COMPUTER-AIDED DISCOVERY OF EXTREMAL UNIT-DISTANCE GRAPHS & QUANTUM CONTEXTUALITY

ANAY AGGARWAL

Abstract. We develop a general framework for the computer-aided discovery of extremal unit-distance graphs (UDGs) in Euclidean spaces and on spheres, motivated by problems in Euclidean Ramsey theory and quantum contextuality. Our methods aim to construct UDGs with large edge-density. We introduce two main computational paradigms: approximation-based methods, which relax unit-distance constraints via (ε, δ)-graphs, and lattice-based methods, which reduce the infinite search space using number-theoretic lattices. We implement and compare three algorithmic approaches—reinforcement learning, simulated annealing, and numerical optimization—demonstrating their effectiveness in constructing dense planar UDGs and offering promising pathways toward new bounds in quantum contextuality. Our framework adapts to higher-dimensional spaces via the Raiskii spindle and naturally embeds into the sphere $\mathbb{S}^d_{1/\sqrt{2}}$, discovering novel UDGs in both cases. This work lays the foundation for computational discovery of many types of extremal UDGs.

1. Introduction

Computer assistance has become increasingly prevalent in mathematical research and scientific research in general. There have been breakthrough results proven with computer assistance, such as the four-color theorem in 1976 by Appel and Haken [3]. More recently, Google DeepMind researchers introduced FunSearch, a framework for mathematical discovery using large language models and reinforcement learning, in their monumental 2023 paper [27]. They use FunSearch to make significant advances regarding multiple problems, most notably the cap-set problem. In addition, there have been major recent developments in the field of artificial intelligence for theorem proving, see [35, 33, 34, 2]. Using interactive proof assistants such as Lean [10] or Coq [31], paired with Large Language Models, these approaches aim to automatically prove mathematical theorems.

We consider a new, less direct approach to automated solving of research-level mathematical problems, one that may align better with human experience. The idea is that in order to gain intuition about a mathematical conjecture of the form "for all X, property Y holds", we have our machine first attempt to construct counterexamples X to the conjecture. If the machine successfully constructs a counterexample, then the conjecture is disproven. If the machine fails to successfully construct a counterexample, the hope is that it has either gained intuition about why a counterexample likely exists or why it likely doesn't (and hence why the conjecture is likely true). We hence focus on coming up with a computational aid that can construct counterexamples to various conjectures.

Wagner's approach [32] to discovering counterexamples in combinatorics pioneered this domain. Using the deep cross-entropy method, Wagner constructs counterexamples to various theorems in combinatorics, mostly from graph theory. He details a general procedure for using the deep cross-entropy method to discover combinatorial counterexamples. Charton et al. expand on this approach in [7] with their novel PatternBoost framework. These approaches have been applied to conjectures in many different domains, from Ramsey Theory [14] to Algebraic Geometry [8].

Date: February 11, 2026.

Key words and phrases. reinforcement learning, unit-distance graphs, kochen-specker set, simulated annealing.

In this paper, our aim is to construct computer-aided methods that can discover counterexamples to a unique class of problems that is more "continuous" than in previous work. In particular, we focus on Euclidean Ramsey Theory: a field that studies coloring and various other extremal problems on continuous (generally Euclidean) spaces.

A famous open problem posed by Edward Nelson in 1950 that is a prototypical example of what we wish to study is the Hadwiger-Nelson problem: that of determining the chromatic number $\chi(\mathbb{R}^2)$ [17]. In other words, determining the minimum number of colors that are needed to color the points of the plane so that no two points unit distance apart are assigned the same color. One may also study natural generalizations, such as computing $\chi(\mathbb{R}^d)$ for $d > 2$ [13] and $\chi(K^d)$ for certain fields K, particularly when $K = \mathbb{Q}(\sqrt{3}, \sqrt{11})$ [20]. Another problem in this area asks whether, for any triangle T, one can color $\mathbb{R}^2$ with two colors such that no monochromatic copy of T exists. We refer the reader to [15] for information on this problem, along with other prominent problems in Euclidean Ramsey Theory.

A useful theorem that allows us to reduce such "continuous" problems to discrete ones is the following:

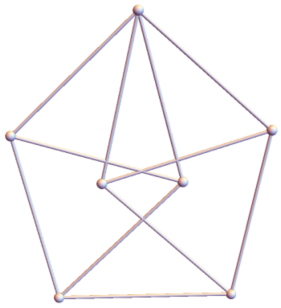
Theorem 1.1 (Compactness Principle)**.** *Let $H = (V, E)$ be a hypergraph where all $X \in E$ are finite (but V need not be). Suppose that, for all $W \subseteq V$, W finite, $\chi(H_W) \leq r$. Then $\chi(H) \leq r$.*

We refer the reader to [16] for the proof of this theorem as well as its broader implications in Ramsey Theory. For our purposes, this theorem reduces problems in $\mathbb{R}^d$ and K^d to producing *finite constructions*. This motivates the following definition.

Definition 1.2. The *unit-distance graph* of a set of points in $\mathbb{R}^d$ (or a subfield) is the undirected graph having those points as its vertices, with an edge between two vertices whenever their Euclidean distance is exactly one.

Hence the Hadwiger-Nelson problem is equivalent to finding the largest r such that there exists a finite unit-distance graph that is r-colorable. This reduction to finite constructions makes such problems great candidates for computer-based approaches. We aim to construct unit distance graphs with extremal properties, such as large chromatic number and high edge density. Such work can be extended naturally to other problems in Euclidean Ramsey Theory.

Currently, it is known that $5 \leq \chi(\mathbb{R}^2) \leq 7$ [25]. We focus on the lower-bound, which comes from constructions of unit-distance graphs with large chromatic number. The *Moser Spindle* [22] (depicted below) has chromatic number 4, showing $\chi(\mathbb{R}^2) \geq 4$. In 2018, de Grey introduced the first unit-distance graph with chromatic number 5 [9], improving the bound to its current state. This graph has 1581 vertices and was discovered with computational aid. One of the goals of the Polymath16 project [25] was to discover smaller unit-distance graphs. In 2022, Jaan Parts discovered the current record, a unit-distance graph with chromatic number 5 on only 509 vertices [23].



The Moser Spindle in $\mathbb{R}^2$

One is also interested in other open problems regarding unit-distance graphs, such as Erdős' unit-distance problem [30], which asks about the maximum number of edges a unit-distance graph on n vertices can have. These problems have been attacked computationally, e.g. in [11]. In this paper, we develop a general framework for computer-aided discovery of extremal unit-distance graphs (UDGs) in the language of this problem.

An interesting application of such examples is in *quantum contextuality*. The Kochen–Specker theorem asserts that in dimension three and higher, it is impossible to assign definite truth values (0 or 1) to all propositions about quantum measurements in a way that preserves functional relationships, ruling out non-contextual hidden variable theories [19]. This result is often proved via explicit configurations of vectors with specific orthogonality constraints. The initial proof boils down to a 117-vector construction, called a Kochen–Specker set. This 117-vector construction has been brought down to 33 vectors in $\mathbb{R}^3$ by Peres [24] (the current record in 3 dimensions). Fewer vectors are required in higher dimensions, e.g. Cabello's construction [6] in $\mathbb{R}^4$ only needs 18. Other extremal Kochen-Specker sets may provide for interesting quantum systems. See for example [28], which introduces a metric q on Kochen-Specker sets. Sets with high q values provide useful insight in the field of quantum logic.

There is a natural correspondence between Kochen-Specker sets and UDGs on the d-sphere with radius $1/\sqrt{2}$, denoted $\mathbb{S}^d_{1/\sqrt{2}}$. With a set of vectors $v_1, v_2, \ldots, v_n \in \mathbb{R}^d$, we may construct corresponding points $p_1, p_2, \ldots, p_n \in \mathbb{R}^d$ by projecting onto the sphere with radius $1/\sqrt{2}$. Two vectors are orthogonal if and only if the corresponding points are at distance 1, and so Kochen-Specker sets with the orthogonality relations are in direct bijection with UDGs. Because of this connection, we are motivated to study the problem of finding dense UDGs on the sphere.

The blueprint for the paper is as follows. In Section 2, we will prove mathematical properties of UDGs that will allow us to formulate possible computational frameworks for discovering extremal UDGs (in particular, UDGs with high edge density). We will focus on $\mathbb{R}^2$. In Section 3, we extend this framework to $\mathbb{R}^d$ and $\mathbb{S}^d_{1/\sqrt{2}}$. We introduce computational methods in Section 4 and combine the framework with these methods in Section 5.

2. Framework

The explicit problem we tackle is, given some search space S (generally $\mathbb{R}^d$ for some d), and a positive integer n, efficiently discover dense UDGs on n vertices that can be embedded in S. This problem extends to other useful problems in Euclidean Ramsey Theory, such as finding the chromatic number of S or solving Erdős' unit-distance problem.

When one tries to exploit the entire continuous search space to find UDGs, there is a problem. The probability of the distance between two randomly chosen points being exactly 1 is 0. Hence, one option is relax the notion of UDGs, and consider *approximation-based methods*. The other type of approach is to use algebraic properties of UDGs to somehow reduce the search space to a finite "lattice". We call these *lattice-based methods*. We will discuss both types of approaches, specifically focused on finding dense UDGs, but this naturally adapts to other types of problems (e.g. maximizing chromatic number). We will begin our study in $\mathbb{R}^2$. For approximation-based methods, this generalizes naturally to higher dimensions. For lattice-based methods, we will discuss how to generalize to $\mathbb{R}^d$ in the next section.

2.1. Approximation-Based Methods.

In order to remove some of the "continuity" in the search space, we may consider the relaxation of the notion of a UDG. In particular, we refer to the following definition.

Definition 2.1 ([12])**.** Given a set of points $P = (p_i)_{i=1}^n$, we may define the ε-unit distance graph of P as the graph G_P on the points in P with two points connected by an edge if their Euclidean distance lies in the interval $[1-\varepsilon, 1+\varepsilon]$.

In the literature, these graphs have mostly been studied through the lens of the chromatic number. Here, the idea is to search for ε-UDGs and hope that they are bona fide UDGs. However, not all ε-UDGs are bona fide UDGs. In fact, we may construct ε-UDGs that are much denser than any bona fide UDGs, and this is quickly learned in practice by computational methods.

Lemma 2.2. *Let $\varepsilon > 0$. Then there exist graphs that are ε-UDGs, yet not bona fide unit-distance graphs.*

Proof. We will prove that there exist much denser ε-UDGs on n vertices that bona fide unit-distance graphs. In particular, we may obtain an edge density of $\Omega(n^2)$ with the following construction. Consider $\lfloor n/2 \rfloor$ distinct points clustered around $(0,0)$ within a radius of $\varepsilon/2$. Consider $\lceil n/2 \rceil$ distinct points clustered around $(1,0)$ within a radius of $\varepsilon/2$. The resulting graph is on n vertices and has at least $\lfloor n/2 \rfloor \lceil n/2 \rceil = \Omega(n^2)$ edges, as desired. This graph is illustrated below. The result follows because unit-distance graphs on n vertices have $O(n^{4/3})$ edges [30].

□

Hence, in order to make approximation-based methods work, we need to be able to modify approximate UDGs so that they are bona fide UDGs. In particular, we wish to avoid the type of construction in Lemma 2.2. This motivates the following definition.

Definition 2.3. Let G be an ε-UDG. Let $\delta > 0$ and suppose that for all $u, v \in V(G)$, $||u - v|| \geq \delta$. Then we call G an (ε, δ)-unit-distance graph.

The main result is as follows:

Theorem 2.4. *There exist functions $\varepsilon(n), \delta(n) > 0$ such that any $(\varepsilon(n), \delta(n))$ unit-distance graph in $\mathbb{R}^d$ on n vertices is a unit-distance graph.*

Proof. The idea is that the (ε, δ) unit-distance graph condition is polynomial. Assume that the graph is connected and not a path graph. For points $p_1, p_2, \ldots, p_n \in \mathbb{R}^d$ that are the vertices of an (ε, δ) unit-distance graph $G = (V, E)$, consider the function

$$f(p_1, \ldots, p_n) = \sum_{ij \in E} (||p_i - p_j||^2 - 1)^2 \prod_{1 \leq i \neq j \leq n} (n^2 - ||p_i - p_j||^2).$$

This function satisfies the property that

$$f(p_1, \ldots, p_n) < (n^2 - \delta^2)^{\binom{n}{2}} (\varepsilon^2 + 2\varepsilon)|E|.$$

Additionally, $f(p_1, \ldots, p_n) = 0$ if and only if all $||p_i - p_j|| = 1$, i.e. G is a unit-distance graph. This is because, by the triangle inequality, $||p_i - p_j|| \leq n$, and equality cannot hold because we assumed that G is not a path graph. Therefore, the result follows by Theorem 1 of [18]. □

Remark 2.5. The ε given by the proof of the above theorem for a fixed δ is doubly exponentially small (of the form $c^{-d^{\mathrm{poly}(n)}}$), so it isn't feasible to use in practice directly. Nonetheless, this makes way for a potential approximate approach, as one may consider such a graph for a relaxed value of ε and exhibit a process involving moving the points around small neighborhoods to reduce ε.

Using this, we may define a framework for finding dense UDGs in $\mathbb{R}^d$:

(1) Specify the number n of vertices of our UDG.
(2) Define a parameter H such that our graph lies in the box $[-H, H]^d$. This tells us how large the space that our graph lives in is. The choice $H = n$ is always satisfactory (by the triangle inequality), but for more efficient search one may choose smaller values of H.

(3) Our search space $\mathscr{S}$ is the set of all subsets of $[-H, H]^d$ of size at most n. Our action space consists of all actions that correspond to adding a point of norm 1 to a point in our state (if possible) or removing a point in our state (if possible), such that all points are at distance at least δ from each other.
(4) For a given state s we may specify the reward function (depending on the computational method used) to be the edge density of the corresponding ε-UDG G_s: $|E(G_s)|/|V(G_s)|$.

Step 4 may be modified based on the problem we're studied. For example, if we're trying to optimize for high chromatic number, we may specify a reward function correlated with the chromatic number of G_s.

2.2. **Lattice-Based Methods.** In order to reduce the action space even further (and hence reduce the complexity of the problem), we may exploit algebraic structure of UDGs. The key observation is the following theorem, allowing us to reduce from the uncountable $\mathbb{R}$ to a countable number field.

Theorem 2.6. *Let G be a finite unit-distance graph G in $\mathbb{R}^d$. Then there exists a number field $K \subset \mathbb{R}$ such that there exists a finite unit-distance graph G' in K^d such that G is a subgraph of G'.*

Proof. Let $(x_{1,1}, x_{1,2}, \ldots, x_{1,d}), (x_{2,1}, \ldots, x_{2,d}), \ldots, (x_{n,1}, \ldots, x_{n,d})$ be the vertices of G, with all $x_{i,j}$ in $\mathbb{R}$. The edges of G are defined by relations of the form

$$\sum_{i=1}^{d} (x_{a,i} - x_{b,i})^2 = 1.$$

The union ϕ of these relations over all edges of G forms a system of polynomial equations in the $x_{i,j}$. By assumption this system has a solution over $\mathbb{R}$. It is enough to show that it also has a solution over some number field $K \subset \mathbb{R}$. Then, we can map the vertices of G to vertices of a graph G' in K^d such that each edge of G has a corresponding edge in G', and the result follows.

Note that while $\mathbb{R}$ is not algebraically closed, it is a real closed field, and the theory of real closed fields is complete. The existence of real solutions to ϕ is a first-order property in the language of ordered fields. Hence by the argument behind the *Lefschetz Principle*, ϕ has a solution over the real algebraic numbers, $\mathbb{R} \cap \overline{\mathbb{Q}}$. Because G is finite, a solution lies in a subfield of $\mathbb{R} \cap \overline{\mathbb{Q}}$, which is a finite extension of $\mathbb{Q}$. This is our desired number field K, and we're done. □

Therefore, instead of searching over the uncountable $\mathbb{R}^d$, we can select K and search over the countable K^d. We can show that the action space is actually *finite* if we specify a denominator with the following theorem.

Theorem 2.7. *Let K be a totally real number field which is finite over $\mathbb{Q}$. Let $m \in \mathbb{N}$ and O_K be the ring of integers of K. Then there are finitely many vectors $\mathbf{v} \in \left(\frac{1}{m} \cdot O_K\right)^d$ such that $||\mathbf{v}|| = 1$.*

Proof. It is enough to show that

$$x_1^2 + x_2^2 + \cdots + x_d^2 = r$$

has finitely many solutions for $x_1, x_2, \ldots, x_d \in O_K$. Let $\omega_1, \omega_2, \ldots, \omega_n$ be an integral basis for O_K. Let $x_j = \sum_{i=1}^{n} \alpha_{ji}\omega_i$, and let $\alpha_j = (\alpha_{j1}, \alpha_{j2}, \ldots, \alpha_{jn})^\intercal$, where all the α_{ji} are integers. Additionally, if $\sigma_1, \sigma_2, \ldots, \sigma_n$ are the embeddings of K, then define the matrix

$$M = \begin{bmatrix} \sigma_1(\omega_1) & \cdots & \sigma_n(\omega_1) \\ \vdots & \ddots & \vdots \\ \sigma_1(\omega_n) & \cdots & \sigma_n(\omega_n) \end{bmatrix},$$

so that

$$Ma_j = \begin{bmatrix} \sigma_1(x_j) \\ \sigma_2(x_j) \\ \vdots \\ \sigma_n(x_j) \end{bmatrix}.$$

For each embedding, we have that

$$\sigma_i(x_1)^2 + \sigma_i(x_2)^2 + \cdots + \sigma_i(x_d)^2 = r,$$

so $\sigma_i(x_j) \leq \sqrt{r}$ for each j and all i. This means that $||Ma_j||_\infty \leq \sqrt{r}$ for all j. But then, because M is invertible,

$$||a_j||_\infty = ||M^{-1}(Ma_j)||_\infty \leq ||M^{-1}||_{\infty\to\infty}||Ma_j||_\infty \leq ||M^{-1}||_{\infty\to\infty}\sqrt{r}.$$

Therefore, $||a_j||_\infty$ is bounded, so there are finitely many solutions for each α_{ji} and hence finally many solutions to the original equation. □

In the case of $\mathbb{R}^2$, we detail a method for algorithmically computing the unit-distances. First, we need to prove Theorem 2.7 in a new way that gives us a tighter bound.

Theorem 2.8. *Let $K \subset \mathbb{R}$ be a number field which is finite over $\mathbb{Q}$. Let $m \in \mathbb{N}$ and O_K be the ring of integers of K. Then there are finitely many vectors $\mathbf{v} \in \left(\frac{1}{m} \cdot O_K\right)^2$ such that $||\mathbf{v}|| = 1$. The number of vectors is bounded above by $I_{K(i)}(m^{\mathrm{rank}(O_K)})|\mathrm{Tors}(O^\times_{K(i)})|$.*

Proof. For every $m \in \mathbb{N}$, we prove that there are finitely many $\mathbf{v} \in (O_K)^2$ such that $||\mathbf{v}|| = m$.

We begin by proving this for $m = 1$. Let us embed $\mathbf{v} \in O_{K(i)}$. Consider the natural norm function $N : K(i) \to K$. The norm $N(x + yi)$ is given by $x^2 + y^2$, so we have $N(\mathbf{v}) = 1$. These elements of norm 1 are all invertible and hence form a group, as $x + yi$ has inverse $x - yi$. In particular, this group is the kernel of the natural map $\varphi : O^\times_{K(i)} \to O^\times_K$. We have that

$$\mathrm{rank}(\ker \varphi) = \mathrm{rank}(O^\times_{K(i)}) - \mathrm{rank}(O^\times_K) + \mathrm{rank}(\mathrm{coker}\ \varphi).$$

Note that if $u \in O^\times_K$ is the norm of $w \in O^\times_{K(i)}$, then u^2 is also a norm, namely that of w^2. Hence the image of φ is a finite index subgroup of $O^\times_K$ and the cokernel has 0 rank. Because $K \subset \mathbb{R}$, K is totally real. Each real embedding of K corresponds to a complex embedding of $K(i)$, so we get by Dirichlet's unit theorem that the rank of the kernel is finite, and hence our theorem is proved for $m = 1$.

Now, a solution to $||\mathbf{v}|| = m$ generates an ideal in $K(i)$. This ideal has finite norm, in particular norm $m^{\mathrm{rank}(O_K)}$. Consider all principal ideals of norm $m^{\mathrm{rank}(O_K)}$. There are finitely many such ideals. It follows from the $m = 1$ case that each ideal has finitely many generators: if two elements generate the same ideal, they differ by multiplication by a unit under the norm map, but there are finitely many units. Therefore there are finitely many vectors $\mathbf{v}$ with magnitude m, and we're done. In particular, the number of such vectors is bounded above by

$$I_{K(i)}(m^{\mathrm{rank}(O_K)})|\ker \varphi|,$$

where $I_K(d)$ denotes the number of ideals in the ring of integers of K with norm d. Because $\ker \varphi$ is a finite subgroup of $O^\times_{K(i)} \cong \mathrm{Tors}(O^\times_{K(i)}) \times \mathbb{Z}^r$ for some r, where $\mathrm{Tors}(G)$ denotes the torsion subgroup of an abelian group G, we can improve this bound to

$$I_{K(i)}(m^{\mathrm{rank}(O_K)})|\mathrm{Tors}(O^\times_{K(i)})|.$$

□

Now, all we must do is specify a number field K and a common denominator m to embed our unit-distance graph in. Then, our action space is finite (even if our observation space is a dense subset of $\mathbb{R}^2$) and the problem is amenable to computational methods. Notice that this idea aligns with the known constructions of unit-distance graphs, as the Moser Spindle corresponds to $K = \mathbb{Q}(\sqrt{3}, \sqrt{11})$ and some $m \mid 2$, and Parts' 509-vertex graph corresponds to $K = \mathbb{Q}(\sqrt{3}, \sqrt{5}, \sqrt{11})$ and some $m \mid 96$. In both these cases, the vectors corresponding to actions are embedded in particular subrings of O_K: for the Moser Spindle this is $\mathbb{Z}[\sqrt{3}, \sqrt{11}]$ and for Parts' graph this is $\mathbb{Z}[\sqrt{3}, \sqrt{5}, \sqrt{11}]$. With this information, we can significantly reduce our search space by only considering ideals of order $m^{\text{rank}(R)}$, where R is the given subring. Either way, one can use Sage [29], a mathematical software system, to quickly generate the (now-finite) action space. Based on past constructions, when inputting K and m it seems that we should set $K = \mathbb{Q}(\sqrt{p_1}, \sqrt{p_2}, \ldots, \sqrt{p_n})$ for odd primes $p_1, \ldots, p_n$ and $m = 2^a \cdot 3^b$ for some $a, b \in \mathbb{N}$.

Remark 2.9. The case of the Moser Spindle is particularly fruitful because the resulting lattice, the *Moser Lattice*, is the location of the densest possible UDGs with at most 21 vertices, and the densest known UDGs with between 22 and 30 vertices (see [11] and [1]). Eventually there must be some threshold point where a different lattice produces denser UDGs, but this is likely for some reasonably large n.

Hence, the following steps allow us to generate a lattice in $\mathbb{R}^2$. We will discuss lattices in higher dimensions in the next Section.

(1) Specify a number field $K = \mathbb{Q}(\sqrt{p_1}, \sqrt{p_2}, \ldots, \sqrt{p_n})$ for odd primes $p_1, \ldots, p_n$ and a common denominator $m = 2^a \cdot 3^b$ for some $a, b \in \mathbb{N}$.
(2) In Sage, compute the torsion subgroup of the group of units of $O_{K(i)}$. This effectively gives us the "roots of unity" in K^2. The code for this is in Appendix C.
(3) Compute in Sage all principal ideals of $O_{K(i)}$ with norm $m^{\text{rank}(O_K)}$ (or $m^{\text{rank}(R)}$ for a subring R), identifying a generator for each one. The code is again in Appendix C.
(4) Test each of these generators, making sure they satisfy $||\mathbf{v}|| = m$ and removing the ones that don't.
(5) We should at this point have a list of generators $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$. For each generator create an "orbit" of vectors with $||\mathbf{v}|| = m$ by multiplying by the precomputed roots of unity. The union of all of our resulting orbits will give us our action space.

Given a lattice $\mathscr{L} \in \mathbb{R}^d$, we may detail a framework for computational discovery of dense UDGs embedded in the lattice.

(1) Specify the number n of vertices of our UDG.
(2) Our search space is a subset of $[-n, n]^d \cap \mathscr{L}$. For our purposes, we may simply let it be $[-n, n]^d$ – this doesn't matter in terms of complexity.
(3) Let $g_1, g_2, \ldots, g_m$ where $m = \text{rank}(\mathscr{L})$ generate $\mathscr{L}$. Then our set of possible actions consists of adding some g_i to some point in our current state (if this keeps the number of vertices at most n) or removing a vertex (if possible).

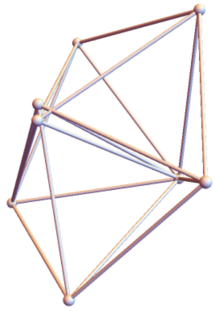
3. UDGs in Higher Dimensions and Spheres

In this section, we'll focus on how to adapt the lattice approach detailed in the previous section to $\mathbb{R}^d$ and $\mathbb{S}^d_{1/\sqrt{2}}$.

3.1. Action Spaces in $\mathbb{R}^d$.

First, we will discuss the problem of finding a suitable action space for UDGs in $\mathbb{R}^d$ for $d > 2$. These results are of independent interest, as they will allow us to compute small extremal UDGs, just as was done in $\mathbb{R}^2$. We do this by considering the lattice generated by the *Raiskii Spindle* [26], a generalization of the Moser Spindle to higher dimensions.

Definition 3.1. Let P be the object formed by gluing two regular unit simplices in $\mathbb{R}^d$ to each other on a face. Join P with a copy of P, say P', at a vertex v. Let v_P and $v_{P'}$ be the vertices of P and P' (respectively) which are not connected to v by an edge. Rotate P' until $||v_P - v_{P'}|| = 1$. The resulting object is the Raiskii Spindle in $\mathbb{R}^d$. We call any lattice $\mathscr{L} \subset \mathbb{R}^d$ containing a Raiskii Spindle a *Raiskii Lattice*. It is not hard to see that the chromatic number of the Raiskii Spindle in $\mathbb{R}^d$ is $d+2$.



The Raiskii spindle in $\mathbb{R}^3$

Clearly we may find a Raiskii Lattice with rank at most $2(d-1)$ with algebraic generators. The question is, in what number field can these generators lie in? When $d = 2$, we get the Moser Lattice $\mathscr{L}_{moser} \subset \mathbb{Q}[\sqrt{3}, \sqrt{11}]^2$. We may generalize this with the following theorem.

Theorem 3.2. *For all $d \geq 2$, there exists a Raiskii Lattice $\mathscr{L}_d \subset \mathbb{Q}[\sqrt{7d^2+8d}, \sqrt{2}, \sqrt{d+1}]^d$ with rank at most $2(d-1)$.*

Proof. We will detail how to construct such a Raiskii Spindle. Let $v_1, v_2, \ldots, v_d$ be vectors such that the tips of the vectors along with the origin forms a regular unit simplex. We may take

$$v_i = \frac{1}{\sqrt{2}}\left(e_i - \left(\frac{1+\sqrt{d+1}}{d}\right)\cdot(1,1,\ldots,1)\right)$$

for an orthonormal basis e_i. It is not hard to check that these vectors satisfy the require condition. Then the last vertex to form P is the tip of $\frac{2(v_1+\cdots+v_n)}{d}$. It is easy to see that the height of the simplex is $\sqrt{\frac{d+1}{2d}}$. Therefore the angle between $\vec{v_P}$ and $\vec{v_{P'}}$, θ, satisfies (by the Law of Cosines)

$$\frac{4(d+1)}{d} - \frac{4(d+1)}{d}\cos\theta = 1,$$

so $\cos\theta = \frac{3d+4}{4d+4}$. Now we may construct P' by rotating the v_i an angle θ about the origin in some plane. We may do this by applying the Givens matrix

$$G(1,2,\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & \cdots & 0 \\ \sin\theta & \cos\theta & 0 & \cdots & 0 \\ 0 & 0 & 1 & & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix},$$

i.e. $v_i' = G(1,2,\theta)v_i$. The vectors v_i and v_i' generate the lattice $\mathscr{L}_d$ in question. Since $\sin\theta = \frac{\sqrt{7d^2+8d}}{4d+4}$, this implies that

$$\mathscr{L}_d \subset \mathbb{Q}[\sqrt{7d^2+8d}, \sqrt{2}, \sqrt{d+1}]^d,$$

as desired. □

Remark 3.3. Complexity of extremal UDG discovery is expected to scale exponentially with the rank of the lattice. Hence it is vital that we keep our lattice of small rank.

3.2. **Action Spaces on the Sphere.** Another natural setting to search for extremal UDGs is on high-dimensional spheres. For example, Kochen-Specker sets are generally written as orthogonality graphs in high dimensions, but these are equivalent to UDGs on spheres in $\mathbb{R}^d$ with radius $1/\sqrt{2}$. Because the sphere is a subset of $\mathbb{R}^d$, it is a corollary of Theorem 2.7 that the search space is again a lattice, so the reader may ask why we are treating this case separately. The reason is that with the lattice approach we must ensure that every point we're adding is indeed on the sphere (and this is a rare occurrence). There is a more natural way to find UDGs on the sphere, especially the sphere with radius $1/\sqrt{2}$, that exploits the sphere's structure.

First, let us address the natural question of whether UDG embeddings in $\mathbb{R}^d$ and $\mathbb{S}^d_{1/\sqrt{2}}$ behave similarly. After all, both spaces are equivalent by stereographic projection (though this does not preserve unit distances). However, it turns out that their UDGs are in fact quite different. We may see this for $d = 2$, for example, with relatively simple counterexamples.

Proposition 3.4. *There exist graphs G which*

(a) cannot be embedded as UDGs in $\mathbb{S}^2_{1/\sqrt{2}}$, but can be embedded as UDGs in $\mathbb{R}^2$
(b) cannot be embedded as UDGs in $\mathbb{R}^2$, but can be embedded as UDGs in $\mathbb{S}^2_{1/\sqrt{2}}$.

Proof.

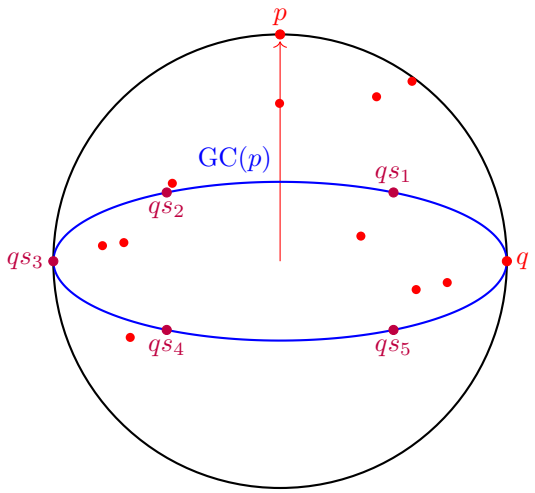
(a) Let G be the unit-distance graph formed by 6 vertices of a unit hexagon along with its center. Let A be the center vertex and label the outer vertices B, C, D, E, F, G in that order. We claim that if this graph were to be embedded in $\mathbb{S}^2_{1/\sqrt{2}}$, the images of B and D are antipodes. This yields a contradiction, because this implies $B = -F = D = -B$. The reason that this is true is simple: the space of vectors on the sphere at a unit-distance from a point on the sphere is a great circle. Hence B and D both lie at an intersection of the corresponding great circles of A and C. Because A and C are a unit-distance apart, these two great circles have exactly 2 intersections. By assumption, the images of B and D are distinct, yielding the result. Note that, by the same reasoning, the Moser Spindle is another example of such a graph.
(b) Take $G = K_{2,2,2}$. This can be embedded on the sphere as the unit octahedron. But it has no UDG embedding in the plane: the vertices in two of the three partitions must form a square, but the other two vertices must both be the circumcenter of this square.

□

So, to specify an action space on the sphere, it will be helpful to get away from the idea of linearly adding unit-distance offsets to the current set of points (as is done for $\mathbb{R}^d$). Instead, we may specify a symmetric set of "angles" on the great circle perpendicular to a given vector in the current set of vectors. This set of angles serves as the generator of our "lattice". The framework is as follows:

(1) Let S be a finite subset of $\mathbb{R}/2\pi\mathbb{Z}$ initially specified.
(2) For each point p in the current set of points, consider the subset $qS \subset \mathrm{GC}(p)$ of the great circle on the sphere perpendicular to p, where q is a chosen vector from the current set.
(3) The action space is $\bigcup_p qS$.

In essence, our action space given a base point p is a set of points on the great circle perpendicular to p. This set of points is characterized by an anchor point q and a set of rotations S. One may compute qS



The Process for $S = \{s_1, s_2, s_3, s_4, s_5\} = \{\pi/3, 2\pi/3, \pi, 4\pi/4, 5\pi/3\}$

using Rodrigues' rotation formula. If we wish to compute the rotation of q an angle θ about the axis p, we may use

$$R_{p,\theta}(q) = q\cos\theta + (\hat{p} \times q)\sin\theta + \hat{p}(\hat{p} \cdot q)(1 - \cos\theta),$$

where $\hat{p} = \frac{p}{||p||}$ is the unit vector in the direction of p. In this case, $\hat{p} \cdot q = 0$ by assumption, so we only need

$$R_{p,\theta}(q) = q\cos\theta + (\hat{p} \times q)\sin\theta.$$

The question of choosing the lattice on the sphere is not as straightforward as $\mathbb{R}^d$. One may choose to use angles that are simple rational multiples of π. For example, $S = \{0, \pi/2, \pi/3, 2\pi/3, \pi, 4\pi/3, 3\pi/2, 5\pi/3\}$. One may also choose to go with angles that show up in popular constructions, such as [24]. We will see in Section 5 that it is hard to predict the result.

4. Computational Approaches

There are three main computational approaches that we use. The current state of the art, [11], uses a diverse beam search. Through stochastic and machine learning methods, we attempt to improve this.

4.1. **Reinforcement Learning.**

Reinforcement learning (RL) is a framework for sequential decision-making where an agent interacts with an environment to learn a policy that maximizes cumulative reward. Formally, this setting is modeled as a Markov Decision Process (MDP), defined by a tuple $(\mathscr{S}, \mathscr{A}, P, R, \gamma)$, where $\mathscr{S}$ is the state space, $\mathscr{A}$ is the action space, P represents transition probabilities, R the reward function, and γ a discount factor.

In combinatorial optimization problems, RL provides a flexible and model-free approach for constructing feasible solutions incrementally. The agent learns a policy $\pi(a \mid s)$ that maps states to actions, gradually building up structures such as graphs, sets, or sequences. Unlike traditional heuristics, RL methods can adaptively learn problem-specific strategies and generalize across problem instances.

Among policy gradient methods, *Proximal Policy Optimization* (PPO) has proven particularly effective

due to its balance of stability and efficiency. PPO updates the policy by maximizing a clipped surrogate objective:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right],$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio and $\hat{A}_t$ is an estimator of the advantage function. The clipping mechanism constrains the policy updates, preventing overly large steps that destabilize learning.

PPO is particularly well-suited to combinatorial domains because it can handle large, discrete action spaces, tolerate sparse or delayed rewards, and benefit from rich neural architectures to model policies over complex structures. This makes it a strong candidate for learning constructive heuristics in problems where the search space is vast and non-differentiable.

4.2. **Simulated Annealing.** Simulated annealing is a probabilistic optimization technique inspired by the annealing process in metallurgy, where a material is heated and then slowly cooled to reduce defects and reach a state of minimum energy. In the context of optimization, simulated annealing explores the solution space of an objective function by probabilistically accepting both improving and worsening moves. The probability of accepting a worse solution decreases over time, governed by a temperature parameter T that gradually cools according to a predefined schedule.

At each step, a candidate solution x' is generated from the current solution x by a local perturbation. The new solution is accepted with probability

$$P(x \to x') = \begin{cases} 1, & \text{if } f(x') \le f(x), \\ \exp\left(-\frac{f(x')-f(x)}{T}\right), & \text{otherwise,} \end{cases}$$

where f is the objective function to be minimized. This mechanism allows the algorithm to escape local minima early on, with convergence to a global minimum more likely if the cooling schedule is sufficiently slow. We refer the reader to [4] for more details.

4.3. **Numerical Optimization.** The website [21] employs a gravity simulation to discover small dense unit distance graphs. We attempt to do something similar using our notion of (ε, δ)-UDGs. In particular, we wish to construct a smooth function between two points that is large when the points are very close to 1, and tapers off quickly when they deviate from 1 even slightly. We want this function to have a "repelling" property in the sense that it should be negative when two points are at distance $\le \delta$. We may do this using the smooth bump functions

$$f(x) = \begin{cases} \exp\left(-\dfrac{1}{\varepsilon^2 - (x-1)^2}\right), & \text{if } |x-1| < \varepsilon \\ 0, & \text{otherwise} \end{cases}$$

and

$$h(x) = \begin{cases} -\exp\left(-\dfrac{1}{\delta^2 - x^2}\right), & |x| < \delta \\ 0, & |x| \ge \delta \end{cases}$$

For a point set $p_1, p_2, \ldots, p_n \in \mathbb{R}^d$ we may define the "reward function"

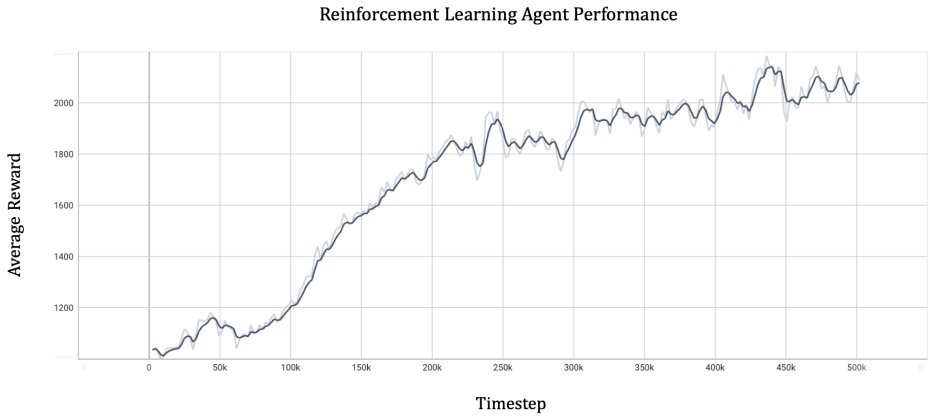
$$R_{\varepsilon,\delta}(p_1, p_2, \ldots, p_n) = \sum_{i,j} f(||p_i - p_j||) + h(||p_i - p_j||).$$

We may then employ gradient descent and other numerical maximization methods to find a global maximum for this function. One may also modify this approach in a way that is similar to [21] to instead

involve particles in space and forces between particles based on the distance between them (repelling force at distance $\leq \delta$ and equilibrium at distance 1), then simulate the trajectories of the points in order to minimize the potential energy of their state.

5. Empirical Results

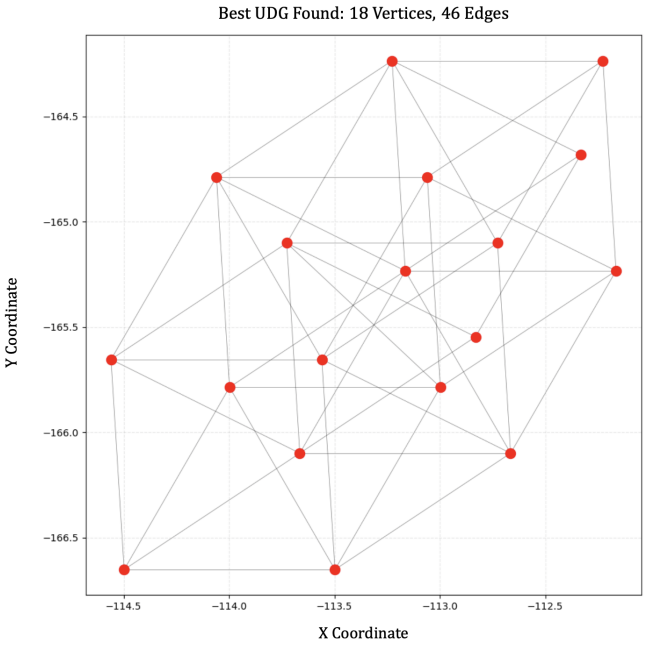
For reinforcement learning, we use the OpenAI Gym environment [5] for implementation. OpenAI Gym requires a static action space, so we set our action space to be the union of all universally acceptable actions, then mask based on the current state. The approximation-based framework underperforms the lattice-based framework. The results are best for the Moser Lattice. In practice, the RL agent ends up learning to consistently come up with decently dense UDGs, but tends to fail to come up with optimally dense UDGs after short amounts of training. Below is a graph showing roughly how well the agent performs when working with $n = 30$ vertices on the Moser Lattice over $5 \cdot 10^5$ episodes of training on a Macbook Pro. The vertical axis represents the agents average reward (with irrelevant scale) and time is on the horizontal axis.



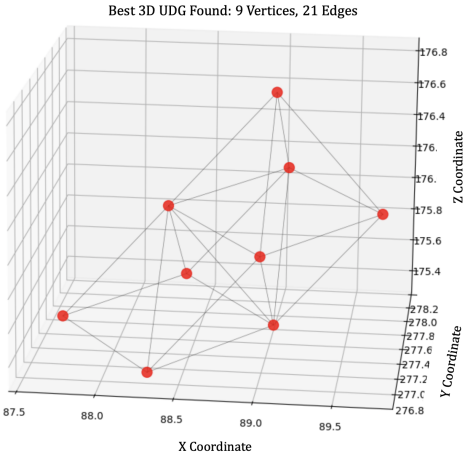
This approach was unfruitful as it requires significantly more compute and produces worse results than simulated annealing.

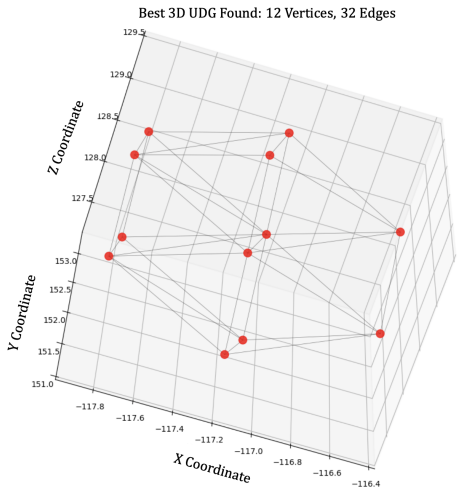
Numerical optimization performed poorly as a consequence of the fact that the input space is high dimensional, and the problem is not naturally continuous. This and the complexity of the reward function made it hard to learn the landscape of UDGs. In addition, the UDG landscape has a plethora of local extrema that it is hard for many numerical optimization methods to escape from to find global extrema (even for methods like stochastic gradient descent). The results found were not much better than [21] for large graphs.

In $\mathbb{R}^2$, simulated annealing was able to reproduce the results in [11] with much less compute power. In particular, multithreaded among 11 cores on a 2019 MacBook Pro, all the densest unit-distance graphs up to 30 vertices were reproduced in just under 30 minutes. The Python code for the main functions are in Appendix A. Below is the densest UDG on 18 vertices, found by simulated annealing.



Similarly, with the Raiskii Spindle in $\mathbb{R}^3$ and the same hardware, simulated annealing finds with high certainty the densest UDGs in three-dimensions for small numbers of vertices. Below are the densest UDGs on 9 and 12 vertices.



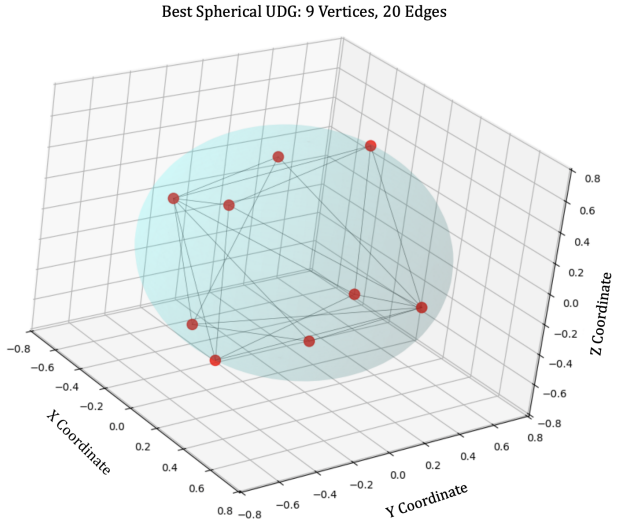


Below is a table of the lower bounds found for $u_3(n)$, maximum number of edges of a UDG in $\mathbb{R}^3$ with n vertices, for small n.

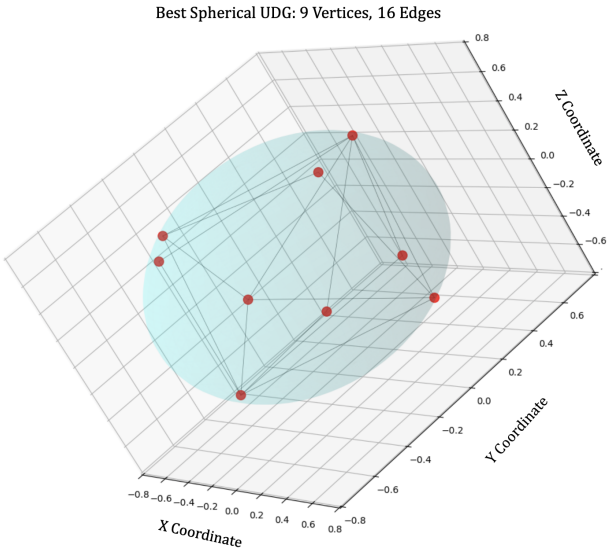
n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$u_3(n) \geq$	0	1	3	6	8	12	15	18	21	25	28	32	36	39	43

The process of upper-bounding these values to find exact counts for $u_3(n)$ is tedious yet straightforward, as it is for $u_2(n)$. The idea is to find small forbidden UDGs and then run a brute-force search. We refer the reader to [1] for details on how to perform this project.

Like $\mathbb{R}^2$ and $\mathbb{R}^3$, simulated annealing worked just as well on the sphere in 3 dimensions. The Python code for the main functions are in Appendix B. However, there was counter-intuitive behavior when choosing angle sets S. Unlike in previous cases, where the Moser and Raiskii spindles are likely the most efficient choices for generators of the action space, there is not a simple choice for S. One may decide to choose simple rational angles, for example with $S_1 = \{0, \pi/2, \pi/3, 2\pi/3, \pi, 4\pi/3, 3\pi/2, 5\pi/3\}$. For S_1, below is the densest UDG found on 9 vertices.



If one decides to make a more complicated choice, the results are not what one would expect. Choosing a 372 angle set that contains all the angles in Peres' construction for example, below is the densest UDG found on 9 vertices.



Because simulated annealing is so definitive in its final answer, the discrepancy is likely not a result of simulated annealing finding suboptimal UDGs but rather a limitation of the generating set. Hence we ask the question: what generating set produces the densest small UDGs on the sphere?

6. Conclusion

This work develops a unified framework for the computational discovery of extremal unit-distance graphs. The central difficulty, the pathological sparsity of exact unit distances in continuous spaces, was addressed in two complementary ways: approximation-based methods that relax the metric constraints in a controlled fashion, and lattice-based methods that exploit the intrinsic algebraic properties of UDGs. Together, these approaches transform an a priori intractable continuous search problem into one that is both finite and computationally navigable.

This framework was tested computationally and, even with the author's compute limitations, produced novel results about UDGs in 3-dimensions and on the sphere. It paves the way for future work in this direction.

This task is not only important because of Erdös' problem on dense UDGs, but because we can use information about dense UDGs to solve important problems about other types of extremal UDGs, as was done famously by de Grey in [9]. Because of the flexibility of the framework developed in this work, many generalizations (to higher dimensions, to number fields, etc.) are now tractable to study by observing computational results.

The Github repository for this paper can be found here.

Acknowledgments

The author would like to thank the MIT PRIMES-USA program and its coordinators for providing the opportunity for this research experience. The author thanks his mentors Andrew Gritsevskiy and Jesse Geneson for providing invaluable guidance, feedback, and resources.

References

[1] Boris Alexeev, Dustin G. Mixon, and Hans Parshall. The erdős unit distance problem for small point sets, 2025. arXiv:2412.11914 [math.CO].

[2] Chenyang An, Zhibo Chen, Qihao Ye, Emily First, Letian Peng, Jiayun Zhang, Zihan Wang, Sorin Lerner, and Jingbo Shang. Learn from failure: Fine-tuning llms with trial-and-error data for intuitionistic propositional logic proving. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 776–790, 2024.

[3] Kenneth Appel and Wolfgang Haken. The solution of the four-color-map problem. *Scientific American*, 237(4):108–121, 1977.

[4] Dimitris Bertsimas and John Tsitsiklis. Simulated annealing. *Statistical Science*, 8(1):10–15, 1993.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. arXiv:1606.01540 [cs.LG].

[6] Adán Cabello, José M. Estebaranz, and Guillermo García-Alcaine. Bell–kochen–specker theorem: A proof with 18 vectors. *Physics Letters A*, 212(4):183–187, 1996.

[7] François Charton, Jordan S. Ellenberg, Adam Zsolt Wagner, and Geordie Williamson. Patternboost: Constructions in mathematics with a little help from ai, 2024. arXiv:2411.00566 [math.CO].

[8] Tom Coates, Alexander Kasprzyk, and Sara Veneziale. Machine learning detects terminal singularities. In *Advances in Neural Information Processing Systems*, volume 36, pages 67183–67194, 2023.

[9] Aubrey D. N. J. de Grey. The chromatic number of the plane is at least 5, 2018. arXiv:1804.02385 [math.CO].

[10] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In *Proceedings of CADE*, 2015.

[11] Peter Engel, Owen Hammond-Lee, Yiheng Su, Dániel Varga, and Pál Zsámboki. Diverse beam search to find densest-known planar unit distance graphs, 2024. arXiv:2406.15317 [math.CO].

[12] Geoffrey Exoo. ε-unit distance graphs. *Discrete & Computational Geometry*, 33(1):117–123, 2005.

[13] Geoffrey Exoo and Dan Ismailescu. On the chromatic number of $\mathbb{R}^n$ for small values of n, 2014. arXiv:1408.2002 [math.CO].
[14] Mohammad Ghebleh, Salem Al-Yakoob, Ali Kanso, and Dragan Stevanović. Reinforcement learning for graph theory, ii. small ramsey numbers, 2024. arXiv:2403.20055 [math.CO].
[15] Ron Graham and Eric Tressler. Open problems in euclidean ramsey theory. In *Ramsey Theory: Yesterday, Today, and Tomorrow*, pages 115–120. Springer, 2011.
[16] Ronald L. Graham, Bruce L. Rothschild, and Joel H. Spencer. *Ramsey Theory*. Wiley, 1991.
[17] Branko Grünbaum, Alexander Soifer, Peter Johnson, and Cecil Rousseau. *The Mathematical Coloring Book: Mathematics of coloring and the colorful life of its creators*. Springer, 2008.
[18] Gabriela Jeronimo, Daniel Perrucci, and Elias Tsigaridas. On the minimum of a polynomial function on a basic closed semialgebraic set and applications. *SIAM Journal on Optimization*, 23(1):241–255, 2013.
[19] Simon Kochen and Ernst P. Specker. The problem of hidden variables in quantum mechanics. *Journal of Mathematics and Mechanics*, 17(1):59–87, 1967.
[20] David A. Madore. The hadwiger–nelson problem over certain fields, 2015. arXiv:1509.07023 [math.CO].
[21] Kenneth J. Moore. Point sets with many unit distances. Online resource.
[22] Leo Moser and William Moser. Solution to problem 10. *Canadian Mathematical Bulletin*, 4:187–189, 1961.
[23] J. Parts. Graph minimization, focusing on the example of 5-chromatic unit-distance graphs in the plane. *Geombinatorics*, 29(4):137–166, 2020.
[24] Asher Peres. Two simple proofs of the kochen–specker theorem. *Journal of Physics A: Mathematical and General*, 24(4):L175, 1991.
[25] Polymath Project. Hadwiger–nelson problem. Online resource.
[26] D. E. Raiskii. Realization of all distances in a decomposition of the space $\mathbb{R}^n$ into $n+1$ parts. *Mathematical Notes*, 7:194–196, 1970.
[27] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, Alhussein Fawzi, Josh Grochow, Andrea Lodi, Jean-Baptiste Mouret, Talia Ringer, and Tao Yu. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2023.
[28] Andrew W. Simmons. How (maximally) contextual is quantum mechanics?, 2017. arXiv:1712.03766 [quant-ph].
[29] W. A. Stein et al. *Sage mathematics software (Version 10.5)*. The Sage Development Team, 2025.
[30] Endre Szemerédi. Erdős's unit distance problem. In *Open Problems in Mathematics*, pages 459–477. Springer, 2016.
[31] The Coq Development Team. The coq reference manual, release 8.18.0, 2023. Available at https://coq.inria.fr.
[32] Adam Zsolt Wagner. Constructions in combinatorics via neural networks, 2021. arXiv:2104.14516 [math.CO].
[33] Sean Welleck, Jiacheng Liu, Ronan Le Bras, Hannaneh Hajishirzi, Yejin Choi, and Kyunghyun Cho. Naturalproofs: Mathematical theorem proving in natural language, 2021. arXiv:2104.01112 [cs.IR].
[34] Minchao Wu, Michael Norrish, Christian Walder, and Amir Dezfouli. Tacticzero: Learning to prove theorems from scratch with deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 9330–9342, 2021.
[35] Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models, 2023. arXiv:2306.15626 [cs.LG].

Appendix A. Simulated Annealing Code in $\mathbb{R}^d$

Below is the Python function to generate the action space in $\mathbb{R}^2$. The corresponding function in $\mathbb{R}^3$ and higher dimensions is essentially the same.

```
def generate_neighbor(points: List[Point], min_vertices: int = MIN,
                      max_vertices: int = MAX) -> List[Point]:
    new_points = points.copy()
    n = len(new_points)

    # Decide to add or remove — 0.2 probability of just removing anyway
    if n <= min_vertices:
        action = 'add'
    elif n >= max_vertices or np.random.rand()<0.2:
        action = 'remove'
    else:
        action = random.choice(['add', 'remove'])

    if action == 'remove':
        new_points.pop(random.randint(0, n - 1))
    else:
        validpt = False
        while validpt == False:
            base_point = random.choice(new_points)
            dx, dy = random.choice(UNIT_DISTANCE_OFFSETS)
            new_point = (base_point[0] + dx, base_point[1] + dy)
            validpt = True
            for p in points:
                if(euclidean_distance(p, new_point) < 0.001):
                    validpt = False
        new_points.append(new_point)
    return new_points
```

Note the parameter $p = 0.2$ of removing a vertex anyway. This can be tuned for optimal results. The simulated annealing function is as follows (make note of the parameters used in the definition).

```
def simulated_annealing(
    initial_points: List[Point],
    max_iterations: int = 1000000,
    initial_temp: float = 100.0,
    cooling_rate: float = 0.999995,
    min_vertices: int = MIN,
    max_vertices: int = MAX
) -> Tuple[List[Point], int]:
    current_points = initial_points.copy()
    current_edges = count_edges(current_points)
    best_points = current_points.copy()
    best_edges = current_edges

    temp = initial_temp
```

```
    for i in range(max_iterations):
        neighbor_points = generate_neighbor(current_points, min_vertices,
                                            max_vertices)
        neighbor_edges = count_edges(neighbor_points)

        # Acceptance probability (maximizing edges)
        if neighbor_edges > current_edges or
           random.random() < math.exp((neighbor_edges - current_edges) / temp):
            current_points = neighbor_points
            current_edges = neighbor_edges

            if current_edges > best_edges:
                best_points = current_points.copy()
                best_edges = current_edges

        # Cool down
        temp *= cooling_rate

    return best_points, best_edges
```

Appendix B. Simulated Annealing Code on the Sphere

Below is the Python code to perform a Rodrigues rotation in $\mathbb{R}^3$.

```
def rodrigues_rotation(point: Point, axis: Point, angle: float) -> Point:
    # Convert to numpy arrays for vector operations
    v = np.array(point)
    k = np.array(axis)

    # Normalize the axis
    k = k / np.linalg.norm(k)

    # Rodrigues' formula
    cos_theta = np.cos(angle)
    sin_theta = np.sin(angle)

    v_rot = (v * cos_theta +
             np.cross(k, v) * sin_theta +
             k * np.dot(k, v) * (1 - cos_theta))

    return (v_rot[0], v_rot[1], v_rot[2])
```

Below are the functions to compute the action space for the sphere in three dimensions.

```
def build_perpendicularity_graph(points: List[Point], tolerance: float = 0.001)
    -> dict:
    perp_graph = {i: [] for i in range(len(points))}
```

```
    for i in range(len(points)):
        for j in range(len(points)):
            if i != j:
                dot_prod = np.dot(np.array(points[i]), np.array(points[j]))
                if abs(dot_prod) < tolerance:
                    perp_graph[i].append(j)

    return perp_graph

def generate_neighbor_spherical(points: List[Point], angle_set: List[float],
                                min_vertices: int = MIN, max_vertices: int = MAX)
                                -> List[Point]:
    new_points = points.copy()
    n = len(new_points)

    # Decide to add or remove
    if n <= min_vertices:
        action = 'add'
    elif n >= max_vertices or np.random.rand() < 0.2:
        action = 'remove'
    else:
        action = random.choices(['add', 'remove'], weights=[60, 40])[0]

    if action == 'remove':
        # Try removing a random point, but check if it disconnects the graph
        remove_idx = random.randint(0, n - 1)
        test_points = new_points[:remove_idx] + new_points[remove_idx+1:]

        # If removal would leave us with 0 edges, don't do it
        if count_edges(test_points) == 0:
            return points  # Return original, no change
        else:
            new_points = test_points
    else:
        # Build perpendicularity graph once
        perp_graph = build_perpendicularity_graph(new_points)

        # Find all P that have at least one perpendicular point
        valid_P_indices = [i for i in range(n) if len(perp_graph[i]) > 0]

        if len(valid_P_indices) == 0:
            return points  # Can't add, return original

        validpt = False
        attempts = 0
        max_attempts = 100

        while not validpt and attempts < max_attempts:
```

```
            attempts += 1

            # Choose base point P that has perpendicular points
            P_idx = random.choice(valid_P_indices)
            P = new_points[P_idx]

            # Choose Q from pre-computed perpendicular points
            Q_idx = random.choice(perp_graph[P_idx])
            Q = new_points[Q_idx]

            # Choose rotation angle from angle set
            theta = random.choice(angle_set)

            # Rotate Q around axis P by angle theta
            new_point = rodrigues_rotation(Q, P, theta)

            # Check if this point is too close to existing points
            validpt = True
            for p in new_points:
                if euclidean_distance(p, new_point) < 0.04:
                    validpt = False
                    break

        if validpt:
            new_points.append(new_point)
        else:
            return points

    return new_points
```

Appendix C. Sage Code to Compute Offsets

Below is the Sage Code corresponding to Theorem 2.8.

```
def roots_of_unity(K):
    G = K.unit_group()
    T = G.torsion_generator()
    order = (T.value()).multiplicative_order()
    torsion_subgroup = [T^k for k in range(order)]
    torsion_subgroup_values = [x.value() for x in torsion_subgroup]
    return torsion_subgroup_values

def ideals_of_norm(K, n):
    ideals_list = []
    for p, k in factor(n):
        Jp = K.ideal(p)
        p_ideals = Jp.factor()
        prime_ideals = [P for P, e in p_ideals]
```

```
        residue_degrees = [P.residue_class_degree() for P in prime_ideals]

        def find_exponents(degrees, target):
            if not degrees:
                return [[]] if target == 0 else []
            current_degree = degrees[0]
            remaining_degrees = degrees[1:]
            solutions = []
            for e in range(0, target // current_degree + 1):
                for sol in find_exponents(remaining_degrees,
                                          target - e*current_degree):
                    solutions.append([e]+sol)
            return solutions

        exponent_combinations = find_exponents(residue_degrees, k)
        ideals_for_pk = []
        for exponents in exponent_combinations:
            ideal_product = K.ideal(1)
            for P, e in zip(prime_ideals, exponents):
                ideal_product *= P^e
            ideals_for_pk.append(ideal_product)
        ideals_list.append(ideals_for_pk)

    from itertools import product
    all_ideals = list(product(*ideals_list))
    final_ideals = [prod(ideals) for ideals in all_ideals]
    return final_ideals

def action_space(K, n):
    return list(set([w * t for w in roots_of_unity(K) for t in ideals_of_norm(K, n)]))
```

PRIMES USA, MIT, Cambridge, MA 02139
Email address: anay.aggarwal.2007@gmail.com