID-PROP493⟩3.0.CO;2-P.

Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(45):493–505, 1998. doi: 10.1002/(SICI)1521-3978(199806)46:4/5⟨493::AID-PROP493⟩3.0.CO;2-P. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291521-3978%28199806%2946%3A4/5%3C493%3A%3AAID-PROP493%3E3.0.CO%3B2-P.

Stephen G. Brush. History of the lenz-ising model. *Rev. Mod. Phys.*, 39:883–893, Oct 1967. doi: 10.1103/RevModPhys.39.883. URL https://link.aps.org/doi/10.1103/RevModPhys.39.883.

P. Bultinck, W. Langenaeker, P. Lahorte, F. De Proft, P. Geerlings, C. Van Alsenoy, and J. P. Tollenaere. The electronegativity equalization method ii: Applicability of different atomic charge schemes. *The Journal of Physical Chemistry A*, 106(34):7895–7901, 2002a. doi: 10.1021/jp020547v.

P. Bultinck, W. Langenaeker, P. Lahorte, F. De Proft, P. Geerlings, M. Waroquier, and J. P. Tollenaere. The electronegativity equalization method i: Parametrization and validation for atomic charge calculations. *The Journal of Physical Chemistry A*, 106(34):7887–7894, 2002b. doi: 10.1021/jp0205463.

Patrick Bultinck, Tom Kuppens, Xavier Girons, and Ramon Carb-Dorca. Quantum similarity superposition algorithm (qssa): a consistent scheme for molecular alignment and molecular similarity based on quantum chemistry. *Journal of Chemical Information and Computer Sciences*, 43(4):1143–1150, 2003. doi: 10.1021/ci0340153. URL https://doi.org/10.1021/ci0340153.

Ramon Carb, Luis Leyda, and Mariano Arnau. How similar is a molecule to another? an electron density measure of similarity between two molecular structures. *International Journal of Quantum Chemistry*, 17(6):1185–1189, 1980. doi: 10.1002/qua.560170612. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/qua.560170612.

Alba Cervera-Lierta. Exact ising model simulation on a quantum computer. *Quantum*, 2: 114, 2018. doi: 10.22331/q-2018-12-21-114.

Henderson James (Jim) Cleaves. *Moiety*, pages 1069–1069. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-11274-4. doi: 10.1007/978-3-642-11274-4_1873. URL https://doi.org/10.1007/978-3-642-11274-4_1873.

Hans-Uwe Dahms. Challenges in medicinal chemistry. *J Chem Bio and Med Chem.*, 1(1), November 2017.

Matthieu E. Deconinck and Barbara M. Terhal. Qubit state discrimination. *Phys. Rev. A*, 81: 062304, Jun 2010. doi: 10.1103/PhysRevA.81.062304. URL https://link.aps.org/doi/10.1103/PhysRevA.81.062304.

David J Diller and Kenneth M Merz. Can we separate active from inactive conformations? *Journal of computer-aided molecular design*, 16(2):105–112, 2002.

David P. DiVincenzo. Quantum computation. *Science*, 270(5234):255–261, 1995. ISSN 0036-8075. doi: 10.1126/science.270.5234.255. URL https://science.sciencemag.org/content/270/5234/255.

John H. Van Drie. Monty kier and the origin of the pharmacophore concept. *Internet Electronic Journal of Molecular Design*, 6(9):271–279, 2007. ISSN 1538-6414. URL http://www.biochempress.com/av06_0271.html.

Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum. *CoRR*, quant-ph/9607014, 07 1996.

EUPATI. Active molecule, Oct 2016. URL https://www.eupati.eu/glossary/active-molecule/. Acessed: 23-06-2019.

E. Farhi, J. Goldstone, S. Gutmann, and H. Neven. Quantum algorithms for fixed qubit architectures, 2017.

Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution, 2000.

Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014.

Miklos Feher and Jonathan M. Schmidt. Multiple flexible alignment with seal: a study of molecules acting on the colchicine binding site. *Journal of Chemical Information and Computer Sciences*, 40(2):495–502, 2000. doi: 10.1021/ci9900682.

David M. Ferguson and Douglas J. Raber. A new approach to probing conformational space with molecular mechanics: random incremental pulse search. *Journal of the American Chemical Society*, 111(12):4371–4378, Jun 1989. ISSN 0002-7863. doi: 10.1021/ja00194a034. URL https://doi.org/10.1021/ja00194a034.

Richard P Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6/7), 1982.

Andrew W Fitzgibbon. Robust registration of 2d and 3d point sets. *Image and Vision Computing*, 21(13):1145 – 1153, 2003. ISSN 0262-8856. doi: https://doi.org/10.1016/j.imavis.2003.09.004. URL http://www.sciencedirect.com/science/article/pii/S0262885603001835. British Machine Vision Computing 2001.

J. A. Grant and B. T. Pickup. A gaussian description of molecular shape. *The Journal of Physical Chemistry*, 99(11):3503–3510, 1995. ISSN 0022-3654. doi: 10.1021/j100011a016.

J. A. Grant, M. A. Gallardo, and B. T. Pickup. A fast method of molecular shape comparison: A simple application of a gaussian description of molecular shape. *Journal of Computational Chemistry*, 17(14):1653–1666, 1996. doi: 10.1002/(SICI)1096-987X(19961115)17:14⟨1653::AID-JCC7⟩3.0.CO;2-K.

L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

Jun Gu, Paul W Purdom, John Franco, and Benjamin W Wah. Algorithms for the satisfiability (sat) problem: A survey. Technical report, Cincinnati Univ oh Dept of Electrical and Computer Engineering, 1996.

Gamini Gunawardena, Jun 2019. URL https://chem.libretexts.org/Ancillary_Materials/Reference/Organic_Chemistry_Glossary/Constitutional_Isomers. Acessed: 26-06-2019.

J. M. Hammersley and D. C. Handscomb. *Monte carlo methods*. Chapman and Hall, 1964. doi: 10.1007/978-94-009-5819-7.

Anne Marie Helmenstine. Macromolecule definition and examples, April 2018. URL https://www.thoughtco.com/definition-of-macromolecule-605324. Acessed: 24-06-2019.

Alexander von Homeyer. *A Superimposition Method for Small Ligand Molecules: Implementation and Application*. PhD thesis, Naturwissenschaftlichen Fakultten der Universitt Erlangen-Nrnberg, 2007.

Ian Hunt. Constitutional isomers, 2009a. URL http://www.chem.ucalgary.ca/courses/350/Carey5th/Ch01/ch1-5.html. Acessed: 04-07-2019.

Ian Hunt. Isomer types, 2009b. URL http://www.chem.ucalgary.ca/courses/350/Carey5th/Ch07/ch7-1.html. Acessed: 27-06-2019.

IBM. Qiskit aqua: Experimenting with max-cut problem and traveling salesman problem with variational quantum eigensolver, a. URL https://github.com/Qiskit/qiskit-tutorials/blob/master/legacy_tutorials/aqua/optimization/max_cut_and_tsp.ipynb. Accessed: 30-04-2020.

IBM. Qiskit aqua: Vehicle routing, b. URL https://github.com/Qiskit/qiskit-tutorials/blob/master/legacy_tutorials/aqua/optimization/vehicle_routing.ipynb. Accessed: 30-04-2020.

IBM. Grover's algorithm, c. URL https://quantum-computing.ibm.com/docs/guide/q-algos/grover-s-algorithm?fbclid=IwAR3a4UP-Q0fGSocmHe3qPEjx0bdqus5fA0lxGdwC4YbwDyQxRoGTifUlv-Y. Acessed: 17-06-2020.

IBM. Ibm unveils world's first integrated quantum computing system for commercial use, 2019. URL https://newsroom.ibm.com/2019-01-08-IBM-Unveils-Worlds-First-Integrated-Quantum-Computing-System-for-Commercia Acessed: 29-01-2020.

Abhijith J., Adetokunbo Adedoyin, John Ambrosiano, Petr Anisimov, Andreas Brtschi, William Casper, Gopinath Chennupati, Carleton Coffrin, Hristo Djidjev, David Gunter, Satish Karra, Nathan Lemons, Shizeng Lin, Alexander Malyzhenkov, David Mascarenas, Susan Mniszewski, Balu Nadiga, Daniel O'Malley, Diane Oyen, Scott Pakin, Lakshman Prasad, Randy Roberts, Phillip Romero, Nandakishore Santhi, Nikolai Sinitsyn, Pieter J. Swart, James G. Wendelberger, Boram Yoon, Richard Zamora, Wei Zhu, Stephan Eidenbenz, Patrick J. Coles, Marc Vuffray, and Andrey Y. Lokhov. Quantum algorithm implementations for beginners, 2020.

Ajay N. Jain. Scoring noncovalent protein-ligand interactions: A continuous differentiable function tuned to compute binding affinities. *Journal of Computer-Aided Molecular Design*, 10(5):427–440, 1996. ISSN 1573-4951. doi: 10.1007/BF00124474. URL https://doi.org/10.1007/BF00124474.

Ajay N. Jain. Surflex: fully automatic flexible molecular docking using a molecular similarity-based search engine. *Journal of Medicinal Chemistry*, 46(4):499–511, 2003. doi: 10.1021/jm020406h.

Rafik Karaman. What is pharmacophore, 2016. URL https://www.researchgate.net/post/What_is_Pharmacophore. Accessed: 04-03-2019.

Simon K. Kearsley and Graham M. Smith. An alternative method for the alignment of molecular structures: Maximizing electrostatic and steric overlap. *Tetrahedron Computer Methodology*, 3(6, Part C):615 – 633, 1990. ISSN 0898-5529. doi: https://doi.org/10.1016/0898-5529(90)90162-2. Three-dimensional chemical structure handling.

Siavash Khallaghi. Pycpd: Tutorial on the coherent point drift algorithm, May 2017. URL http://siavashk.github.io/2017/05/14/coherent-point-drift/. Acessed: 06-03-2020.

Oliver Korb, Peter Monecke, Gerhard Hessler, Thomas Sttzle, and Thomas E. Exner. pharmacophore: Multiple flexible ligand alignment based on ant colony optimization. *Journal of Chemical Information and Modeling*, 50(9):1669–1681, 2010. doi: 10.1021/ci1000218.

Danai Koutra, Ankur Parikh, Aaditya Ramdas, and Jing Xiang. Algorithms for graph similarity and subgraph matching. In *Proc. Ecol. Inference Conf*, volume 17, 2011.

Paul Labute, Chris Williams, Miklos Feher, Elizabeth Sourial, and Jonathan M. Schmidt. Flexible alignment of small molecules. *Journal of Medicinal Chemistry*, 44(10):1483–1490, 2001. doi: 10.1021/jm0002634.

Polo C-H Lam, Ruben Abagyan, and Maxim Totrov. Ligand-biased ensemble receptor docking (ligbend): a hybrid ligand/receptor structure-based approach. *Journal of computer-aided molecular design*, 32(1):187–198, Jan 2018. ISSN 1573-4951. doi: 10.1007/s10822-017-0058-x. URL https://www.ncbi.nlm.nih.gov/pubmed/28887659.

Christian Lemmen and Thomas Lengauer. Computational methods for the structural alignment of molecules. *Journal of Computer-Aided Molecular Design*, 14, 04 2000. doi: 10.1023/a:1008194019144.

Tracy Levin, 2000. URL http://cs.smith.edu/~istreinu/Teaching/Courses/274/Spring00/StudProj/Tracy/project3.html. Acessed: 25-06-2019.

Andrew Lucas. Ising formulations of many np problems. *Frontiers in Physics*, 2:5, 2014. ISSN 2296-424X. doi: 10.3389/fphy.2014.00005. URL https://www.frontiersin.org/article/10.3389/fphy.2014.00005.

Luis Gonzlez MacDowell. Flexible molecules, 2003. URL http://catalan.quim.ucm.es/html.uk/invest/molflex/molflex.htm. Acessed: 26-06-2019.

J. Marialke, R. Krner, S. Tietze, and Joannis Apostolakis. Graph-based molecular alignment (gma). *Journal of Chemical Information and Modeling*, 47(2):591–601, 2007. doi: 10.1021/ci600387r.

J. M. McCarthy. *Introduction to Theoretical kinematics*. MDA Press, 2013. available on iPad through iBookstore.

C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326329, October 1960. ISSN 0004-5411. doi: 10.1145/321043.321046. URL https://doi.org/10.1145/321043.321046.

Molsoft. Molsoft icm d3r docking challenge success, 2017. URL http://www.molsoft.com/news.html#n2017. Acessed: 12-02-2020.

Molsoft. Molsoft outperforms a range of other methods for docking pose and affinity prediction accuracy in d3r grand challenge 3, 2018. URL http://www.molsoft.com/news.html#gc3. Acessed: 12-02-2020.

G.P. Moss. Basic terminology of stereochemistry (iupac recommendations 1996). *Pure and Applied Chemistry*, 68(12):21932222, 1996. doi: https://doi.org/10.1351/pac199668122193.

A. Myronenko and X. Song. Point set registration: Coherent point drift. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(12):2262–2275, Dec 2010. ISSN 0162-8828. doi: 10.1109/TPAMI.2010.46.

Ana Neri and Afonso Rodrigues. Ibm q experience. Lecture notes from the class Quantum Computation ,MAPi Doctoral Programme in Computer Science, 2018. URL https://anac.nery.name/static/media/IBM_Q_Experience___class.27ba5124.pdf?fbclid=IwAR1-pB5ezfayf_1lP8A0B0qVmUj8xml2mFjzlyD-iWElP5TVocWpn3mB3EE. Acessed: 17-06-2020.

Marco Neves. Project ideas. Powerpoint presentation given by Marco Neves, at BIAL (pharmaceutic company), when was first presented the case study present in this master dissertation., 2018.

Marco A. C. Neves, Maxim Totrov, and Ruben Abagyan. Docking and scoring with icm: the benchmarking results and strategies for improvement. *Journal of computer-aided molecular design*, 26(6):675–686, Jun 2012. ISSN 1573-4951. doi: 10.1007/s10822-012-9547-0. URL https://www.ncbi.nlm.nih.gov/pubmed/22569591.

Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, USA, 10th edition, 2011. ISBN 1107002176.

Lakna Panawala. What is the active site of an enzyme, May 2017. URL https://pediaa.com/what-is-the-active-site-of-an-enzyme/. Acessed: 24-06-2019.

Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1):4213, Jul 2014. ISSN 2041-1723. doi: 10.1038/ncomms5213. URL https://doi.org/10.1038/ncomms5213.

John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018. ISSN 2521-327X. doi: 10.22331/q-2018-08-06-79. URL https://doi.org/10.22331/q-2018-08-06-79.

Xiao-Yu Qing, Samantha Lee, Joren De Raeymaecker, Jeremy Tame, Kam Zhang, Marc De Maeyer, and Arnout Voet. Pharmacophore modeling: Advances, limitations, and current utility in drug discovery. *Journal of Receptor, Ligand and Channel Research*, 7:81–92, 11 2014. doi: 10.2147/JRLCR.S46843.

John W. Raymond and Peter Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16(7): 521–533, 2002. ISSN 1573-4951. doi: 10.1023/A:1021271615909. URL https://doi.org/10.1023/A:1021271615909.

Douglas A Reynolds. Gaussian mixture models. *Encyclopedia of biometrics*, 741, 2009.

Thomas S. Rush, J. Andrew Grant, Lidia Mosyak, and Anthony Nicholls. A shape-based 3-d scaffold hopping method and its application to a bacterial protein-protein interaction. *Journal of Medicinal Chemistry*, 48(5):1489–1495, 2005. ISSN 0022-2623. doi: 10.1021/jm0401630.

Dina Schneidman-Duhovny, Oranit Dror, Yuval Inbar, Ruth Nussinov, and Haim J Wolfson. Deterministic pharmacophore detection via multiple flexible alignment of drug-like molecules. *Journal of Computational Biology*, 15(7):737–754, 2008. doi: 10.1089/cmb.2007.0130.

Peter H. Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, Mar 1966. ISSN 1860-0980. doi: 10.1007/BF02289451. URL https://doi.org/10.1007/BF02289451.

P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

Wolfgang Sippl. Chapter 28 - pharmacophore identification and pseudo-receptor modeling. In Camille Georges Wermuth, editor, *The Practice of Medicinal Chemistry (Third Edition)*, pages 572–586. Academic Press, New York, third edition edition, 2008. ISBN 978-0-12-374194-3. doi: https://doi.org/10.1016/B978-0-12-374194-3.00028-7. URL http://www.sciencedirect.com/science/article/pii/B9780123741943000287.

Michael B. Smith. Chapter 1 - retrosynthesis, stereochemistry, and conformations. In Michael B. Smith, editor, *Organic Synthesis (Third Edition)*, pages 1–76. Academic Press, Oxford, third edition edition, 2010. ISBN 978-1-890661-40-3. doi: https://doi.org/10.1016/B978-1-890661-40-3.50001-6. URL http://www.sciencedirect.com/science/article/pii/B9781890661403500016.

T. W. Graham Solomons and Craig B. Fryhle. *Organic Chemistry*. John Wiley & Sons, Inc., 10 edition, 2009. ISBN 9780470556597.

Jonatan Taminau, Gert Thijs, and Hans De Winter. Pharao: Pharmacophore alignment and optimization. *Journal of Molecular Graphics and Modelling*, 27(2):161–169, 2008. ISSN 1093-3263. doi: https://doi.org/10.1016/j.jmgm.2008.04.003.

Carlos Tavares, Sofia Oliveira, Vitor Fernandes, Andrei Postnikov, and Mikhail Vasilievskiy. Quantum simulation of the groundstate stark effect in small molecules: A case study using ibm q. *Soft Computing*. Submitted.

Maxim Totrov. Atomic property fields: Generalized 3d pharmacophoric potential for automated ligand superposition, pharmacophore elucidation and 3d qsar. *Chemical Biology & Drug Design*, 71(1):15–27, 2008. doi: 10.1111/j.1747-0285.2007.00605.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1747-0285.2007.00605.x.

Nenad Trinajstic. *Chemical Graph Theory*. CRC Press, 2 edition, February 1992. ISBN 9780849342561.

AS Verkman. Drug discovery in academia. *American Journal of Physiology-Cell Physiology*, 286(3):C465–C474, 2004. doi: 10.1152/ajpcell.00397.2003.

Christophe LMJ Verlinde and Wim GJ Hol. Structure-based drug design: progress, results and challenges. *Structure*, 2(7):577 – 587, 1994. ISSN 0969-2126. doi: https://doi.org/10.1016/S0969-2126(00)00060-5. URL http://www.sciencedirect.com/science/article/pii/S0969212600000605.

Dave Wecker, Matthew B. Hastings, and Matthias Troyer. Training a quantum optimizer. *Phys. Rev. A*, 94:022309, Aug 2016. doi: 10.1103/PhysRevA.94.022309. URL https://link.aps.org/doi/10.1103/PhysRevA.94.022309.

C.G. Wermuth, C.R. Ganellin, Per Lindberg, and Lester Mitscher. Glossary of terms used in medicinal chemistry. *Pure and Applied Chemistry - PURE APPL CHEM*, 70(5):1129–1143, 1998. doi: https://doi.org/10.1351/pac199870051129.

Douglas B. West. *Introduction to Graph Theory*. Pearson Education, 2 edition, 2001. ISBN 81-7808-830-4.

John Wright. Quantum information theory and holevos bound, 2015. URL https://www.cs.cmu.edu/~odonnell/quantum15/lecture18.pdf. Acessed: 30-01-2020.

J. Wu, Y. Wan, and Z. Su. Bayesian rigid point set registration using logarithmic double exponential prior. In *2013 IEEE Third International Conference on Information Science and Technology (ICIST)*, pages 1360–1364, March 2013. doi: 10.1109/ICIST.2013.6747790.

Sheng-Yong Yang. Pharmacophore modeling and applications in drug discovery: challenges and recent advances. *Drug Discovery Today*, 15(11):444–450, 2010. ISSN 1359-6446.

doi: https://doi.org/10.1016/j.drudis.2010.03.013. URL http://www.sciencedirect.com/science/article/pii/S135964461000111X.

P. Závodszky, J. Kardos, Svingor, and G. A. Petsko. Adjustment of conformational flexibility is a key event in the thermal adaptation of proteins. *Proceedings of the National Academy of Sciences of the United States of America*, 95(13):7406–7411, Jun 1998. ISSN 0027-8424. doi: 10.1073/pnas.95.13.7406. URL https://www.ncbi.nlm.nih.gov/pubmed/9636162.

Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2):119–152, 1994. ISSN 1573-1405. doi: 10.1007/BF01427149.

GLOSSARY

A

active conformation conformation of a molecule as it binds to target proteins (Diller and Merz, 2002). 13

active molecule In research and development (R&D) of medicines, is a chemical compound that has pharmacological or biological activity likely to be therapeutically useful (EUPATI, 2016). 9

active site is the region of an enzyme where substrate molecules bind (Panawala, 2017). 10

agonist Substance which binds to cell receptors normally responding to naturally occurring subtances which produces a response of its own (Book, 2014). 127

antagonist Substance that reverses or reduces the effect induced by an agonist; Substance that attaches to and blocks cell receptors that normally bind naturally occurring substances (Book, 2014). xi, xii, 10, 18

B

binding site Along with the catalytic site, form the active site of the enzyme. The binding site binds and orients the substrate (Panawala, 2017). 4, 10, 11

C

catalyst substance that increases the rate of transfer of a reaction without affecting the position of equilibrium (Book, 2014). 10

catalytic site Along with the binding site, form the active site of the enzyme. The catalytic site carries out the catalysis, reducing the chemical activation energy (Panawala, 2017) . 127

conformation The spatial arrangement of the atoms affording distinction between stereoisomers which can be interconverted by rotations about formally single bonds (Moss, 1996) . v, xi–xiv, xvii, 4, 12–15, 17, 18, 20, 21, 24–26, 31–35, 37, 38, 43, 50–52, 54, 55, 61, 66, 67, 77–80, 101, 102, 105, 106, 127

conformational analysis consists of the exploration of energetically favorable spatial arrangements (shapes) of a molecule (conformations) using molecular mechanics, molecular dynamics, quantum chemical calculations or analysis of experimentally-determined structural data, e.g., NMR or crystal structures. Molecular mechanics and quantum chemical methods are employed to compute conformational energies, whereas systematic and random searches, Monte Carlo, molecular dynamics, and distance geometry are methods (often combined with energy minimization procedures) used to explore the conformational space (Book, 2014). 13

conformational flexibility flexibility/fluctuations of a molecule's conformations due to internal energy fluctuations (Závodszky et al., 1998). 11, 13, 14

conformational isomer One of a set of stereoisomers, each of which is characterized by a conformation corresponding to a distinct potential energy minimum (Moss, 1996). 13, 128

conformational space the space encompasing all possible positions of the molecule (Levin, 2000). 11, 13, 14, 22, 40, 43, 128

conformer Contracted version of conformational isomer. xii, 12–14, 19, 20

constitutional formulae or structural formulae, is a formula which gives information about the way the atoms in a molecule are connected and arranged in space (Book, 2014). 35, 128, 129

constitutional isomers or structural isomers are compounds with the same molecular formula but different structural formulas (constitution) (Gunawardena, 2019; Book, 2014). 12

E

enzyme is a protein molecule that can act as a biological catalyst. The molecules that enzymes act upon are called substrates (Panawala, 2017). 4, 127

F

functional group atom, or a group of atoms, that has similar chemical properties whenever it occurs in different compounds (Book, 2014). 1, 9, 129

G

GMM (*Gaussian Mixture Model*), is a parametric probability density function represented as a weighted sum of Gaussian component densities (Reynolds, 2009). xiv, 46–48, 50

H

Hamiltonian energy function (wavefunction) of a physical system. Quantum mechanical operator associated with the system energy. The system energy is composed by the kinetic and potential energy.. 28, 29, 69–77, 81, 83, 84, 105

I

isomer One of several molecular entities that have the same molecular formula but different constitutional formulae or different *stereochemical formulae* (Moss, 1996). 12, 13, 129

L

ligand If it is possible or convenient to regard part of a polyatomic molecular entity as central, then the atoms, groups or molecules bound to that part are called ligands (Book, 2014). xi–xiii, 3, 4, 9–11, 13, 21, 22, 24–26, 31, 32, 58

M

macromolecule a molecule with a very large number of atoms. Macromolecules typically have more than 100 component atoms (Helmenstine, 2018). 9, 10

metastable ,in physics and chemistry, is a particular excited state of an atom, nucleus, or other system that has a longer lifetime than the ordinary excited states and that generally has a shorter lifetime than the lowest, often stable, energy state, called the ground state. A metastable state may thus be considered a kind of temporary energy trap or a somewhat stable intermediate stage of a system the energy of which may be lost in discrete amounts (bri, 2018). 14

moieties plural of moiety. Parts of a molecule. 9

moiety is generally used to signify part of a molecule (Book, 2014); In organic chemistry, the term moiety is used to denote a portion of a molecule, which may be a functional group, or describe a portion of a molecule with multiple functional groups which share common structural aspects (Cleaves, 2011). 129

molecular formula For compounds consisting of discrete molecules, a formula according with the relative molecular mass (Book, 2014). 12, 128

O

oracle *black box*/operator capable of recognize solutions to a specific problem (Nielsen and Chuang, 2011). xiv, 71, 86, 87, 96

Q

quantum noise unwanted interactions within a real quantum system, that is, an open system. Example: if the state of a qubit is represented by two positions of a electron, then that electron will interact with other charged particles, which act as a source of uncontrolled noise affecting the state of the qubit (Nielsen and Chuang, 2011). 107

S

stereochemical formulae A three-dimensional view of a molecule either as such or in a projection (Book, 2014). xii, 31, 128, 129

stereoisomer Isomers that possess identical constitution, but which differ in the arrangement of their atoms in space (Moss, 1996). 12, 128

steric that has an effect on a chemical or physical property (structure, rate or equilibrium) of a molecule (Book, 2014). 9, 17, 18

structural formula see constitutional formulae. 128

superpose Bring two particular stereochemical formulae (or models) into coincidence (or to be exactly superposable in space, and for the corresponding molecular entities or objects to become exact replicas of each other) by no more than translation and rigid rotation (Book, 2014). 11, 13, 14, 23, 31, 32, 52, 80, 130

superposing The act of superpose. 9, 13

supramolecular relating to or denoting structures of two or more molecular entities held together and organized by means of intermolecular (noncovalent) binding interactions (Book, 2014). 9

T

toffoli gate reversible gate with three input bits and three output bits. Two of them are control bits, they are unaffected by the action of the gate. The third is the *target* bit that is flipped if both control bits are set to 1, and otherwise is left alone. Check more details in Nielsen and Chuang (2011). 71, 95, 96

A

CODE - A CLASSICAL APPROACH WITH PYTHON

Check the next link for entire codes.
https://github.com/msofiasoliveira/MasterDissertation/

A.1 IMPORTS REQUIRED

```
from gurobipy import *
import numpy as np
import math as m
import time
import sys
from itertools import combinations as comb
from scipy.special import comb as comb2
from functools import partial
from pycpd import rigid_registration
```

A.2 DATA TREATMENT

```
class Point( object ):
    def __init__( self, x, y, z, data ):
        self.x, self.y, self.z = x, y, z
        self.data = data

    def __str__ (self):
        return "Ponto do tipo %s. Coordenadas %s %s %s" % (self.data, self.x,
           self.y, self.z)
```

```
import csv

csv.register_dialect('myDialect', delimiter = ',', skipinitialspace=True)

with open('featureXYZ_update_new.csv', 'r') as csvDataFile:
    csvReader = csv.reader(csvDataFile, dialect = 'myDialect')

database = []

with open('featureXYZ_update_new.csv', 'r') as csvDataFile:
    csvReader = csv.reader(csvDataFile, dialect = 'myDialect')

mol = -1
molecule_type = ""
idconf = -1

for idx, row in enumerate(csvReader):
    if (row[0] != 'pdb'):
     if (molecule_type != row[0]):
                molecule_type = row[0]
                database.append([])
                mol = mol+1
                idconf= -1

            if( idconf != int(row[1])):
                print("new conformation: ",row[1])
                idconf = idconf+1
                database[mol].append([])

            types=""
            if "H" in row[2]:
                types="H"
            if "P" in row[2]:
                types ="P"

            print("Adding the point",types,"with coordinates ",row[3],row[4],row[5])
            database[mol][idconf].append(Point(float(row[3]),
                                               float(row[4]),
                                               float(row[5]),
                                               types))
```

A.3 MAIN CODE

In order to not over-extend this appendix, and because a large part of the main code was already approach during the document, refer to the following link to check the code.
https://github.com/msofiasoliveira/MasterDissertation/blob/master/Classic.ipynb

A.4 AUXILIARY ALTERED LIBRARIES

A.4.1 *CPD algorithm*

```
from builtins import super
import numpy as np
from .expectation_maximization_registration import
expectation_maximization_registration

class rigid_registration(expectation_maximization_registration):
    def __init__(self, R=None, t=None, s=None, *args, **kwargs):
        super().__init__(*args, **kwargs)
        if self.D != 2 and self.D != 3:
            message = 'Rigid registration only supports 2D or 3D point clouds.
            Instead got {}.'.format(self.D)
            raise ValueError(message)
        if s == 0:
            raise ValueError('A zero scale factor is not supported.')
        self.R = np.eye(self.D) if R is None else R
        self.t = np.atleast_2d(np.zeros((1, self.D))) if t is None else t
        self.s = 1 if s is None else s

    def update_transform(self):
        muX = np.divide(np.sum(np.dot(self.P, self.X), axis=0), self.Np)
        muY = np.divide(np.sum(np.dot(np.transpose(self.P), self.Y), axis=0),
            self.Np)
        self.XX = self.X - np.tile(muX, (self.N, 1))
        YY      = self.Y - np.tile(muY, (self.M, 1))

        self.A = np.dot(np.transpose(self.XX), np.transpose(self.P))
        self.A = np.dot(self.A, YY)

        U, _, V = np.linalg.svd(self.A, full_matrices=True)
        C = np.ones((self.D, ))
        C[self.D-1] = np.linalg.det(np.dot(U, V))

        self.R = np.transpose(np.dot(np.dot(U, np.diag(C)), V))
        self.YPY = np.dot(np.transpose(self.P1), np.sum(np.multiply(YY, YY),
            axis=1))
        self.s = 1
        self.t = np.transpose(muX) - self.s * np.dot(np.transpose(self.R),
             np.transpose(muY))

    def transform_point_cloud(self, Y=None):
        if Y is None:
            self.TY = self.s * np.dot(self.Y, self.R) + self.t
            return
        else:
            return self.s * np.dot(Y, self.R) + self.t

    def update_variance(self):
        qprev = self.q

        trAR = np.trace(np.dot(self.A, self.R))
        xPx = np.dot(np.transpose(self.Pt1), np.sum(np.multiply(self.XX, self.XX),
            axis =1))
        self.q = (xPx - 2 * self.s * trAR + self.s * self.s * self.YPY) /
          (2 * self.sigma2) + self.D * self.Np/2 * np.log(self.sigma2)
        self.err = np.abs(self.q - qprev)
        self.sigma2 = (xPx - self.s * trAR) / (self.Np * self.D)
        if self.sigma2 <= 0:
            self.sigma2 = self.tolerance / 10
    def get_registration_parameters(self):
        return self.s, self.R, self.t
```

Sub-Algorithm - Expectation Maximization Registration

```
import numpy as np

def initialize_sigma2(X, Y):
    (N, D) = X.shape
    (M, _) = Y.shape
    XX = np.reshape(X, (1, N, D))
    YY = np.reshape(Y, (M, 1, D))
    XX = np.tile(XX, (M, 1, 1))
    YY = np.tile(YY, (1, N, 1))
    diff = XX - YY
    err  = np.multiply(diff, diff)
    return np.sum(err) / (D * M * N)

class expectation_maximization_registration(object):
    def __init__(self, X, Y, sigma2=None, max_iterations=100, tolerance=0.001,
    w=0, *args, **kwargs):
        if type(X) is not np.ndarray or X.ndim != 2:
            raise ValueError("The target point cloud (X) must be at a
             2D numpy array.")
        if type(Y) is not np.ndarray or Y.ndim != 2:
            raise ValueError("The source point cloud (Y) must be a 2D
             numpy array.")
        if X.shape[1] != Y.shape[1]:
            raise ValueError("Both point clouds need to have the same
             number of dimensions.")

        self.X              = X
        self.Y              = Y
        self.sigma2         = sigma2
        (self.N, self.D)    = self.X.shape
        (self.M, _)         = self.Y.shape
        self.tolerance      = 0.5
        self.w              = w
        self.max_iterations = max_iterations
        self.iteration      = 0
        self.err            = self.tolerance + 1
        self.P              = np.zeros((self.M, self.N))
        self.Pt1            = np.zeros((self.N, ))
        self.P1             = np.zeros((self.M, ))
        self.Np             = 0

    def register(self, callback=lambda **kwargs: None):
        self.transform_point_cloud()
        if self.sigma2 is None:
            self.sigma2 = initialize_sigma2(self.X, self.TY)
        self.q = -self.err - self.N * self.D/2 * np.log(self.sigma2)
        while self.iteration < self.max_iterations and self.err > self.tolerance:
            self.iterate()
            if callable(callback):
                kwargs = {'iteration': self.iteration, 'error': self.err,
                     'X': self.X, 'Y': self.TY}
                callback(**kwargs)

        return self.TY, self.get_registration_parameters(), self.err

    def get_registration_parameters(self):
        raise NotImplementedError("Registration parameters should be defined
         in child classes.")

    def iterate(self):
        self.expectation()
        self.maximization()
        self.iteration += 1

    def expectation(self):
        P = np.zeros((self.M, self.N))

        for i in range(0, self.M):
            diff     = self.X - np.tile(self.TY[i, :], (self.N, 1))
            diff     = np.multiply(diff, diff)
            P[i, :]  = P[i, :] + np.sum(diff, axis=1)
        c = (2 * np.pi * self.sigma2) ** (self.D / 2)
        c = c * self.w / (1 - self.w)
        c = c * self.M / self.N

        P = np.exp(-P / (2 * self.sigma2))
        den = np.sum(P, axis=0)
        den = np.tile(den, (self.M, 1))
        den[den==0] = np.finfo(float).eps
        den += c

        self.P   = np.divide(P, den)
        self.Pt1 = np.sum(self.P, axis=0)
        self.P1  = np.sum(self.P, axis=1)
        self.Np  = np.sum(self.P1)

    def maximization(self):
        self.update_transform()
        self.transform_point_cloud()
        self.update_variance()
```

B

CODE - A QUANTUM APPROACH WITH QISKIT

Check the next link for entire codes.
https://github.com/msofiasoliveira/MasterDissertation/

B.1 QUANTUM SAT

```
class QuantumOptimizer:
    def __init__(self, d, p, max_trials=1000):

        self.d = d
        self.p = p
        self.size = len(d)
        self.max_trials = max_trials

def binary_representation(self,x_sol=[]):

        p = self.p
        d = self.d
        size = self.size
        ones = np.ones(size)

        g = [x / 2 for x in np.subtract(ones,d)]

        c = np.sum([x / 2 for x in np.subtract(d,ones)])

        try:
            fun = lambda x: np.dot(d,x) - np.dot(ones,x)
            cost = fun(x_sol)
        except:
            cost = 0

        return g, c, cost

def construct_hamiltonian(self):

        p = self.p
        d = self.d
        size = self.size

        N = size

        gz,cz, _ = self.binary_representation()

        pauli_list = []
        for i in range(N):
            if gz[i] != 0:
                wp = np.zeros(N)
                vp = np.zeros(N)
                vp[i] = 1
                pauli_list.append((gz[i], Pauli(vp, wp)))

        pauli_list.append((cz, Pauli(np.zeros(N), np.zeros(N))))

        return cz, pauli_list

    def check_hamiltonian(self):

        cz, op = self.construct_hamiltonian()
        Op = WeightedPauliOperator(paulis=op)

        qubitOp, offset = Op, 0

        result0 = NumPyEigensolver(operator=qubitOp).run()
        result = result0['eigenstates'].to_matrix(massive=True)

        quantum_solution = self._q_solution(np.real(result[0]).tolist(),self.size)
        ground_level = np.real(result0['eigenvalues'][0]) + offset

        return quantum_solution, ground_level

    def vqe_solution(self):

        cz, op = self.construct_hamiltonian()
        Op = WeightedPauliOperator(paulis=op)

        qubitOp, offset = Op, cz

        aqua_globals.random_seed = 10598

        num_qubits = qubitOp.num_qubits
        var_form = RY(qubitOp.num_qubits, depth=5, entanglement='linear')
        optimizer = SPSA(max_trials=self.max_trials)
        algo = VQE(qubitOp, var_form, optimizer)

        backend = provider.get_backend('ibmq_qasm_simulator')

        quantum_instance = QuantumInstance(backend,
                                           seed_simulator=aqua_globals.random_seed,
                                           seed_transpiler=aqua_globals.random_seed,
                                           skip_qobj_validation=False)
        result = algo.run(quantum_instance)

        quantum_solution_dict = result['eigenstate']

        q_s = max(quantum_solution_dict.items(), key=operator.itemgetter(1))[0]
        quantum_solution= [int(chars) for chars in q_s]
        quantum_solution = np.flip(quantum_solution, axis=0)

        _,_,level = self.binary_representation(x_sol=quantum_solution)
        return quantum_solution_dict, quantum_solution, level

    def _q_solution(self, v, N):
        for x in range(len(v)):
            if v[x] == max(v):
                index_value = x
                break

        string_value = "{0:b}".format(index_value)

        while len(string_value)<N:
            string_value = '0'+string_value

        sol = list()
        for elements in string_value:
            if elements == '0':
                sol.append(0)
            else:
                sol.append(1)

        sol = np.flip(sol, axis=0)

        return sol
```

B.2 SOLUTION SEARCH

```
def initial(data, ordem):
    N = len(data)
    y = randrange(N-1)
    v_threshold = data[y]
    M = v_threshold + 1

    nb=N-1
    qy = nb.bit_length()
    maxq=ordem.bit_length()
    if qy>maxq:
        qaux=qy
    else:
        qaux=maxq
    data_b = {}

    for i in range(len(data)):
        string_index = bin(i)[2:].zfill(qy)
        string_value = bin(data[i])[2:].zfill(maxq)
        data_b[string_index]=string_value

    dim = len(data_b[string_index])
    vv=-1
    data_t = {}
    data_simp = {}

    for a in range(dim):
        count=0
        vv=vv+1
        for x in data_b.values():
            if x[a]=='1':
                vs='v'+str(vv)
                #ps='P'+str(count)
                k=data_b
                ps=list(k)[count]
                if vs in data_t:
                    tps.append(ps)
                    data_t[vs]= tps
                else:
                    data_t[vs]= [ps]
                    tps=[ps]
            count=1+count
        if data_t != {}:
            data_simp[vs] = symp_aux(qy, data_t[vs])

    return data_simp, qy, maxq, qaux, N, M

def symp_aux(nq,ls):
    global i0,i3,i
    nq

    if nq <= 10:
```

```
             i, i0, i1, i2, i3, i4, i5, i6, i7, i8 =
                              symbols('i,i0, i1, i2, i3, i4, i5, i6, i7, i8')
        else:
            print('To many qubits')

        lis=[i0, i1, i2, i3, i4, i5, i6, i7, i8]

        def negifzero(x,y):
            if x=='0':
                na = ~y
            else:
                na = y
            return na

        def getand(l):
            for ll in range(len(l)):
                if ll==0:
                    na = negifzero(l[ll],i0)
                    a = na
                elif ll==1:
                    na = negifzero(l[ll],i1)
                    a = a & na
                elif ll==2:
                    na = negifzero(l[ll],i2)
                    a = a & na
                elif ll==3:
                    na = negifzero(l[ll],i3)
                    a = a & na
                elif ll==4:
                    na = negifzero(l[ll],i4)
                    a = a & na
                elif ll==5:
                    na = negifzero(l[ll],i5)
                    a = a & na
                elif ll==7:
                    na = negifzero(l[ll],i6)
                    a = a & na
                elif ll==8:
```

```
                na = negifzero(l[ll],i7)
                a = a & na
            else:
                na = negifzero(l[ll],i8)
                a = a & na

        return a

    def func_aux(i, lis, ls):
        a = i
        for l in range(len(ls)):
            if a != i:
                a = a | getand(ls[l])
            else:
                a = getand(ls[l])
        return a


    fa = func_aux(i, lis, ls)

    sfa = simplify_logic(fa)

    return sfa

def gate_cx(I, qt, qc,qr):
    dataIstr = str(I)
    if dataIstr[0]=='~':
        i= int(dataIstr[2])
        qc.x(qr[i])
        qc.cx(qr[i],qr[qt])
        qc.x(qr[i])
    else:
        i= int(dataIstr[1])
        qc.cx(qr[i],qr[qt])

def gate_toffoli(data, target, qc):
    v = list(data.values())
    k = list(data.keys())
```

```
        icf = int(str(k[0])[1])
        ics = int(str(k[1])[1])
        if v[0] and v[1]:
            qc.ccx(qr[icf],qr[ics],qr[target])
        elif v[0] and not(v[1]):
            qc.x(qr[ics])
            qc.ccx(qr[icf],qr[ics],qr[target])
            qc.x(qr[ics])
        elif not(v[0]) and v[1]:
            qc.x(qr[icf])
            qc.ccx(qr[icf],qr[ics],qr[target])
            qc.x(qr[icf])
        else:
            qc.x(qr[icf])
            qc.x(qr[ics])
            qc.ccx(qr[icf],qr[ics],qr[target])
            qc.x(qr[ics])
            qc.x(qr[icf])

def toffoli_init(c1,c2,target,qc,qr):
    icf = int(str(c1[0])[1])
    ics = int(str(c2[0])[1])
    if c1[1] and c2[1]:
        qc.ccx(qr[icf],qr[ics],qr[target])
    elif c1[1] and not(c2[1]):
        qc.x(qr[ics])
        qc.ccx(qr[icf],qr[ics],qr[target])
        qc.x(qr[ics])
    elif not(c1[1]) and c2[1]:
        qc.x(qr[icf])
        qc.ccx(qr[icf],qr[ics],qr[target])
        qc.x(qr[icf])
    else:
        qc.x(qr[icf])
        qc.x(qr[ics])
        qc.ccx(qr[icf],qr[ics],qr[target])
        qc.x(qr[ics])
        qc.x(qr[icf])
    return target

def toffoli_mid(c1, c2, target, qc,qr):
    ics = int(str(c2[0])[1])
    if c2[1]:
        qc.ccx(qr[c1],qr[ics],qr[target])
    elif not(c2[1]):
        qc.x(qr[ics])
        qc.ccx(qr[c1],qr[ics],qr[target])
        qc.x(qr[ics])
    return target

def toffoli_mid_rev(c1, c2, target, qc,qr):
    ics = int(str(c2[0])[1])
    if c2[1]:
        qc.ccx(qr[c1],qr[ics],qr[target])
    elif not(c2[1]):
        qc.x(qr[ics])
        qc.ccx(qr[c1],qr[ics],qr[target])
        qc.x(qr[ics])
    return target

def mark_or_zero(v,n,m,qc,qr):
    target = n+m
    control = n
    for i in v:
        if i=='1':
            qc.cx(qr[control],qr[target])
        control= control+1

def mark_zero(c,t,qc,qr):
    qc.cx(qr[c],qr[t])

def mark_other(c,t,qc,qr):
    lc = len(c)
    final_t=t
    if lc == 2:
        qc.x(qr[c[1]])
```

```
            qc.ccx(qr[c[0]],qr[c[1]],qr[t])
            qc.x(qr[c[1]])
        else:
            t=t+1
            qc.ccx(qr[c[0]],qr[c[1]],qr[t])
            ic_1=t
            t=t+1
            count=2
            for x in c[2:]:
                count=count+1
                ic_2=x
                if lc==count:
                    qc.x(qr[ic_2])
                qc.ccx(qr[ic_1],qr[ic_2],qr[t])
                if lc==count:
                    qc.x(qr[ic_2])
                ic_1=t
                t=t+1
            qc.cx(qr[ic_1],qr[final_t])
            #reverse
            t=t-1
            ic_1=ic_1-1
            for x in reversed(c[1:-1]):
                if lc==count:
                    qc.x(qr[ic_2])
                qc.ccx(qr[ic_1],qr[ic_2],qr[t])
                if lc==count:
                    qc.x(qr[ic_2])
                ic_1=ic_1-1
                t=t-1
                ic_2=x
                count=count-1
            #t = t-1
            qc.ccx(qr[c[0]],qr[c[1]],qr[t])

def findsolution(data,method,backend):
    try:
        top = max(data)
```

```
data_simp, qy, maxq, qaux, N, M = initial(data, top)

y = randrange(N-1)
v_threshold = data[y]
v_threshold_old = top+1

m0= int((45/4)*(m.sqrt(N)))

# method ='Grover'
J = methods(N,M,method)

#use qasm_simulator
backend = Aer.get_backend(backend)
shots = 1

print("qy:",qy," maxq:", maxq," qaux:", qaux," N:", N," M:",
      M ," m0:", m0," J:", J,"initial v_threshold: ",v_threshold)

if qy+maxq+qaux > 32:
    raise Exception('too many qubits')

for i in range(m0):
    print("Iteration number ",i+1," of",m0)
    if (v_threshold != v_threshold_old):
        print("           Making Circuit")
        start = time.time()
        grover_circuit = grover(qy, maxq, qaux, data_simp, v_threshold,J)
        print("           time taken (seconds):",time.time() - start)


    job=execute(grover_circuit, backend, shots=shots)
    print("           Running Circuit")
    start = time.time()
    result_S = job.result()
    print("           time taken (seconds):",time.time() - start)

    print("           Getting Results")
    start = time.time()
```

```
            counts_sim = result_S.get_counts(grover_circuit)
            print("           time taken (seconds):",time.time() - start)

            y_temp=int(list(counts_sim.keys())[0], 2)

            v_threshold_old = v_threshold
            if y_temp < len(data):
                if data[y_temp] <= v_threshold:
                    v_threshold = data[y_temp]
                    a = y_temp
                    print(a,v_threshold)

        print('RESULT:', '\n          index',a,'\n          value', v_threshold)

        return a, v_threshold

    except Exception as e:
        print("Not possible to run, ",e)
```

C

HARDWARE SPECIFICATIONS

General	
Operating System	Linux Mint 19.2 Cinnamon
Processor	Intel Core i7 − 4702MQ @ 2.20GHz ×4
Memory	5.8 GB
Graphics	Intel Corporation 4th Gen Core Processor Integrated Graphics Controller
Processor details	
CPU	Intel Core i7 − 4702MQ
Topology	Quad Core
Min/Max core Speed	800/3200 MHz

Table 9: Hardware specifications of first device (A) used.

General	
Operating System	Windows 10 Pro Version 1903
Processor	Intel Core i7 − 4770K @ 3.50GHz
Memory	15.9 GB
Graphics	NVIDIA Geforce GTX 1080 Ti
Processor details	
CPU	Intel Core i7 − 4770K
Topology	Quad Core
Min/Max core Speed	800/3900 MHz

Table 10: Hardware specifications of second device (B) used.

This dissertation was supported by BIAL, in the context of a cooperation agreement with the University of Minho, and integrated within the KLEE research project - POCI-01-0145-FEDER-030947 (KLEE), supported by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme.