Universiteit Leiden
The Netherlands

Opleiding Informatica

Exploring the effectiveness of SMT solving for proving and disproving the existence of Mutually Unbiased Bases

Oliver ten Hoor

Supervisors:
Dr. Alfons Laarman & Arend-Jan Quist, MSc

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

August 7, 2025

Abstract

Mutually Unbiased Bases (MUB's) are fundamental in quantum information theory, with applications in quantum state tomography, cryptography, and error correction. While complete sets of MUB's are known for prime-power dimensions, the exact number of MUB's in non-prime-power dimensions, such as dimension six, remains an open problem. Zauner conjectured in 1999 that no more than three mutually unbiased bases exist in this dimension, yet a formal upper bound has not been proven. This thesis investigates whether Satisfiability Modulo Theories (SMT) solvers, specifically dReal4, can be used to algorithmically verify or refute this conjecture. We encode the mathematical conditions defining MUB's into an SMT acceptable format and explore both continuous and quantized formulations to reduce complexity. Our approach translates the problem into one of real-number constraint satisfaction, leveraging dReal's δ-complete decision procedures to perform bounded searches for additional MUB's. Through experimental analysis in lower dimensions (d = 2, 3, 4), we demonstrate that dReal can reconstruct known MUB's and identify when no further unbiased bases exist under quantized assumptions. However, the computational complexity of higher dimensions ($d \geq 4$) presents significant scalability challenges. While full formal verification remains limited by proof infrastructure and solver capabilities, this work lays the groundwork for using SMT-based reasoning in the automated study of quantum foundational problems.

Contents

1 Introduction

Quantum computing could revolutionize computing as we know it. Although we have yet to develop a meaningful quantum computer, its potential applications have already become evident. For example, cryptography and protein folding [1] are two problems that can be significantly improved with quantum computers, however, for now there are only simulations.
MUB's have useful applications in the foundations of quantum mechanics and certain quantum algorithms. They provide a way to represent quantum states so that measurements in one basis give no information about measurements in another, which is critical for secure communication protocols and quantum state tomography. Despite their importance, finding complete sets of MUB's in higher-dimensional quantum systems remains a challenging open problem.
This thesis focuses on exploring computational techniques to analyze and search for MUB's, particularly using Satisfiability Modulo Theories (SMT) solvers. Such tools offer a promising approach for encoding complex quantum constraints and leveraging computational power to gain insights that are difficult to obtain analytically. By advancing this line of research, we contribute to the deeper understanding necessary to unlock the potential of quantum technologies.

1.1 Current state of the art

We define $\mathcal{M}(d)$ as the number of orthonormal bases in dimension d inside a complex Hilbert space. We are interested in finding these bases because of their application in quantum information theory. These applications include quantum state reconstruction [2] and quantum error correction codes [3]. Prime-power dimensions admit complete sets of MUB's of size $d + 1$. When d is a prime or a prime-power ($d = p^m$), one can explicitly construct $d + 1$ mutually unbiased bases [4], we call this a full dimension. For example $d = 2$ has three MUB's, $d = 3$ has four MUB's and $d = 4$ has five MUB's. More generally, it is known that for every dimension at least three MUB's exist [4].
For $d = 2$ through $d = 5$, all the MUB's are known [5]. For non-prime-power dimensions, the maximal number of MUB's is unknown in general. This is also the case for $d = 6$. All known methods failed to prove the upper bound for $d = 6$ and strong evidence indicates $\mathcal{M}(6) = 3$. Zauner's conjecture asserts that $\mathcal{M}(6) = 3$ [6], although this has not yet been proven. It has been proven that $\mathcal{M}(6) \geq 3$ [7]. The conjecture can therefore be proven by showing $\mathcal{M}(6) < 4$. Our research question is therefore: *How can SMT-solvers be used to prove or disprove the conjecture of Zauner?*

1.2 Related Work

A lot of research has been done on MUB's because of their importance in quantum computing [5][7]. As mentioned before, all bases have been found for dimensions two through five [5]. Therefore, most of the research being done on MUB's revolves around dimension six. A big part of this research includes proving how many bases exist in the sixth dimension. This is done in different ways such as in the research of Raynal, Lü and Englert [8] where they try to prove $\mathcal{M}(6) < 4$ by the average distance between the MUB's. Again, just like Zauner they could only prove that it is likely and no definitive proof was found.
There are also multiple researches which try to construct multiple MUB's using Hadamard matrices [7][9]. Again, no fourth basis was found in these studies so the question still stands whether or not

$\mathcal{M}(6) < 4$ holds.
Mortimer has reduced the problem to a polynominal optimization problem over the reals [10]. He then used two different numeric methods to confirm known MUB's in small dimensions. The algorithms used are Lagrange-multiplier Newton search and semidefinite-programming branch-and-bound algorithm. Larger dimensions were infeasible to compute.

1.3 Contributions

In this thesis we will investigate the use of dReal's SMT solvers for finding MUB's. We will try to verify known bases in dimensions 2 through 6, as well as proving the maximum existing MUB's for certain dimensions. Due to the lack of compatible proof checkers, formal verification of solver output is not done in this thesis. We will try two different encoding's of the MUB problem and compare the effectiveness of both. We will also compare how different precisions of the SMT solver influence it's efficiency and effectiveness.
Ultimately, we will try to prove or disprove the conjecture of Zauner. We did not expect to prove or disprove this due to the complexity of the problem and the lack of a compatible proof checker. We do however expect to provide new insights on using dReal's SMT solvers for finding MUB's.
All implementations will be publicly available on GitHub [11].

1.4 Overview

This thesis is set up as follows.

- Section 2 will introduce important theoretical background on mutually unbiased bases such as the properties of MUB's. It will also include an overview of all the currently known bases for the dimensions we are interested in. It will also introduce satisfiability modulo theories solvers and which one we will be using in this thesis.

- Section 3 will contain the analytic encoding of the MUB problem, so that it will be compatible with the SMT solver. We will also discuss the quantized version of the MUB encoding, which we will heavily use due to its reduced complexity. We will also discuss the general complexity of the MUB.

- Section 4 will discuss the implementation of the problem. How we will be using SMT solvers for the problem. Including the problem with constant variables and which precision to use for the SMT solver. Here we will also calculate the expected outcomes of the SMT solver based on the encoding we will discuss in section 3 and how we will create the scripts used to calculate MUB's.

- Section 5 contains the results of the calculations performed in this thesis. For dimensions two through five, we will try to find all known existing bases. For dimensions two and three, we will also prove the maximum number of bases in their respective dimensions.

- Section 6 will discuss the results gathered in section 5. We will also discuss the limitations encountered in this thesis as well as points for further research.

2 Background

In this section, we will discuss the theoretical background of MUB's and all known bases for certain dimensions. We will also discuss the background of the two different SMT solvers considered and the proof files they create. We will also talk about the verification of such proof files.

2.1 Mutually unbiased bases

2.1.1 Theoretical background MUB's

A Hilbert space is a mathematical structure that generalizes the notion of Euclidean space to potentially infinite dimensions, while preserving key geometric concepts such as distance and angles. It is a complete inner product space, meaning that it has an inner product (allowing you to define lengths and angles) and is complete with respect to the metric induced by that inner product (so limits of convergent sequences stay within the space).
In quantum mechanics, the state of a system is typically represented as a vector in a complex Hilbert space, often denoted $\mathbb{C}^d$ for a finite-dimensional system [12].

A basis of vectors $M_0 = \{|\psi^0\rangle, |\psi^1\rangle, ..., |\psi^d\rangle\}$ is called normal when for all $i \in \{0, ..., d\}$ the vectors have unit length:

$$\||\psi^i\rangle\| = \sqrt{|\psi^i\rangle \cdot |\psi^i\rangle} = \sqrt{\langle\psi^i|\psi^i\rangle} = 1. \tag{1}$$

When this holds, we say that the basis has been normalized. This essentially means that the lengths of all vectors in M_0 have length 1.
The same basis M_0 is orthogonal when they are at right angles to each other:

$$|\psi^i\rangle \cdot |\psi^j\rangle = \langle\psi^i|\psi^j\rangle = 0 \text{ when } i \neq j \tag{2}$$

When both these assumptions hold, we say that the basis is orthonormal.
A pair of orthonormal bases $M_0 = \{|\psi_0^0\rangle, |\psi_0^1\rangle, ..., |\psi_0^{d-1}\rangle\}$ and $M_1 = \{|\psi_1^0\rangle, |\psi_1^1\rangle, ..., |\psi_1^{d-1}\rangle\}$ is called mutually unbiased if:

$$|\langle\psi_0^j|\psi_1^k\rangle|^2 = \frac{1}{d} \text{ for all } j, k \in \{0, 1, .., d-1\} \tag{3}$$

This means that any vector of one basis has equal overlap with all the vectors of the other. Therefore, knowing a state's outcome in one basis yields no information about outcomes in the other. If a measurement in basis M_0 yields a deterministic result, then a measurement in basis M_1 will give a uniformly random outcome [13].

2.1.2 Known bases

In this section, we will discuss the already known bases for different dimensions. We will discuss dimensions two, three, four and six, for these will be relevant in this thesis. For all the following dimensions, we will assume the standard basis $M_0 = \{|0\rangle, ..., |d-1\rangle\}$ and $\omega = e^{2\pi i/3}$. The standard basis can be assumed without loss of generality because any orthonormal basis can be transformed into the standard basis by a unitary change of basis, which preserves mutual unbiasedness.

Dimension 2

Dimension two is a full dimension, meaning $\mathcal{M}(2) = d + 1$. Using this, we know there exist two bases on top of the standard basis. They are the following [5]

$$M_1 = \left\{ \frac{|0\rangle+|1\rangle}{\sqrt{2}}, \frac{|0\rangle-i|1\rangle}{\sqrt{2}} \right\}$$
$$M_2 = \left\{ \frac{|0\rangle+i|1\rangle}{\sqrt{2}}, \frac{|0\rangle-|1\rangle}{\sqrt{2}} \right\}.$$

Dimension 3

Dimension three is also a full dimension, therefore, there are 3 bases on top of the standard basis. They are the following [5]

$$M_1 = \left\{ \tfrac{1}{\sqrt{3}}(1,1,1), \tfrac{1}{\sqrt{3}}(1,\omega,\omega^2), \tfrac{1}{\sqrt{3}}(1,\omega^2,\omega) \right\}$$
$$M_2 = \left\{ \tfrac{1}{\sqrt{3}}(1,\omega,\omega), \tfrac{1}{\sqrt{3}}(1,\omega^2,1), \tfrac{1}{\sqrt{3}}(1,1,\omega^2) \right\}$$
$$M_3 = \left\{ \tfrac{1}{\sqrt{3}}(1,\omega^2,\omega^2), \tfrac{1}{\sqrt{3}}(1,1,\omega), \tfrac{1}{\sqrt{3}}(1,\omega,1) \right\}.$$

Dimension 4

Again, dimension four is a full dimension, therefore there exist 4 additional bases on top of the standard basis. They are the following [5]

$$M_1 = \left\{ \tfrac{1}{2}(1,1,1,1), \tfrac{1}{2}(1,1,-1,-1), \tfrac{1}{2}(1,-1,-1,1), \tfrac{1}{2}(1,-1,1,-1) \right\}$$
$$M_2 = \left\{ \tfrac{1}{2}(1,-1,-i,-i), \tfrac{1}{2}(1,-1,i,i), \tfrac{1}{2}(1,1,i,-i), \tfrac{1}{2}(1,1,-i,i) \right\}$$
$$M_3 = \left\{ \tfrac{1}{2}(1,-i,-i,-1), \tfrac{1}{2}(1,-i,i,1), \tfrac{1}{2}(1,i,i,-1), \tfrac{1}{2}(1,i,-i,1) \right\}$$
$$M_4 = \left\{ \tfrac{1}{2}(1,-i,-1,-i), \tfrac{1}{2}(1,-i,1,i), \tfrac{1}{2}(1,i,-1,i), \tfrac{1}{2}(1,i,1,-i) \right\}.$$

Dimension 6

We know $\mathcal{M}(6) \geq 3$. Concluding from this we know there exist at least two additional bases on top of the standard basis. They are the following [7][8].

$$M_1 = \left\{ \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ \omega \\ \omega^2 \\ 1 \\ \omega \\ \omega^2 \end{pmatrix}, \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ \omega^2 \\ \omega \\ 1 \\ \omega^2 \\ \omega \end{pmatrix}, \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{pmatrix}, \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ \omega \\ \omega^2 \\ -1 \\ -\omega \\ -\omega^2 \end{pmatrix}, \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ \omega^2 \\ \omega \\ -1 \\ -\omega^2 \\ -\omega \end{pmatrix} \right\}$$

$$M_2 = \left\{ \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ 1 \\ \omega^2 \\ i \\ i\omega^2 \\ i \end{pmatrix}, \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ \omega \\ \omega \\ i\omega \\ i\omega \\ i \end{pmatrix}, \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ \omega^2 \\ 1 \\ i\omega^2 \\ i \\ i \end{pmatrix}, \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ 1 \\ \omega^2 \\ -i \\ -i\omega^2 \\ -i \end{pmatrix}, \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ \omega \\ \omega \\ -i\omega \\ -i\omega \\ -i \end{pmatrix}, \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ \omega^2 \\ 1 \\ -i\omega^2 \\ -i \\ -i \end{pmatrix} \right\}.$$

2.2 Satisfiability modulo theories

Satisfiability modulo theories (SMT) solvers extend classical Boolean satisfiability (SAT) solvers by supporting rich mathematical theories, such as arithmetic, trigonometry, and differential equations, while retaining SAT's core reasoning framework. Where SAT solvers operate on purely discrete logic, SMT solvers can handle more complex formulas, like math equations involving real numbers, integers, or other advanced constraints. This makes them indispensable for problems like the MUB problem, where trigonometric functions and exact real arithmetic are central. In this thesis, we focus on dReal4, a δ-complete SMT solver tailored for nonlinear real arithmetic.
SMT solvers take a mathematical formula as input and search for a satisfiable assignment of the variables. Take, for example, the following formula:

$$x^2 + y^2 = 10 \tag{4}$$

where $0 \leq x \leq 10$ and $0 \leq y \leq 10$. The SMT solver will try to find values such that the formula will be correct. In this example we see there are multiple assignments for the variables for the formula to be satisfied, for example $x = 1$ and $y = 3$, because $1^2 + 3^2 = 10$.
SMT solvers work on the notion of proof via exhaustion. When the formula is satisfiable the SMT solver will return a model containing the assignments for the variables. When on the other hand the formula turns out not to be satisfiable, the SMT solver will return a proof file that proves the formula is unsatisfiable [14] by showing that there are no assignments that work for the formula. The solver will then return the value `unsat`.
In the example discussed above, if we change the problem to $x^2 + y^2 = 201$, there are no satisfying assignments for the variables x and y. This is because the maximum possible sum of these variables would be $10^2 + 10^2 = 200$, given the constraints. In this case the solver would return `unsat` and could provide a proof file.

2.2.1 dReal4

dReal4 is an automated reasoning tool. It is specifically strong for problems that can be encoded in first-order logic problems over the real numbers. It implements the framework of δ-complete descision procedures [15] [16]. This means the solver operates with a specified precision, denoted by δ, when searching for a satisfying assignment, since without it there would be infinitely many real numbers to consider within any interval. When a satisfying assignment is found, dReal4 will return δ-sat. We can then also ask for it to produce a model. This will essentially show the range in which the different variables can lie for the system to be δ-sat (within the range δ).
dReal supports various kinds of logic for their solver. The primary focus of dReal4 is solving satisfiability problems involving nonlinear real arithmetic constraints, for example, trigonometric functions such as sin and cos. This is supported via the QR_NRA (Quantifier-Free Nonlinear Real Arithmetic) logic flag.
In this thesis, we will be working with QF_NRA logic because of the nature of the MUB problem which works with trigonometric functions, as we will see in section 3.1.

2.2.2 Z3

While dReal4 is the primary solver used in this thesis due to its support for δ-complete reasoning over the reals, it is worth briefly discussing an alternative: Z3 [17].

This is an SMT solver developed by Microsoft Research. This solver uses SMT-LIB instead of the SMT-LIBv2 that dReal4 uses, which is the main difference influencing our choice of dReal4 over Z3. The SMT-LIB version, just as Z3, does not support trigonometric functions as standard. In this thesis, we will use sin and cos functions heavily. We can approximate these values with Z3 but this brings other challenges with it.

- How precise do you have to approximate?
- How does this approximation influence the complexity and thus solving time?
- How can rounding errors or numerical instability influence solver correctness?
- How do you validate that the approximate model correctly captures the behavior of the original system within acceptable tolerances?

These challenges highlight a key limitation of using Z3 for the MUB problem. Since Z3 expects exact arithmetic and lacks δ-bounded reasoning, it is not well suited for problems where small numerical tolerances and nonlinear trigonometric constraints play a central role. In contrast, dReal4's ability to reason with δ-approximations makes it a better fit for this context. Although Z3 remains a powerful general-purpose solver, it is less practical in domains where exact satisfiability is too restrictive or computationally infeasible due to the nature of the mathematical objects involved.

2.2.3 Formal verification

The version of dReal that we are going to use lacks a key feature in proof verification. The developers have not yet implemented a proof function. When a certain problem turns out to be `unsat` we have no way of creating a proof. Fortunately, an older version dReal3, has this function. We can use this version on the problems we know to be `unsat` to create a proof. We can then use this proof in combination with a certified checker to verify our findings. This is an essential step in automated proofing for several reasons.

- **Solver correctness:** Like any complex software system, SMT solvers are susceptible to bugs and implementation errors. A solver may incorrectly declare a formula `unsat` due to subtle flaws in its search strategy, numerical precision, or constraint propagation logic. Certified checkers help mitigate this risk by independently verifying whether the proof of unsatisfiability is logically valid.
- **Cross-tool validation:** Different solvers may implement different algorithms and heuristics, and may even interpret certain edge cases differently. By using a certified checker, ideally one decoupled from the solver, we ensure that our conclusions are not solver-specific. This helps build confidence that the result is not an artifact of a particular implementation but a sound mathematical conclusion.
- **Reproducibility and auditability:** Proof objects can be stored and shared, allowing others to independently verify the result without relying on the same solver or environment. This is particularly important in scientific applications, safety-critical domains, and formal mathematics, where reproducibility and traceability are essential.

3 Formalizing MUB Constraints for SMT Solving

3.1 Analytic encoding of MUB constraints

The encoding described in this section is based on the idea of my supervisor, Arend-Jan Quist [18]. To enable the SMT solver to search for the MUB's, we first have to encode the properties into a form that the SMT solver can work with. As mentioned before in the background section 2.1.1, for a basis of vectors to be mutually unbiased in $\mathbb{C}^d$ Hilbert space, it has to abide to two properties. The first being orthonormal (1)(2) and the second being mutually unbiased (3).
We assume we want to check whether $\mathcal{M}(d) \geq l + 1$. In this case we can always pick as the first basis $M_0 = \{|0\rangle, |1\rangle, \ldots, |d-1\rangle\} = \{(1, 0, \ldots, 0), (0, 1, 0, \ldots, 0), \ldots, (0, \ldots, 0, 1)\}$, because this basis always holds up, as seen in section 2.1.2. We are then going to look for the remaining bases $M_1, ..., M_l$.
If we pick M_0 as shown above then the bases M_1 to M_l take the form $M_m = \{|\psi_m^0\rangle, \ldots, |\psi_m^{k-1}\rangle\}$, consisting of complex vectors. These complex vectors consist of d phases which represent the angle of that component in the d-dimensional complex plane. So a vector in $d = 3$ has three phase components. This can formally be written as

$$|\psi_m^j\rangle = \frac{1}{\sqrt{d}}(a_m^j(1), a_m^j(2), \ldots, a_m^j(d)) \tag{5}$$

where every component $a_m^j(z)$ (phase component) is a value denoted in the form $e^{i\pi f(j,m,z)}$, where $f(j, m, z)$ is a real value where j is the vector inside a basis, $j \in \{0, \ldots, d-1\}$, m is the basis in which this vector resides, $m \in \{1, \ldots, \ell\}$ and z is the entry index of the vector, $z \in \{1, \ldots, d\}$. In this study we are interested in finding this $f(j, m, z)$ value because we can reconstruct the vectors with the translation mentioned above.

As mentioned above we have to constrain the bases to the properties of MUB's (1)(2)(3).

- M_m is an orthonormal basis for $m \in \{1, ..., l\}$. For a basis to be orthonormal the inner product of all the phases between two vectors in the same basis has to be zero. Formally this means we have $\langle \psi_m^j \mid \psi_m^k \rangle = 0$ for all m if $j \neq k$. The inner product of two vectors is defined as the sum of the product of two corresponding phases. This can be rewritten as

$$\sum_{z=1}^{d} (a_m^j(z))^* a_m^k(z) = 0. \tag{6}$$

- M_m and $M_{m'}$ are mutually unbiased for $m, m' \in \{1, ..., l\}$ with $m \neq m'$. For two bases to be mutually unbiased the square of two vectors from different bases should equal $\frac{1}{d}$, formally written as $|\langle \psi_m^j \mid \psi_{m'}^k \rangle|^2 = \frac{1}{d}$ for all m, m', j, k with $m \neq m'$. This can be rewritten as

$$\left| \sum_{z=1}^{d} (a_m^j(z))^* a_{m'}^k(z) \right|^2 = d \tag{7}$$

  for all $m, m' \in \{1, \ldots, \ell\}$ such that $m \neq m'$ and for all $j, k \in \{0, \ldots, d-1\}$.

Working out the equations from above which formalize mutually unbiasedness and orthonormality we can define the following SMT problem to check whether $\mathcal{M}(d) \geq l+1$. We have to eliminate the imaginary part of the phases and just look for the $f(j,m,z) \in [0,2)$ values of the phases for $j \in \{0,...,d-1\}, m \in \{1,...,l\}, z \in \{1,...,d\}$.
First the mutually unbiased property (7), for every $m, m' \in \{1,...,l\}$ such that $m \neq m'$ and for all $j,k \in \{0,...,d-1\}$ we have the formula above (7). We defined

$$a_m^j(z) = e^{i\pi f(j,m,z)} \tag{8}$$

The complex conjugate would then be

$$(a_m^j(z))^* = e^{-i\pi f(j,m,z)} \tag{9}$$

Substitute in the earlier formula (7) gives

$$\left| \sum_{z=1}^{d} e^{-i\pi f(j,m,z)} \cdot e^{i\pi f(j,m',z)} \right|^2 = \left| \sum_{z=1}^{d} e^{i\pi (f(j,m',z) - f(j,m,z))} \right|^2 \tag{10}$$

We then use Euler's formula, $e^{i\theta} = \cos(\theta) + i\sin(\theta)$, and the modulo squared of complex numbers, $|(x+iy)|^2 = x^2 + y^2$

$$\left| \sum_{z=1}^{d} e^{-i\pi f(j,m,z)} \cdot e^{i\pi f(j,m',z)} \right|^2 =$$
$$\left| \sum_{z=1}^{d} \cos\left(\pi (f(k,m',z) - f(j,m,z))\right) \right|^2 + \left| \sum_{z=1}^{d} \sin\left(\pi (f(k,m',z) - f(j,m,z))\right) \right|^2 \tag{11}$$

So the final encoding of the mutually unbiased property will look like so

$$\left| \sum_{z=1}^{d} \cos(\pi (f(k,m',z) - f(j,m,z))) \right|^2 + \left| \sum_{z=1}^{d} \sin(\pi (f(k,m',z) - f(j,m,z))) \right|^2 = d \tag{12}$$

For the orthonormal constraint (6) mentioned above we again substitute the phase values with their complex values (8) (9)

$$\sum_{z=1}^{d} e^{-i\pi f(j,m,z)} \cdot e^{i\pi f(k,m,z)} = \sum_{z=1}^{d} e^{i\pi (f(k,m,z) - f(j,m,z)} \tag{13}$$

We then again use Euler's formula

$$\sum_{z=1}^{d} \cos(\pi (f(k,m,z) - f(j,m,z))) + i \sum_{z=1}^{d} \sin(\pi (f(k,m,z) - f(j,m,z))) = 0 \tag{14}$$

Because the above equation is a complex number equal to 0, both its real and imaginary part must be 0, $X + iY = 0 \implies X = 0$ and $Y = 0$. From that we can deduce the following two functions for the orthonormality constraint, for all $j,k \in \{0,...,d-1\}$ where $j \neq k$ and for all $m \in \{1,...,l\}$

$$\sum_{z=1}^{d} \cos(\pi (f(k,m,z) - f(j,m,z))) = 0 \text{ and } \sum_{z=1}^{d} \sin(\pi (f(k,m,z) - f(j,m,z))) = 0 \tag{15}$$

So following from this, we can use the formulas (12) and (15) for a SMT solver.

3.2 Quantize

Due to the rapid growth of the constraints in our problem, one of my supervisors came up with the idea of quantizing the problem [18]. In the current above discussed version of encoding, the solver will have to show that all the real values of the range of the variables are unsatisfiable. Even with small precision values, there are a lot of values to check. The idea is instead of searching for $f(j,m,z) \in [0,2)$, we will look for $f(j,m,z) \in \left\{ \frac{x}{q} | x \in \{0,1,...,2 \cdot q - 1\} \right\}$ with $q = d \cdot 2^n$. We can use the value of n to decide how many values we want to look for. If we pick a low value for n, we will look over less values for the problem. It will suffice to take $q = d$ for $d = 1,2,3,4$ to look for $\mathcal{M}(d) = l + 1$[6].
If we look at $d = 2$ we can see that we are going to look for

$$f(j,m,z) \in \left\{ \frac{0}{2}, \frac{1}{2}, \frac{2}{2}, \frac{3}{2} \right\}. \tag{16}$$

Comparing these to the values seen before in section 3.3, we can see that all the values are indeed one of these values.
This method will drastically decrease complexity for our problem, however we can not formally prove the question of $\mathcal{M}(6) < 4$ with it. There could exists a basis in between these values which we are not looking for. If there were to be a basis within these values for $d = 6$ we would be able to find it. So we can disprove the question of $\mathcal{M}(6) < 4$ but never prove it this way.

3.3 MUB check

When we have found a basis we can check with the formulas mentioned earlier that they are indeed mutually unbiased. We can show this by trying to input the values of the bases of an already known dimension, for example $d = 2$. For this dimension it is proven that there are 3 bases including the standard basis. For this dimension we take standard basis. The other two bases are as discussed in section 2.1.2. We can rewrite for example the first vector of the first basis into the vector form we discussed earlier (5)

$$|\psi_1^1\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \tfrac{1}{\sqrt{2}}(1,1).$$

We can now clearly see what phase values this vector has, $a_1^1(1) = 1$ and $a_1^1(2) = 1$. With these phase values we can calculate what their $f(j,m,z)$ values should be using the definition mentioned earlier (8)

$$a_1^1(1) = 1 = e^{i\pi f(j,m,z)}.$$

Here we can see that the corresponding value with $a_1^1(1) = 1$ should be $f(j,m,z) = 0$. Using this method to find all the other f-values from these two bases, we get the following values

$$\begin{gathered} f(1,1,1) = 0,\ f(2,1,1) = 0 \\ f(1,1,2) = 0,\ f(2,1,2) = 1 \\ f(1,2,1) = 0,\ f(2,2,1) = 0 \\ f(1,2,2) = 0.5,\ f(2,2,2) = 1.5. \end{gathered}$$

We can check whether or not the bases are mutually unbiased by running the f-values through the MUB constraints mentioned earlier (12)(15). The orthonormal constraints would look like so

$$\begin{gathered}\cos(\pi(0-0))+\cos(\pi(0-1))=0\\ \sin(\pi(0-0))+\sin(\pi(0-1))=0\\ \cos(\pi(0-0))+\cos(\pi(0.5-1.5))=0\\ \sin(\pi(0-0))+\sin(\pi(0.5-1.5))=0.\end{gathered}$$

These values all hold up. Now to look at the mutually unbiasedness constraints

$$\begin{gathered}|\cos(\pi(0-0))+\cos(\pi(0-1))|^2+|\sin(\pi(0-0))+\sin(\pi(0-1))|^2=2\\ |\cos(\pi(0-0))+\cos(\pi(0-1.5))|^2+|\sin(\pi(0-0))+\sin(\pi(0-1.5))|^2=2\\ |\cos(\pi(0-0))+\cos(\pi(1-1.5))|^2+|\sin(\pi(0-0))+\sin(\pi(1-1.5))|^2=2\\ |\cos(\pi(0-0))+\cos(\pi(1-1.5))|^2+|\sin(\pi(0-0))+\sin(\pi(1-1.5))|^2=2.\end{gathered}$$

Also these values hold up to the constraints set earlier, so these are indeed mutually unbiased bases.

3.4 Complexity

A big part of the problem is the complexity of the MUB's. When the dimension gets higher the problem rapidly gets very complex. We analyze the amount of variables and constraints we have to calculate with dimension d and amount of bases l.
First we have to calculate how many phase values ($f(j,m,z)$) the solver has to find. Because every basis has d vectors and every vector has d phase values as shown in (5), we get $d \cdot d$ phase values per basis. So over all the bases we have the following amount of phase values

$$l(d \cdot d) = l \cdot d^2.$$

Over all the bases we have to calculate how many orthonormality (15) constraints we have to calculate. All vectors inside of a basis have to be orthonormal to all the other vectors inside the basis, so there are $\frac{d(d-1)}{2} \cdot 2 = (d^2 - d)$ constraints per basis so in total

$$l(d^2 - d).$$

Then for the MUB constraints each pair of distinct bases must be mutually unbiased. For each pair of bases you need d^2 constraints (one per inner product of each pair of vectors from the two bases). This looks like the following

$$\tfrac{l(l-1)}{2} \cdot d^2.$$

The number of constraints increases rapidly as the dimension and the number of bases grow. We can visualize this in the following graphs. In the following graphs the full bases have been used in the form of $\mathcal{M}(d) = l + 1$, so this means $d = l$.

The growing amount of constraints and phases is not the only reason for the growing complexity. The constraints themselves also grow in terms of variables. The orthonormal constraints (15) are based on the phases of two vector of the same basis. For every two vectors they have to sum all the phases. In the example showed in section 3.3 the orthonormal constraints consist of two parts, because of the two phases. If we take for example a vector in dimension three, every constraint will exist of three parts. The same is true for the mutually unbiased constraints, only double because they consist of two sums over the phases.

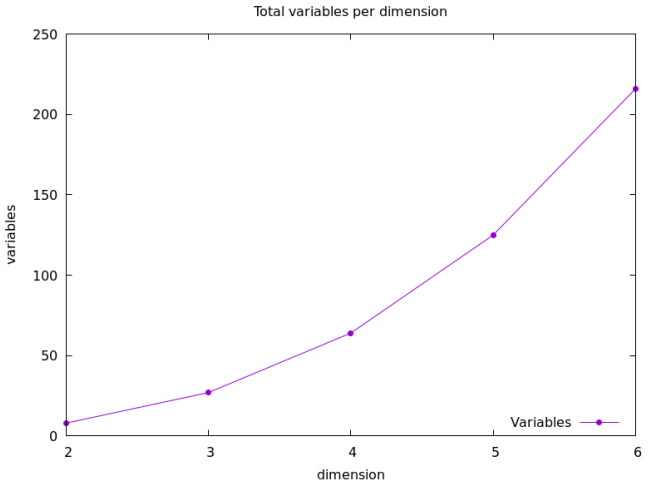

Figure 1: Variables per dimension

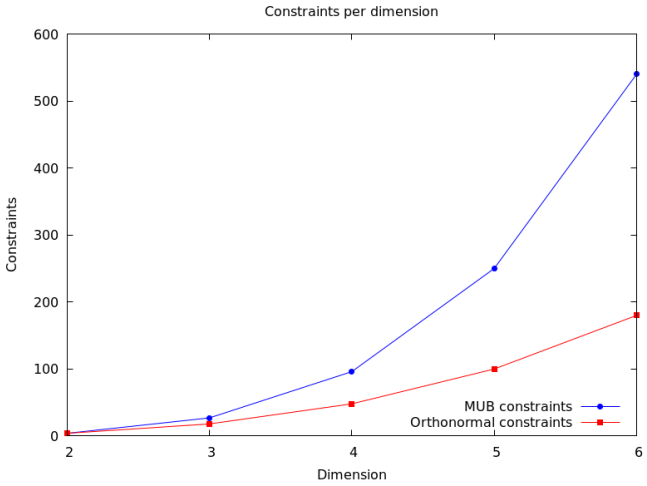

Figure 2: Constraints per dimension

4 Implementation

As mentioned above, we will be using an SMT solver in this thesis. The one we will be using is dReal4 [19]. This solver uses the SMT-LIBv2 [14] format to encode problems. We will translate the MUB problem into mathematical formulas such that the SMT-LIBv2 format will suffice for the encoding.

4.1 SMT-LIB v2

The SMT solver we used in this research, dReal4, uses the SMT-LIBv2 [14] format for input. SMT-LIBv2 is an SMT language supported by multiple research groups world-wide to create a general language/interface for SMT solvers.
The format uses prefix notation also known as Polish notation, this means the operator comes first followed by its operands. For example, x = 5 is written as = x 5.
If we take the formula used earlier, $x^2 + y^2 = 10$, and encode it in the SMT-LIBv2 format. We first have to start by defining which logic we will be using, for the example this will be QF_NRA. This means our problem is quantifier-free and uses Nonlinear Real Arithmetic. We set this option as the first line of our file like so:

(set-logic QF_NRA).

Next we have to set the precision with which the SMT solver will work. Because we are using real values we have to set a precision on which the SMT solver will work. We set the solver precision to 0.001, as this provides a practical balance between computational efficiency and accuracy for our problem. We set this option like so:

(set-option :precision 0.001).

Then we can start with the actual program. The first step is to declare all the variables that we will be using. Since version 2 of the SMT-LIB format there is no distinction anymore between

functions and variables [14], and all will be declared as functions. We will declare our variables as real numbers like so:

(declare-fun x () Real)
(declare-fun y () Real).

The next step is to declare the bounds of the variables we are going to use, as mentioned before using prefix notation:

(assert (and (>= x 0.0) (< x 10.0)))
(assert (and (>= y 0.0) (< x 10.0))).

The final line of real code that we need is the actual formula of which the SMT solver will have to find the values.

(assert(= (+ (^ x 2) (^ y 2)) 10)).

And finally you have to add a line that begins the SMT solver.

(check-sat)

This is the general format of SMT-LIBv2 files. When run with the dReal4 solver this will be the output:

delta-sat with delta = 0.001
x : [1.054028216685762, 1.054221226796856]
y : [2.9813784739564562, 2.9814467156785205].

For each variable x and y dReal returns an interval guaranteed to contain a true solution. The two values represent the lower and upper bound. If we take for example the midpoints of both these ranges we get $x = 1.0541$ and $y = 2.9814$. If we sum the squares of both these values we get, $1.0541^2 + 2.9814^2 = 1.11112681 + 8.88874596 \approx 9.9998$. This values lies between 10 ± 0.001 so δ-sat.

4.2 Constant variables

Known δ-sat problems resulted in an `unsat`. This is because dReal4 uses δ-semantics in combination with how it handles floating point rounding and how it approximates trigonometric functions. When we assert for example the following function for orthonormality where $f(j, m, z) = fjm\_z$

(assert (= 0.0 (+ (cos (* pi (- f10_0 f11_0))) (cos (* pi (- f10_1 f11_1))))))

dReal4 expects this to be exactly 0.0, which is incompatible with the workings of the SMT solver. It will more likely be some value very close to zero, like -0.001 or 0.001. A workaround for this is to declare a variable with a very tight range around zero as follows

(declare-fun VariableZero () Real)
(assert (and (> VariableZero -0.0001) (< VariableZero 0.0001))).

Instead of asserting the function to be 0.0 we can assert it to be the variable VariableZero which has some margin of error.
This solution works for a single constraint but when we start to combine multiple constraints and we use for every constraint the same VariableZero variable we again get `unsat`. Because we assert all the different constraints to be the same zero variable it expects to get the same result from every constraint. Now again we have the same problem that not every constraint results the exact same value close to zero, so the solver returns `unsat`. A way to combat this problem is to make a new variable for every constraint that is made. A major drawback of this approach is that it has a big impact on the complexity and the time it takes to solve.
In our encoding for the orthonormal constraints it falls apart into 2 separate constraints (15), one using sin and the other using cos. During our testing we noticed that all the values of the variables used for the cos functions looked alike, and also so for the sin functions. We assumed that sin and cos get interpreted differently. So we tried to make a single outcome variable for all the sin functions and another for the cos, $cosZero$ and $sinZero$. This worked and so we could reduce the amount of one extra variable per constraint, which would come to $l(d^2 - d)$ extra variables, to 2 extra variables in total. We had the same problem with the mutual unbiased constraints (12), but that was combated with a single variable $dVar$ with $dVar \in [d - \delta, d + \delta]$, because of the nature of the mutually unbiased constraints being a sum of a sin component and a cos component.

4.3 Precision

The precision parameter δ plays a big role in dReal's solvers, because unlike other SMT solvers which aim for exact symbolic answers, dReal uses δ-sat for nonlinear real arithmetic. The precision parameter directly controls the numerical error in the solution. The smaller the precision gets, the more accurate the answer will be but also the more complex the calculation will get. Our setup was not capable of computing the continuous problem for most of the dimensions so we mostly used the quantized version of the problem, where the precision played a smaller role. For the calculations the solver has completed, we varied the precision from 0.1 to 0.000001.

4.4 Proof files

When running the dReal4 SMT solver on problems, they either return δ-sat or `unsat`. When it returns δ-sat it has found an assignment for all the variables within the set precision. When this is the case, it will return a model with the found ranges for the problem. The difficult case is to prove that the SMT solver worked when it returned `unsat`. Assuming the solver's correctness, this result disproves the existence of a model satisfying the MUB problem. These proofs grow rapidly in size as the scale of the problem increase. As mentioned before the dReal4 SMT solver lacks the functionality of creating proof files on an `unsat` result. With the version dReal3 we can produce proof files for the MUB problem. dReal3 produces human readable proof files and also checker readable proof files. Sadly, the build-in proof checker of dReal is not yet supported. Therefore we can not check our proofs with dReal. While we can manually verify the correctness of proofs by inspecting dReal's human-readable proof files for small instances, this approach becomes infeasible for larger problems due to the rapid growth in proof size. Because of the nature of the dReal3/4 solvers using δ-sat instead of regular SMT, there is no third party verification software we can apply on our problem. We can use dReal3/4 to prove δ-sat and create human readable proofs, but

not formally verify the proofs.

4.5 Calculating f-values

As seen in the encoding of the MUB problem in section 3.1 this thesis is centered around finding the f-values of the mutually unbiased bases. We see in section 2.1.2 the entries of different vectors of bases. Because we will forward certain bases to the SMT solver and we want to interpret the results of the SMT solver, we first have to see which phase entries correspond with which f-values. The different phase entries we see in the known bases are: 1, -1, ω, $-\omega$ ω^2, $-\omega^2$, i, $-i$, $i\omega$, $-i\omega$, $i\omega^2$, $-i\omega^2$ where $\omega = e^{2\pi i/3}$. Using the the form of the entries as discussed in (8) we can deduct the following f-values:

- $\mathbf{1} : e^{i\pi \cdot 0} = e^0 = 1 \rightarrow f = 0$
- $\mathbf{-1} : e^{i\pi \cdot 1} = e^{i\pi} = -1 \rightarrow f = 1$
- $\omega : e^{\pi i \cdot (2/3)} = e^{2\pi i/3} = \omega \rightarrow f = \frac{2}{3}$
- $-\omega : e^{\pi i \cdot (5/3)} = e^{5\pi i/3} = -\omega \rightarrow f = \frac{5}{3}$
- $\omega^{\mathbf{2}} : e^{\pi i \cdot (4/3)} = e^{4\pi i/3} = \omega^2 \rightarrow f = \frac{4}{3}$
- $-\omega^{\mathbf{2}} : e^{\pi i \cdot (1/3)} = e^{\pi i/3} = -\omega^2 \rightarrow f = \frac{1}{3}$
- $\mathbf{i} : e^{i\pi \cdot \left(\frac{1}{2}\right)} = e^{i\pi/2} = i \rightarrow f = \frac{1}{2}$
- $-\mathbf{i} : e^{i\pi \cdot (-\frac{1}{2})} = e^{-i\pi/2} = -i \rightarrow f = -\frac{1}{2}$. This can be rewritten as $f = \frac{3}{2}$ because of mod 2.
- $\mathbf{i}\omega$: We already know ω and i , so $i \cdot \omega = e^{i\pi/2} \cdot e^{i\pi \cdot (2/3)} = e^{i\pi\left(\frac{2}{3}+\frac{1}{2}\right)} = e^{i\pi \cdot \frac{7}{6}} \rightarrow f = \frac{7}{6}$.
- $-\mathbf{i}\omega$: We already know ω and $-i$, so $(-i) \cdot \omega = e^{-i\pi/2} \cdot e^{i\pi \cdot (2/3)} = e^{i\pi((-1/2)+(2/3)} = e^{i\pi(1/6)} \rightarrow f = \frac{1}{6}$
- $\mathbf{i}\omega^{\mathbf{2}}$: We already know ω^2 and i, so $i \cdot \omega^2 = e^{i\pi/2} \cdot e^{i\pi \cdot (4/3)} = e^{i\pi((4/3)+(1/2))} = e^{i\pi \cdot (11/6)} \rightarrow f = \frac{11}{6}$
- $-\mathbf{i}\omega^{\mathbf{2}}$: We already know ω^2 and $-i$, so $(-i) \cdot \omega^2 = e^{-i\pi/2} \cdot e^{i\pi \cdot (4/3)} = e^{i\pi \cdot ((4/3)-(1/2))} = e^{i\pi \cdot (5/6)} \rightarrow f = \frac{5}{6}$

4.6 .smt2 file creation

Because of the many constraints and variables, it is not feasible to write all the scripts by hand. Therefore we made a python script that generates SMT-LIBv2 files for calculating MUB's. At the top of the script there are a few constants that need to be initialized based on the situation you are looking for. The variables d, l and v are pretty straight forward and denote the dimension, amount of bases and amount of vectors in each basis respectively. There is also a Boolean variable *quantized* which denotes if you want to quantize the problem or look for the phase values $f(j, m, z) \in [0, 2)$. When the quantized option is picked you also have to denote how quantized you want to make the problem. This can be done with the variable n. Note that like mentioned before in section 3.2, for some small values we will use the dimension for the quantized variable.

5 Results

In this section, we will give the results of the experiments that were conducted. We give a proof of concept for finding MUB's using SMT solvers by finding correct MUB's for the known dimension $d = 2$ through $d = 5$. We also look at runtimes of the experiments in which we tried finding all known MUB's for certain dimensions. Lastly, we also look at what influence precision has on the runtimes of the experiments. All the scripts written and proof files can be found in the GitHub repository [11].

5.1 Hardware

All the experiments are run on a Dell Inc. Latitude 7490 machine with 16.0 GiB of memory and the Intel® Core™ i5-8350U CPU @ 1.70GHz × 8.

5.2 $\mathcal{M}(2)$

To start of with the problem in the second dimension. Firstly, we tried to input all the known values, from section 3.3, and the MUB constraints (12)(15) to see if the solver and our encoding worked. We define all the variables and implement all the constraints in the correct form like in section 4.1. The solver resulted δ-sat with delta = 0.001.

5.2.1 Quantized version

In this section we are going to use the method described before (3.2), where we will discretize the values $f(j, m, z)$. For lower dimensions it was enough to use $q = d$. So we will assert all the phase variables to be one of the values calculated earlier in eq. (16). The resulting output is as follows

delta-sat with delta = 0.001
pi : [3.1415926535897927, 3.1415926535897931]
d : [2, 2]
l : [2, 2]
f10_0 : [0, 0]
f10_1 : [0, 0]
f11_0 : [0, 0]
f11_1 : [1, 1]
f20_0 : [0, 0]
f20_1 : [0.5, 0.5]
f21_0 : [0, 0]
f21_1 : [1.5, 1.5]
sinZero : [-5.6655388976479895e-16, -1.2246467991473554e-16]
cosZero : [-1.4432899320127035e-15, 1.4432899320127035e-15]
dVar : [1.9999999999999942, 2]

You can see that we indeed got the correct values for the phase values as calculated in section 3.3. We can see that also the ranges of $dVar$ and $sinZero$ & $cosZero$ lie sufficiently close to the expected values of 2 and 0, with this δ-precision.

5.2.2 $\mathcal{M}(2) < 4$

When we try to prove $\mathcal{M}(2) = 3$, we have to show $\mathcal{M}(2) < 4$. Because we know there exist three bases for the second dimension, showing $\mathcal{M}(2) < 4$ will prove $\mathcal{M}(2) = 3$. Using the quantized version, we get an `unsat` result. Because dReal4 does not provide a proof file at an `unsat` result we also have to run the script with the modified version for dReal3.

5.3 $\mathcal{M}(3)$

From the third dimension on we have to use quantized versions of the problem because of the solving times. When we try to solve $\mathcal{M}(3) < 4$ with $f(m, j, z) \in [0, 2)$ we get too large solving times. We have run the solver for 100+ hours and not get a result. We could not run the solver longer because of outside factors.
We are going to tackle the problem using the quantized version. The already known bases for the third dimension on top of the standard basis $\mathcal{M}_0$ are discussed in section 2.1.2.
When we run the solver on our quantized version of the problem, with $q = d$ because of the lower dimension we get the a δ-sat result.

5.3.1 $\mathcal{M}(3) < 5$

We can try to prove $\mathcal{M}(3) < 5$ the same way we tried proving $\mathcal{M}(2) < 4$. We create a file that looks for $\mathcal{M}(3) < l + 1$ with $l = 4$. This file will look for four additional bases upon the standard basis. If we want to create a proof file we have to use dReal3's proof function. When this script is run it returns the expected value of `unsat`. We used the quantized version of the encoding because of the calculation time.

5.4 $\mathcal{M}(4)$

In the fourth dimension we also get trouble with too large solve times. We will therefore first try to find a smaller amount of bases and then work our way up.
The bases in the fourth dimension are known, if we pick M_0 as the standard basis we can define the other four as mentioned in section 2.1.2. We will first try to look for two bases instead of the total of four. We will again use our standard smtCreator.py file to make a file with $d = 4$ and $l = 2$. We make use of quantized versions because we already know what values the bases can contain and to keep solving time low. When run we get δ-sat. When we look at the values the solver returns we see that it does not match precisely. These are the values the solver returns

f10_0 : [0, 0] - f10_1 : [0, 0] - f10_2 : [0, 0] - f10_3 : [0, 0]
f11_0 : [0, 0] - f11_1 : [0, 0] - f11_2 : [1, 1] - f11_3 : [1, 1]
f12_0 : [0, 0] - f12_1 : [1, 1] - f12_2 : [1.5, 1.5] - f12_3 : [0.5, 0.5]
f13_0 : [0, 0] - f13_1 : [1, 1] - f13_2 : [0.5, 0.5] - f13_3 : [1.5, 1.5]
f20_0 : [0, 0] - f20_1 : [0, 0] - f20_2 : [0, 0] - f20_3 : [1, 1]
f21_0 : [0, 0] - f21_1 : [1, 1] - f21_2 : [0, 0] - f21_3 : [0, 0]
f22_0 : [0, 0] - f22_1 : [0, 0] - f22_2 : [1, 1] - f22_3 : [0, 0]
f23_0 : [0, 0] - f23_1 : [1, 1] - f23_2 : [1, 1] - f23_3 : [1, 1]

where every line represents a vector. We can see that the found values do not precisely compare to the expected values. This is because you can represent the MUB's in a lot of mathematically equivalent ways. Any unitary transformation applied uniformly to all the bases will preserve mutually unbiasedness.
The next step is looking for an extra basis. Again this file is quantized because we already know the expected values and solver speed. Suddenly, the solving time explodes. We have let this script run for 4881 minutes and 59 seconds, which is the equivalent to more than three full days of computing. We also tried computing the program for finding all four bases. We let this run for 6051 minutes and 12 seconds, which is over 100 hours of computing. For both these cases we could not get a final result on the used setup because of high computing times in combination with practical reasons.

5.5 $\mathcal{M}(5)$

For the problem in dimension five the solve times get infeasible large from the first calculation we try to do. We first tried to find 2 bases in dimension five. We make use of the quantized version because we suspect very high solve times. We let the calculation run for 734 minutes before we had to abort for practical reasons. We could therefore not get any meaningful results for the fifth dimension. We did write the script we suspect would work for finding all the bases in the fifth dimension, because we use the same method for finding the bases as for dimension two through four. We let this run for a total of 6118 minutes and 48 seconds. This is equivalent to over 100 hours of computing. Because of these high calculation times we have not finished finding the bases for dimension five.

5.6 $\mathcal{M}(6)$

When we apply all the methods mentioned before we can make a .smt2 file which would theoretically return whether or not $\mathcal{M}(6) < 4$. This file again has a too large solve time. We have run it 250+ hours with no result. The same goes for the quantized version. We then made a version where we would give the values of the existing bases and just try to look for one more. This would greatly decrease unnecessary complexity. We again have the standard computational basis in dimension six, M_0. The other two known bases are as discussed in section 2.1.2. We have already calculated what the corresponding $f(j, m, z)$ values are in section 4.5.
Using these known bases and the corresponding f-values from each vector we can construct a .smt2 file which is already partly-filled. We can use the same setup as the scripts before but assert for the phase values of the known bases the corresponding f-values. We can then also lose the constraints for everything that has nothing to do with M_3. For example the orthonomal constraints between of M_1 and M_2. We already know these bases are orthonormal. We can also lose a lot of the mutually unbiased constraints for we already know all the values of M_1 and M_2 satisfy the mutually unbiased constraints between each other.
Still with this decrease in complexity it is not feasable for the setup to be able to complete.

5.7 Runtimes

In this section we are going to look at the runtime of the experiments that were conducted. In table 1 we see the runtimes and sizes of the proof files of the different experiments conducted. All

the experiments are the quantized version and used $\delta = 0.001$.

Table 1: Runtime, proof size and result of different quantized experiments

Experiment	**Runtime (s)**	**Size proof file (GB)**	**Result**
$\mathcal{M}(2) < 3$	4.00×10^{-3}	N/A	δ-sat
$\mathcal{M}(2) < 4$	34.8	0.067	unsat
$\mathcal{M}(3) < 4$	1.86×10^{-1}	N/A	δ-sat
$\mathcal{M}(3) < 5$	7.44×10^{3}	14.45	unsat
$\mathcal{M}(4) < 3$	40.0	N/A	δ-sat
$\mathcal{M}(4) < 4$	3.63×10^{5}	N/A	timeout
$\mathcal{M}(5) < 3$	4.40×10^{4}	N/A	timeout
$\mathcal{M}(5) < 6$	3.67×10^{5}	N/A	timeout
$\mathcal{M}(6) < 4$	9.00×10^{5}	N/A	timeout

In table 2 we see the runtimes of the different continuous experiments conducted, for all the experiments we used $\delta = 0.001$.

Table 2: Runtime, proof size and result of different continuous experiments

Experiment	**Runtime (s)**	**result**
$\mathcal{M}(2) < 3$	1.69×10^{4}	δ-sat
$\mathcal{M}(2) < 4$	2.05×10^{5}	timeout
$\mathcal{M}(3) < 4$	2.65×10^{4}	δ-sat
$\mathcal{M}(3) < 5$	2.64×10^{5}	timeout
$\mathcal{M}(4) < 3$	2.93×10^{5}	timeout

In figure 3 we compare the runtimes of two quantized experiments to different precisions. The precesions range from 10^{-1} to 10^{-6}. All the experiments returned the same output. For this experiment we decided to use two scripts. One where the expected result was δ-sat and one where the expected result was unsat. The δ-sat script we used is mub_d4_l2.smt2 and the unsat script used is mub_d2_l3.smt2.

6 Conclusions and Discussion

6.1 Conclusion

The goal of this thesis was to research how SMT solvers like dReal4 can best be used to prove or disprove the conjecture of Zauner. We have discussed different solvers and different methods for using SMT solvers. We have found that indeed dReal4 is best used to prove the conjecture of Zauner, because of its support for trigonometrical functions, and dReal3 is necessary to disprove the conjecture because of it's function of providing a proof file. However, the proof generated by the dReal3 solver cannot yet be formally verified, we can create a human readable proof and construct a proof based on this file.

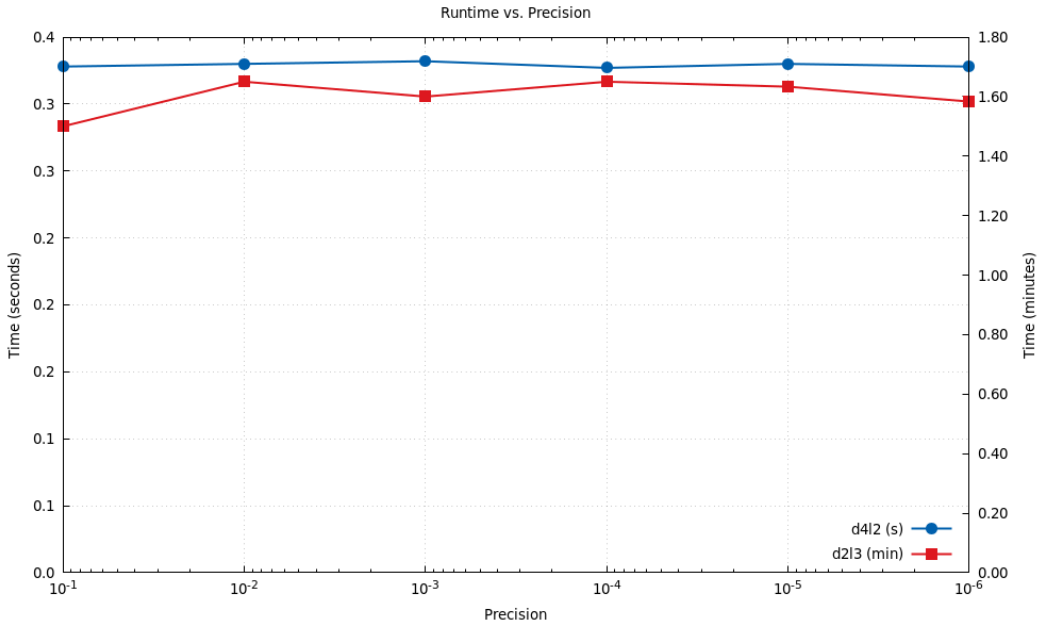

Figure 3: Average runtime of two quantized experiments with different precision

Different methods of encoding the mutually unbiased problem have been considered, including using the entire search space for $f(j, m, z) \in [0, 2)$ and a quantized version discussed in section 3.2. We see that if a basis within the quantized bounds exists, using quantized bounds is very efficient as seen in the significantly lower runtimes and can be used in proving the conjecture. However with an `unsat` result this version will not disprove the conjecture, for there could exist a basis in between these values.

We expected the precision to have more impact on the `unsat` script than on the δ-sat script. We thought this because the δ-sat script terminates early and is guided through the search space in the same manner no matter the precision. Whereas at the `unsat` script the search space increased and still the entire search space has to be used. In the results we see that a higher precision (lower value) does not really impact the runtime like we expected. The differences are minimal and vary per precision. We think this is because we use a quantized version of the problem, and therefore the precision does not influence the search space. A useful technique for finding additional bases is forwarding certain information. An example of this is used in the final version of the $d = 6$ problem, named mub_d6_l3_Fixed2.smt2. We believe this version is most efficient for disproving the conjecture and finding an additional basis. To prove the conjecture you have to use an unquantized version in combination with a proof checker.

These results lie within our expectations of the thesis. We can construct MUB's using the dReal4 SMT solver for smaller dimensions, however for large dimensions the calculations could not be performed within reasonable time on our setup.

6.2 Discussion

As found during the literature search, using SMT solvers to find MUB's has not yet been performed. Concluding from our research, we feel this identified research gap is a new area with a lot of

potential for finding MUB's. In the cases where the computational power of our setup was sufficient to produce a result within reasonable time, we have seen the speed and scalability of automated reasoning. In our research for example, when you have a working method in dimension two, scaling to dimension three is done efficiently. This could have a big effect on finding MUB's in the future, especially in non prime power dimensions where the construction of MUB's is mathematically very complex. Using SMT solvers we can bypass the construction of MUB's mathematically. We have made the first step towards using SMT solvers for finding MUB's.

The biggest limitation in our research was the computing power. Using our solutions on lower dimensions resulted in no problems in regards of computation time. However, when calculating for higher dimensions ($d > 4$) the calculation time became problematic. From the fifth dimension onward the solver has not finished running a single solution. The longest we have run a script was over 200 hours without a result. Because of these long computation times in combination with using a general purpose laptop, we had to cancel calculations after a certain amount of time, because of various reasons.

For further research we would suggest using super computers to cut down computation time. When using the method discussed in this thesis in combination with super computers further research could also include looking for MUB's in higher unknown dimensions. In future research we would also recommend using methods to reduce complexity even further, for example methods discussed by Mortimer [10].

Another area where further research would be valuable is the development of a proof checker capable of verifying proofs generated by the dReal SMT solver. One potential approach could involve translating dReal's proof output into a format compatible with more general-purpose proof checkers.

More further research could explore the quantized problem more deeply. While we used continuous SMT solvers for the quantized version, SAT solvers, being inherently discrete, may offer a more efficient approach.

Within this thesis, we have laid the foundation for the use of SMT solvers to prove or disprove the conjecture of Zauner.

References

[1] Soumen Pal, Manojit Bhattacharya, Sang-Soo Lee, and Chiranjib Chakraborty. Quantum computing in the next-generation computational biology landscape: From protein folding to molecular dynamics. *Molecular Biotechnology*, 66(2):163–178, 2024.

[2] William K Wootters and Brian D Fields. Optimal state-determination by mutually unbiased measurements. *Annals of Physics*, 191(2):363–381, 1989.

[3] Daniel Gottesman. Class of quantum error-correcting codes saturating the quantum hamming bound. *Phys. Rev. A*, 54:1862–1868, Sep 1996.

[4] M. Combescure. The mutually unbiased bases revisited, 2006.

[5] Stephen Brierley, Stefan Weigert, and Ingemar Bengtsson. All mutually unbiased bases in dimensions two to five, 2010.

[6] Andreas Klappenecker and Martin Roetteler. Constructions of mutually unbiased bases, 2003.

[7] Stephen Brierley and Stefan Weigert. Constructing mutually unbiased bases in dimension six. *Physical Review A*, 79(5), May 2009.

[8] Philippe Raynal, Xin Lü, and Berthold-Georg Englert. Mutually unbiased bases in six dimensions: The four most distant bases. *Physical Review A*, 83(6), June 2011.

[9] Ingemar Bengtsson, Wojciech Bruzda, Åsa Ericsson, Jan-Åke Larsson, Wojciech Tadej, and Karol Życzkowski. Mutually unbiased bases and hadamard matrices of order six. *Journal of Mathematical Physics*, 48(5):052106, 05 2007.

[10] Luke Mortimer. Mutually unbiased bases as a commuting polynomial optimisation problem. February 2024. Preprint, submitted February 28, 2024.

[11] Olivier ten Hoor. MUBwSMT: Mutually unbiased bases with satisfiability modulo theories. `https://github.com/Olivertenhoor/MUBwSMT`, 2025. Accessed: 2025-06-12.

[12] Christopher A. Fuchs. On the quantumness of a hilbert space, 2004.

[13] Maria Prat Colomer, Luke Mortimer, Irénée Frérot, Máté Farkas, and Antonio Acín. Three numerical approaches to find mutually unbiased bases using bell inequalities. *Quantum*, 6:778, 2022.

[14] Clark Barrett, A. Stump, and Cesare Tinelli. The smt-lib standard - version 2.0. In *Proceedings of the 8th international workshop on satisfiability modulo theories, Edinburgh, Scotland,(SMT '10)*, 2010.

[15] Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. δ-complete decision procedures for satisfiability over the reals. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning – IJCAR 2012*, volume 7364 of *Lecture Notes in Computer Science*, pages 286–300. Springer, 2012.

[16] Jekyll and Skinny Bones. dreal. `https://dreal.github.io/`, 2021. Accessed: 11/06/2025.

[17] Nikolaj Bjørner and Leonardo de Moura. Z3 theorem prover, 2025. Accessed: 2025-06-16.

[18] Arend-Jan Quist. Private communication, 2025. Private communication.

[19] Sicun Gao, Soonho Kong, and Edmund M. Clarke. dreal: An smt solver for nonlinear theories over the reals. In Maria Paola Bonacina, editor, *Automated Deduction – CADE-24*, pages 208–214, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.