An Algorithmic Approach to Analyzing Kochen-Specker Systems

1 Introduction

In 1967, Simon Kochen and Ernst Specker developed a theorem that placed constraints on the types of hidden-variable theories that may be used to explain quantum mechanical phenomenon. Specifically, they demonstrated the existence of a set $\mathcal{S} \subset \mathbb{R}^3$ such that there is no value function $f : \mathcal{S} \to \{0, 1\}$ satisfying

$$f(\vec{u}) + f(\vec{v}) + f(\vec{w}) = 1$$

for all triples $(\vec{u}, \vec{v}, \vec{w})$ of mutually orthogonal vectors in $\mathcal{S}$. To do so, Kochen and Specker constructed a set of 117 vectors that they had to manually verify did not meet the above condition given any value function. As mathematicians have started to turn to computers to solve mathematical problems such as the four color theorem, we may ask whether the same could be done for certain Kochen-Specker systems. That is, can a computer tell us whether a Kochen-Specker systems verifies the theorem above?

2 Constructing a Kochen-Specker System

Before developing a coloring algorithm, we must specify the types of mathematical objects we want to undergo this coloring process. While Kochen and Specker considered vectors in $\mathbb{R}^3$, we will be restricting ourselves to 3-dimensional integer vectors so that we may be able to better explore the number theoretic implications of the Kochen-Specker Theorem. We define the sets of vectors we want to color as follows:

$$\mathcal{S}(N) = \{\mathbf{v} \in \mathbb{Z}^3 : ||\mathbf{v}||^2 \text{ divides a power of } N\}.$$

The reasoning for this is that we can equate a Kochen-Specker coloring of this set with a Kochen-Specker coloring of rank-1 projection matrices whose entries lie in the same mathematical space as the orthogonal projection matrix of vectors in $\mathcal{S}(N)$. However, $\mathcal{S}(N)$ is an infinite set. Since computers can only store a finite amount of data, we must consider a subset of the original set. Thus, we will pivot towards considering the smaller set

$$s(N) = \{\mathbf{v} \in \mathbb{Z}^3 : ||\mathbf{v}||^2 \text{ divides } N\} \subset \mathcal{S}(N).$$

However, if we ever need more vectors to generate new coloring information for the set $\mathcal{S}(N)$, we can simply select a power of N as input for the set $s(N)$, as

$$s(N^k) \subset \mathcal{S}(N)$$

for all positive integers k.

To actually generate the set $s(N)$ in our Python program, we leverage a package that allows us to solve Diophantine equations of the form $x^2 + y^2 + z^2 = N$. By determining a solution to this equation, we get the entries of a vector in $s(N)$. Thus, all possible solutions to a fixed Diophantine equation yields all possible integers vectors in the set $s(N)$. In our GitHub repository, the `Sum_of_Squares_and_Divisibility.py` file contains the function `sum_of_squares_solutions` which extracts and permutes all solutions to the Diophantine equation mentioned above (the latter

ensures that we include negative solutions/vector entries). It returns a tuple with these solutions which we can interpret as the vector representations we are interested in coloring.

Now, to minimize the search our algorithm must perform, we consider primitive, well-signed vectors. Recall that an integer vector is *primitive* if its entries have a greatest common divisor equal to 1. An integer vector $\mathbf{v} \in \mathbb{Z}^3 \setminus 0$ is *well-signed* if either:

- $\mathbf{v}$ has only one nonzero entry which is positive,
- $\mathbf{v}$ has two nonzero entries and its first nonzero entry is positive, or
- $\mathbf{v}$ has three nonzero entries, at least two of which are positive.

The `Primitive_Well_Signed_Vectors.py` file contains the functions `primitive` and `well_signed`, which determine whether a tuple obtained from calling the `sum_of_squares_solutions` function is primitive and well-signed. These helper functions are used in the `primitive_well_signed_solutions` function, which takes as a parameter the positive integer N corresponding to the set of vectors $s(N)$ and returns the set of vectors we want to color according to the Kochen-Specker Theorem.

3 The Coloring Algorithm

We developed three different Python functions that automates the coloring of our set of vectors $s(N)$. All these versions, however, have similarities that can perhaps be abstracted in subsequent modifications. Specifically, we begin our coloring process with an empty dictionary that will contain the coloring of each vector and by determining the parity of N. If N is odd, then we add $(1,0,0)$, $(0,0,1)$, and $(0,1,0)$ to our dictionary. Notably, Cortez and Reyes were able to show that, without loss of generality, we can assign $(1,0,0)$ the value (i.e. color) 1 while assigning $(0,0,1)$ and $(0,1,0)$ the value 0. The reason for this is there exists a matrix such that each vector can be mapped to the each other. If N is even, then we also add $(1,0,-1)$, $(1,0,1)$, $(1,1,0)$, $(1,-1,0)$, $(0,1,1)$, and $(0,1,-1)$ to the dictionary. For similar reason as the vectors we added before, we can assign $(1,0,-1)$ and $(1,-1,0)$ the value 1 and the remaining 4 vectors the value 0. These basis vectors allow our algorithm to follow an iterative approach that will color the remaining set of vectors in $s(N)$ by comparing their orthogonality relative to those that have already been assigned a value.

The first thing we do is determine whether any 1-vectors in `color_dict` are orthogonal to the vectors in $s(N)$. Recall that in a Kochen-Specker system, all triples of mutually orthogonal vectors must have exactly one vector assigned the value 1 while the rest are assigned the value 0. Thus, if the dot product of a 1 vector and a vector $v \in s(N)$ yields 0, then we will assign v the value 0 and add it to `color_dict`. If however, we find that v was already in `color_dict` and was assigned the value 1, then the algorithm stops and returns that a contradiction has been identified. That is, any orthogonal triple containing the 1 vector we considered and v has more than one vector assigned the value 1.

Next we take the cross product of two orthogonal 0 vectors in `color_dict` so that we generate a third vector that is orthogonal to both 0-vectors. Since we altogether have an orthogonal triple, the vector we produced v must be assigned the value 1. If, however, we find that v was already in `color_dict` and was assigned the value 0, then the algorithm stops and returns that a contradiction has been identified.

4 Spectral Graph Theory

Let $G = (V, E)$ be a graph where V is the set of vertices representing vectors in a Kochen-Specker system and E is a set of edges between two vertices. Suppose $v_1, v_2 \in V$. There exists an edge $\{v_1, v_2\} \in E$ if the integer vectors corresponding to v_1 and v_2 are orthogonal.

[INSERT EXAMPLE FROM NOTES]

Now, let A be the adjacency matrix corresponding to the graph G and $\mathbf{c}$ be a coloring vector whose entries are defined as follows:

$$c_i = \begin{cases} 1 & \text{if the vector } v_i \text{ is assigned the value 1}\,, \\ 0 & \text{otherwise.} \end{cases}$$

We claim that the resulting vector $\mathbf{d} = A\mathbf{c}$ has entries where the ith entry corresponds to the number of 1 vectors adjacent to the v_i vector.

$$(A\mathbf{c})_i = \sum_j a_{ij} c_i$$