

The maximum cardinality of a planar seven-distance set

Alon Amit and GPT-6 Astra

24 October 2026

Abstract

We give a computer-assisted proof that the maximum cardinality of a planar set determining at most seven distinct positive distances is sixteen. A diameter-graph argument reduces a hypothetical seventeen-point example to an eleven-point set with at most six distances, except for a thirteen-cycle obstruction. We exclude the cycle by an exact interval certificate. Using a published exhaustive count of sixty-two candidate distance patterns, we classify the eleven-point cores by exact Gröbner-basis calculations and prove that none can have the required seventeen-point extension. All extension calculations use rational or algebraic arithmetic. An explicit sixteen-point triangular-lattice set attains the bound.

1 Introduction

For a finite set $X \subset \mathbb{R}^2$, put

$$\Delta(X) = \{\|x - y\| : x, y \in X, x \neq y\}, \quad g_2(s) = \max\{|X| : |\Delta(X)| \leq s\}.$$

All distances are Euclidean and positive. We use “at most” so the property is hereditary.

Theorem 1. *Every finite subset of $\mathbb{R}^2$ determining at most seven distances has cardinality at most sixteen. This bound is attained; hence $g_2(7) = 16$.*

Erdős and Fishburn [1] established the planar maxima through five distances and exhibited sixteen-point seven-distance sets. Wei [2] proved $g_2(6) = 13$; Szöllősi and Östergård [3] completed the classification at cardinality thirteen. Budianto [4] obtained partial exclusions for a hypothetical seventeen-point seven-distance set using diameter graphs. Bao and Yu [5] studied specified triangular-lattice palettes and conjectured the unrestricted value sixteen.

Our proof uses the exhaustive candidate count at order eleven in [3, Table 9], rather than repeating the large combinatorial search that produced it. We reconstruct all sixty-two candidate patterns by exhibiting that many inequivalent realizable patterns. Exact algebra then exhausts their metrics. This distinction matters: a count of colored patterns alone is not a classification of geometric realizations. The remaining extension and cycle obstructions are finite computations whose coverage is justified geometrically. The proof is computer-assisted, not a proof-assistant formalization.

2 Published inputs and notation

Write $D = \text{diam } X$ and let X_D be the endpoints of pairs at distance D . The diameter graph $G_D(X)$ has vertex set X and an edge xy exactly when $\|x - y\| = D$. Its induced graph on X_D has no

isolated vertices. Write $m = |X_D|$. A set independent in $G_D(X)$ has at most six distances when X has at most seven.

We use the following published facts.

- (i) A convex h -gon determines at least $\lfloor h/2 \rfloor$ distances. An odd convex $(2k+1)$ -gon determining k distances is regular. For even $h \geq 8$, a convex h -gon determining $h/2$ distances is regular or is obtained by deleting one vertex from a regular $(h+1)$ -gon. See [1, Lemma 2].
- (ii) Diameter endpoints are extreme vertices, and disjoint diameter segments cross. A planar diameter graph has no even cycle and at most one cycle. If its graph on m nonisolated vertices is not C_m , its independence number is at least $\lceil m/2 \rceil$. See [2, Lemmas 3–5 and 9].
- (iii) A set with at most six distances has at most thirteen points [2, 3].
- (iv) The exhaustive generation of [3, Sections 3.1 and 4, Table 9] leaves exactly sixty-two candidate distance patterns at order eleven, using at most six colors, modulo permutations of vertices and colors. Every eleven-point planar set with at most six distances is represented among these candidates. This finite-generation result is an external input to our proof.

All references to a convex polygon mean its extreme vertices, in cyclic boundary order. We write R_n for the vertices of a regular n -gon and O for its center.

3 Relation to Budianto's reductions

Budianto's thesis [4] studies a hypothetical seventeen-point seven-distance set by the number m of endpoints of diameter pairs. He excludes $m \leq 8$ and $m \geq 14$, and leaves $9 \leq m \leq 13$ partially unresolved. In particular, pages 9–13 establish that when $m = 13$, the maximum degree of the diameter graph is two; its remaining forms are C_{13} and unions of paths. The tables for $9 \leq m \leq 12$ are expressly incomplete, and are not used here as classification theorems.

The larger-diameter extension obstruction for the three thirteen-point six-distance configurations already appears in Budianto's argument on pages 15–19. For the hexagram, he obtains six possible additional points and three incompatible opposite pairs; the two circular configurations admit no addition introducing a larger diameter. These obstructions already explain why a thirteen-point core is useful. The present proof goes further by classifying and extending eleven-point cores. His exclusion of fourteen and fifteen diameter endpoints on pages 7–9 also precedes our hull reduction; the squared-distance sum proof gives a direct argument for the stronger bound on all hull vertices. The full thirteen-point classification is supplied by Szöllősi and Östergård [3, Theorem 16].

These conclusions do not give a bound of eighteen in the thesis. The exceptional thirteen-cycle remains explicitly unresolved there. The following route through smaller distance sets is a separate finite classification and extension argument.

4 A bound on the number of hull vertices

Lemma 2. *If $|X| \geq 16$ and $|\Delta(X)| \leq 7$, then $\text{conv } X$ has at most thirteen vertices. In particular, $|X_D| \leq 13$.*

Proof. The convex-polygon bound gives at most fifteen hull vertices. If there are fifteen, they form R_{15} . A noncentral point has any given distance to at most two vertices of R_{15} , since two distinct circles intersect in at most two points. It therefore sees at least eight distances. The center sees

the circumradius, which is not an existing chord length: $2 \sin(\pi j/15) = 1$ has no integral solution $1 \leq j \leq 7$. Thus there is no additional point.

If there are fourteen hull vertices, they form R_{14} or R_{15} with one vertex v deleted. Normalize the circumcenter to 0 and radius to 1. An additional point q is in their convex hull and has $r = \|q\| < 1$. The center is again excluded because neither 14 nor 15 is divisible by 6. Otherwise the fourteen distances from q must realize all seven old distances exactly twice.

For R_{14} the sum of its seven distinct squared chord lengths is

$$\sum_{j=1}^7 (2 - 2 \cos(2\pi j/14)) = 16.$$

Since the vertices sum to zero, summing the fourteen squared distances from q gives $14(1 + r^2) = 32$, contrary to $r < 1$.

For $R_{15} \setminus \{v\}$ the distinct squared chord lengths sum to 15, and the fourteen vertices sum to $-v$. The same calculation gives

$$14(1 + r^2) + 2q \cdot v = 30, \quad 7r^2 + q \cdot v = 8.$$

But $7r^2 + q \cdot v \leq 7r^2 + r < 8$. This excludes fourteen vertices as well. $\square$

5 Extensions of a regular thirteen-gon

Lemma 3. *If X contains R_{13} and determines at most seven distances, then $X \subseteq R_{13} \cup \{0\}$, where 0 is the center of R_{13} .*

Proof. Normalize the circumradius to 1. Set $a = \pi/13$, $c_j = \cos(2ja)$ for $0 \leq j \leq 6$, and $\beta_j = 2 - 2c_j$ for $1 \leq j \leq 6$. Let $q \notin R_{13} \cup \{0\}$. Each distance from q occurs at most twice among the thirteen vertices, so there are exactly seven distinct values. There is an equal-distance pair; its perpendicular bisector is a reflection axis of the regular polygon and contains both 0 and q . As the polygon has odd order, we may write $q = tv_0$ for some vertex v_0 and nonzero real t . The seven distinct squared distances are

$$f_j = t^2 + 1 - 2tc_j \quad (0 \leq j \leq 6).$$

Their set must contain all six β_j , since their union with those six values has size at most seven. Equivalently, the affine map $F(z) = (1 - t^2)/2 + tz$ maps six of $c_0, \dots, c_6$ onto $c_1, \dots, c_6$. Let c_h be the omitted argument.

Put $K = 2 \sin a$ and $g_j = c_j - c_{j+1} = K \sin((2j+1)a)$ for $0 \leq j \leq 5$. The following strict inequalities follow by product-to-sum identities:

$$g_1^2 - g_0g_2 = K^2 \sin^2(2a) > 0, \tag{1}$$

$$g_1g_4 - g_2g_5 = K^2 \sin a \sin(2a) > 0, \tag{2}$$

$$g_0g_4 - g_1g_5 = -K^2 \sin a \sin(2a) < 0. \tag{3}$$

If $t > 0$, order is preserved. For $h = 0$, F fixes the endpoints of $\{c_1, \dots, c_6\}$ and is the identity. For $1 \leq h \leq 4$, it fixes c_5, c_6 and is again the identity, now an impossibility since c_0 is retained. For $h = 5$ or 6 , the first three arguments map to c_1, c_2, c_3 , forcing $g_1/g_0 = g_2/g_1$, contrary to (1). The identity case gives $t = 1$, an existing vertex.

If $t < 0$, the five consecutive gaps in the retained six-element argument set must be proportional to $(g_5, g_4, g_3, g_2, g_1)$. For $h = 0$ this forces $g_1g_4 = g_2g_5$, contradicting (2). For $h = 1$ it forces

$(g_0 + g_1)g_4 = g_2g_5$, an even stronger contradiction. For $h = 2$, the last two gaps force $g_1g_4 = g_2g_5$ again. For $3 \leq h \leq 6$, the first two gaps force $g_0g_4 = g_1g_5$, contradicting (3). Thus no such q exists. $\square$

6 A certified rigidity theorem for the thirteen-cycle

This section gives a finite, exact certificate for the only additional geometric obstruction needed in our proof of the main theorem. The computation uses integer arithmetic throughout; numerical root finding is not a proof input. In particular, the assertions below concern all real configurations satisfying the hypotheses, not a bounded coordinate search.

Lemma 4. *Let $Y \subset \mathbb{R}^2$ have thirteen points. If its diameter graph is C_{13} and Y determines at most seven distances, then Y is similar to the vertex set of a regular thirteen-gon.*

We first give the geometric reduction, then specify and verify the finite algebraic assertion on which the lemma depends.

6.1 The order of the diameter edges and the two side lengths

The endpoints of diameters are in strictly convex position. Write them as $p_0, \dots, p_{12}$ in counter-clockwise order, with subscripts modulo thirteen, and normalize the diameter to one. Every two vertex-disjoint diameter segments cross. Fix one diameter edge and delete its endpoints from the diameter cycle. The remaining eleven vertices form a path, all of whose edges cross the fixed diameter. Its vertices consequently alternate between the two open half-planes bounded by that diameter's line. The two boundary arcs between its endpoints therefore contain five and six vertices. Thus every diameter edge joins indices differing by six or seven. Each vertex has two diameter neighbors, so both such edges occur. The diameter cycle is exactly the step-six star.

For a boundary side p_jp_{j+1} , the perpendicular bisector passes through p_{j+7} : this vertex is at distance one from both endpoints. The line through the side midpoint and p_{j+7} meets the convex polygon exactly in the segment between these two points. Hence the two boundary arcs between them lie strictly in the two respective open half-planes. Applying the resulting comparison of distances to successive sides gives

$$|p_i - p_{i+1}| < |p_i - p_{i+2}| < \dots < |p_i - p_{i+6}| = 1 \quad (i \in \mathbb{Z}/13\mathbb{Z}). \quad (4)$$

In particular, there are at least six distances. If there are only six, every boundary side has the shortest length. If there are seven, every boundary side is one of the two shortest lengths.

Traverse the diameter star in steps of six. At each star vertex the two neighbors are consecutive in the convex order, so the angle between the two incident diameter segments is the apex angle of an isosceles triangle with two sides of length one and a boundary side as its base. If just one boundary-side length occurs, these angles are all equal. The star then has equal edge lengths and equal turns and is regular, proving the lemma in this case.

Suppose the boundary sides have two distinct lengths $a < b$. Their apex angles are

$$\alpha = 2 \arcsin(a/2), \quad \beta = 2 \arcsin(b/2), \quad 0 < \alpha < \beta. \quad (5)$$

Let ν of the thirteen successive star angles equal β , and the other $u = 13 - \nu$ equal α . The signed exterior turn at a star vertex is $\pi - \alpha$ or $\pi - \beta$. The step-six star on convexly ordered vertices has rotation index six. Therefore

$$u\alpha + \nu\beta = \pi, \quad 1 \leq \nu \leq 12, \quad 0 < \alpha < \frac{\pi}{13}. \quad (6)$$

For completeness, the rotation-index assertion does not depend on any metric regularity. During a deformation through strictly convex labeled polygons, each successive star triple remains counterclockwise and noncollinear. Its exterior turn consequently varies continuously in $(0, \pi)$, and the rotation index remains an integer-valued constant. The space of strictly convex labeled polygons is connected. One explicit way to deform a polygon containing the origin to a cyclic polygon is to interpolate its Minkowski functional with the Euclidean norm and move each vertex along its ray to the corresponding unit level set; for positive interpolation parameter these level sets are strictly convex. The points on the resulting circle can then be moved to equally spaced positions while preserving their order. For the regular step-six star the sum of exterior turns is 12π , as asserted.

6.2 The closure polynomial

Encode the successive star angles by a binary word $b = (b_0, \dots, b_{12})$, taking $b_k = 1$ at an angle β and $b_k = 0$ at an angle α . Set

$$B_k = \sum_{j < k} b_j, \quad \theta = \frac{\alpha}{\nu}, \quad \omega = e^{-\pi i/\nu}, \quad z = e^{-i\theta}. \quad (7)$$

Equation (6) gives $\beta = \pi/\nu - u\theta$. After rotating the first unit edge to the positive real axis, the successive edge vectors are

$$e_k = (-1)^k \omega^{B_k} z^{\nu k - 13B_k} \quad (0 \leq k \leq 12). \quad (8)$$

Indeed, the direction of the k th edge is $k\pi - \sum_{j < k} ((1 - b_j)\alpha + b_j\beta)$. The required closure condition is therefore

$$P_b(z) := \sum_{k=0}^{12} (-1)^k \omega^{B_k} z^{\nu k - 13B_k} = 0, \quad 0 < \theta < \frac{\pi}{13\nu}. \quad (9)$$

Multiplication by a power of z changes P_b into an ordinary polynomial $\tilde{P}_b \in \mathbb{Z}[\omega][z]$ without changing its zeros on the unit circle. The exponents are positions in an integer walk whose successive increments are ν or $-u$. Its total positive increment is $u\nu$, so

$$\deg \tilde{P}_b \leq u\nu \leq 42. \quad (10)$$

There are $2^{13} - 2 = 8190$ nonconstant binary words. Alternatively, cyclic shifts and reversals give 378 classes. The verifier supports both the reduced enumeration and an enumeration of all 8190 words. The latter avoids any reliance on the symmetry reduction.

6.3 Exact removal of regular and degenerate solutions

Let r be the inverse of thirteen modulo 2ν and set $\eta = \omega^r$. The inverse exists because $1 \leq \nu \leq 12$. Define the monic degree-twelve polynomial

$$F_\nu(z) = \frac{z^{13} - \omega}{z - \eta} = \sum_{j=0}^{12} \eta^{12-j} z^j. \quad (11)$$

Every $\tilde{P}_b$ is divisible by F_ν in $\mathbb{Z}[\omega][z]$. To see this, at a root of $z^{13} = \omega$ equation (9) reduces to

$$\sum_{k=0}^{12} (-z^\nu)^k.$$

Here $(-z^\nu)^{13} = -\omega^\nu = 1$. The geometric sum vanishes unless $z^\nu = -1$. Among the thirteen roots of $z^{13} = \omega$, the latter condition holds precisely at η , since $\gcd(13, \nu) = 1$. The divisor is monic, and its division into $\tilde{P}_b$ is also checked directly by the verifier with a zero remainder.

On the closed arc

$$z = e^{-i\theta}, \quad 0 \leq \theta \leq \frac{\pi}{13\nu}, \quad (12)$$

the polynomial F_ν can vanish only at the upper endpoint. This is the regular solution $\alpha = \beta = \pi/13$. Its removed roots therefore include no nonregular admissible solution. Write

$$\tilde{P}_b(z) = F_\nu(z)(z-1)^{h_b}Q_b(z), \quad Q_b(1) \neq 0, \quad (13)$$

where the maximal integer $h_b \geq 0$ is found by exact evaluation and synthetic division. The removed factor $z-1$ corresponds only to the excluded endpoint $\theta = 0$. All coefficient operations in this step take place in

$$\mathbb{Z}[\omega] \cong \mathbb{Z}[T]/(\Phi_{2\nu}(T)),$$

where $\Phi_{2\nu}$ is constructed by exact monic divisions of $T^j - 1$. No algebraic-number approximation is used for factor removal.

Proposition 5. *For each of the 8190 nonconstant binary words b of length thirteen, the quotient Q_b in (13) is nonzero throughout the closed arc (12).*

The following gives the complete certification procedure for this finite assertion, rather than treating a numerical root computation as evidence of nonexistence.

6.4 The integer interval certificate

Fix the common scale $S = 10^{50}$. An integer pair (L, U) represents the real interval $[L/S, U/S]$. Addition, negation and integer scaling are exact. For multiplication, form the four integer products of endpoints and round the least product down and the greatest product up after dividing by S . Division by a positive integer is rounded in the same way. Thus every arithmetic operation encloses all possible real inputs from its operand intervals. A complex interval is a pair of such real intervals, and complex multiplication uses $(a+ib)(c+id) = (ac-bd) + i(ad+bc)$.

A rational enclosure for π is obtained from Machin's identity

$$\pi = 16 \arctan(1/5) - 4 \arctan(1/239). \quad (14)$$

For each arctangent the verifier takes the first one hundred terms of its alternating power series and encloses the remainder by the next term. The resulting rational endpoints are rounded outward to scale S .

Sine and cosine are evaluated by the Taylor polynomials through degrees 121 and 120 respectively, using outward-rounded interval operations. Every argument used for a coefficient lies in $[0, \pi)$: in the power basis of $\mathbb{Z}[\omega]$ it is $j\pi/\nu$ with $0 \leq j < \varphi(2\nu) \leq \nu$. Every arc argument lies in $[0, \pi/(13\nu)]$. All arguments are therefore in $[0, 4]$. Taylor's theorem supplies the common conservative error bound

$$\frac{4^{121}}{121!} < \frac{1}{S}. \quad (15)$$

The displayed integer inequality $4^{121}S < 121!$ is itself asserted in the verifier. One unit is added outward to each Taylor interval, in addition to all rounding enclosures already accumulated by the arithmetic.

For a dyadic parameter interval

$$I = \left[\frac{a\pi}{13\nu 2^d}, \frac{(a+1)\pi}{13\nu 2^d} \right],$$

monotonicity of sine and cosine on $[0, \pi/2]$ gives an enclosing complex interval Z_I for $e^{-i\theta}$, $\theta \in I$. Complex interval Horner evaluation gives an enclosure $Q_b(Z_I)$. The interval is accepted only if either the real or the imaginary component strictly excludes zero. If it is not accepted, it is replaced by its two equal halves. Starting with the full closed arc, every subdivision preserves coverage, including both endpoints. Failure to certify an interval by depth thirty raises an exception; it is never treated as success.

In pseudocode, with every indicated polynomial operation exact, the complete proof check is:

1. Enumerate every nonconstant binary word b of length thirteen.
2. Compute $\nu, B_k, \tilde{P}_b$ and F_ν ; divide by F_ν and assert that the remainder is zero.
3. Remove all exact factors $z - 1$, obtaining Q_b with $Q_b(1) \neq 0$.
4. Enclose its coefficients at the specified embedding $\omega = e^{-\pi i/\nu}$ by the integer interval procedure above.
5. Start with the full interval $[0, \pi/(13\nu)]$. Repeatedly accept an interval whose Horner enclosure excludes zero, or bisect it.
6. Assert that every interval has been accepted. Record the finite partition, its size and its greatest depth.

The supplied verifier `certify_cycle13.py`, which uses only the Python standard library, completes this check. The observed exact certificate sizes are

Enumeration	Words	Accepted intervals	Greatest depth
Dihedral representatives	378	393	3
All nonconstant words	8190	8459	3

The full enumeration is reproduced by

```
python3 certify_cycle13.py --all-words
```

The implementation, including every interval primitive, is short enough to be printed with the paper. The recorded partitions provide an additional check on complete coverage; the verifier also constructs and checks them anew, so trusting a precomputed data file is unnecessary.

Proof of Lemma 4. The preceding geometric reduction proves regularity if only one boundary-side length occurs. Otherwise it gives a nonconstant binary word and a zero of its closure polynomial at $0 < \theta < \pi/(13\nu)$. The factors removed in (13) have no zeros on this open arc. Proposition 5, verified by the finite integer interval certificate, excludes a zero of Q_b there as well. This contradiction proves the lemma. $\square$

7 Reconstructing the smaller six-distance cores

A *distance pattern* on r vertices is a symmetric matrix $M \in \{0, 1, \dots, k\}^{r \times r}$ whose diagonal entries are zero and whose off-diagonal entries are positive, with every color $1, \dots, k$ used. A realization assigns pairwise distinct positive squared distances to the colors and realizes the resulting matrix by distinct points of $\mathbb{R}^2$. Patterns are identified under simultaneous permutations of rows and columns and under arbitrary permutations of their positive colors. In particular, a color label does not encode the numerical order of a distance.

The exhaustive generation in [3, Sections 3.1 and 4, Table 9] leaves fifteen candidate patterns on twelve vertices and sixty-two on eleven vertices, for *at most* six colors. These are counts of patterns passing necessary algebraic conditions, not counts of already classified real configurations. Every planar realization must occur among the candidates. Consequently, exhibiting that many inequivalent realizable patterns reconstructs the entire candidate list. It remains necessary to determine every metric realization of each pattern.

7.1 The twelve-point patterns

Use triangular coordinates

$$(a, b) \mapsto \left(a + \frac{b}{2}, \frac{\sqrt{3}b}{2}\right), \quad Q(a, b) = a^2 + ab + b^2.$$

Table 1 gives eleven twelve-point lattice sets $L_0, \dots, L_{10}$. In its row i , the entry in column B_a means that the points (a, b) with $b \in B_a$ are included. A dash means that B_a is empty. The squared-distance palettes are

$$S_9 = \{1, 3, 4, 7, 9\}, \quad S_{12} = S_9 \cup \{12\}, \quad S_{13} = S_9 \cup \{13\}.$$

Set	B_0	B_1	B_2	B_3	B_4	Palette
L_0	0, 1, 2, 3	-1, 0, 1, 2	0, 1	-1, 0	–	S_{13}
L_1	0, 1, 2, 3	0, 1, 2	-1, 0, 1, 2	0	–	S_{12}
L_2	0, 1, 2, 3	0, 1, 2	0, 1	-1, 0, 1	–	S_{13}
L_3	0, 1, 2	-2, -1, 0, 1	-1, 0, 1	-2, -1	–	S_{13}
L_4	0, 1, 2	-1, 0, 1, 2	-1, 0, 1	-1, 0	–	S_9
L_5	0, 1, 2	-1, 0, 1	-2, -1, 0, 1	-1, 0	–	S_{12}
L_6	0, 1, 2	-1, 0, 1	-1, 0, 1	-2, -1, 0	–	S_{13}
L_7	0, 2	-2, -1, 0, 1	-1, 0, 1	-2, -1	-1	S_{13}
L_8	0, 2	0, 1	-1, 0, 1	-2, -1, 0, 1	-1	S_{13}
L_9	0	-2, -1, 0, 1	-2, -1, 0	-3, -2, 0	-2	S_{12}
L_{10}	0	-2, -1, 0, 1	-2, 0	-3, -2, -1, 0	-2	S_{12}

Table 1: Eleven lattice realizations of twelve-point patterns. Entries in each B_a column list the permitted second coordinates.

The other four patterns have the displayed realizations

$$R_{13} \setminus \{v\}, \quad R_{12}, \quad (R_{12} \setminus \{v\}) \cup \{O\}, \quad R_{11} \cup \{O\},$$

where R_n is a regular n -gon and O its center. The pattern of each displayed set is obtained by assigning one color to each distinct squared distance.

Here is an elementary certificate that the fifteen patterns are inequivalent. For a color c , let e_c be the number of edges of that color. For each vertex v , let $d_c(v)$ be its number of incident edges of color c , including zero values. Form the invariant

$$\mathcal{I}(M) = (\{\{e_c : c\}\}, \quad \{\{\{\{(e_c, d_c(v)) : c\} : v\}\}\},$$

where braces with doubled inner delimiters denote multisets. This is unchanged by permutations of vertices or colors. Direct integer calculation gives fifteen different invariants for the fifteen displayed patterns. Together with Table 9 of [3], this exhausts the candidate patterns on twelve vertices. Distinct invariant values prove inequivalence; no converse assertion about this invariant is needed.

7.2 The eleven-point patterns

Order each L_i lexicographically by (a, b) , with indices $0, 1, \dots, 11$; write v_j for the point of index j . The machine-readable name `Li minus j` means $L_i \setminus \{v_j\}$. Deleting the indices listed in Table 2, one at a time, gives fifty-two lattice patterns with distinct invariants $\mathcal{I}$. The remaining ten circular patterns are given in Table 3. Throughout that table, identify the vertices of R_n with $1, \zeta_n, \dots, \zeta_n^{n-1}$, where $\zeta_n = \exp(2\pi i/n)$; a deleted index a means the vertex ζ_n^a .

Source	Indices deleted, one at a time	Count
L_0	0, 1, 2, 3, 4, 5, 6	7
L_1	0, 1, 4, 5, 7, 8, 11	7
L_2	0, 1, 4, 5, 7, 10	6
L_3	0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11	11
L_4	0, 4	2
L_5	0, 1, 4, 5, 8, 9	6
L_6	4, 5	2
L_7	0, 3, 4	3
L_8	2, 4, 5, 8	4
L_9	2, 3, 6, 7	4
Total		52

Table 2: Representatives of the fifty-two eleven-point lattice patterns.

All sixty-two invariants are distinct. The lattice realizations are checked using the integer form Q ; the circular realizations follow from the chord formulas. Hence the sixty-two patterns exhaust the published candidate count at order eleven. Equivalently, one may generate all one-vertex deletions from the fifteen twelve-point representatives, retain one representative for each of the sixty-one distinct invariants that occur, and adjoin $R_{10} \cup \{O\}$. This gives the same list.

7.3 Exact metric realization checks

The pattern count alone does not classify actual point sets. To determine metrics, take one of the reconstructed r -vertex patterns M with k colors. Normalize its first squared distance to $d_1 = 1$, set $d_0 = 0$, and write $d_i = x_i$ for $2 \leq i \leq k$. For indices $1 \leq i, j \leq r - 1$, define

$$G_{ij} = d_{M_{0i}} + d_{M_{0j}} - d_{M_{ij}}.$$

Pattern	Displayed realization	Cyclotomic multipliers j
C_0	$R_{13} \setminus \{\zeta_{13}^0, \zeta_{13}^{12}\}$	1, 2, 3, 4, 5, 6
C_1	$R_{12} \setminus \{\zeta_{12}^0\}$	1, 5
C_2	$(R_{12} \setminus \{\zeta_{12}^0, \zeta_{12}^{11}\}) \cup \{O\}$	1, 5
C_3	$(R_{12} \setminus \{\zeta_{12}^1, \zeta_{12}^{11}\}) \cup \{O\}$	1, 5
C_4	$(R_{12} \setminus \{\zeta_{12}^2, \zeta_{12}^{11}\}) \cup \{O\}$	1, 5
C_5	$(R_{12} \setminus \{\zeta_{12}^3, \zeta_{12}^{11}\}) \cup \{O\}$	1, 5
C_6	$(R_{12} \setminus \{\zeta_{12}^5, \zeta_{12}^{11}\}) \cup \{O\}$	1, 5
C_7	$(R_{11} \setminus \{\zeta_{11}^0\}) \cup \{O\}$	1, 2, 3, 4, 5
C_8	R_{11}	1, 2, 3, 4, 5
C_9	$R_{10} \cup \{O\}$	1, 3

Table 3: Ten circular eleven-point patterns. Multiplying every retained vertex index by j modulo n gives the cyclotomic realizations to be checked in the metric calculation. Some of these realizations have different shapes despite having equivalent distance patterns.

For a planar realization this is twice the Gram matrix of the vectors based at vertex zero. Thus each principal three-by-three minor vanishes. Only necessity is used: we do not assume that vanishing principal minors suffices for arbitrary complex symmetric matrices to have rank two.

Let I_M be the ideal over $\mathbb{Q}$ generated by these minors. Introduce a variable u and compute a lexicographic Gröbner basis of

$$I_M + \left\langle u \prod_{i=2}^k x_i - 1 \right\rangle \subset \mathbb{Q}[u, x_2, \dots, x_k], \quad u > x_2 > \dots > x_k.$$

Every genuine realization satisfies this system, since none of its pairwise distances is zero. Eliminating u therefore gives necessary polynomial equations for every genuine metric. No floating-point calculation or approximate equality test is involved.

For a lattice pattern, the desired certificate is that the eliminated basis consists precisely of the linear polynomials fixing all normalized squared distances to those of its displayed coordinates. For a circular pattern, the certificate is triangular: each x_i except the last is a polynomial in a final parameter t , with a nonzero constant coefficient on x_i , followed by one univariate polynomial in t .

The circular solutions can be exhausted without numerical root finding. For each multiplier j in Table 3, evaluate

$$d_a(j) = \frac{2 - \zeta_n^{aj} - \zeta_n^{-aj}}{2 - \zeta_n^j - \zeta_n^{-j}}, \quad d_O(j) = \frac{1}{2 - \zeta_n^j - \zeta_n^{-j}}$$

for chord colors and the center color, respectively. All arithmetic is in $\mathbb{Q}[z]/(\Phi_n(z))$. Check every eliminated equation, and check that the resulting values of t are pairwise distinct and that their number equals the degree of its univariate equation. These checks prove that no other complex solution for t exists. Triangularity then fixes all other squared distances. Since a complete squared-distance matrix determines a Euclidean configuration up to isometry, this exhausts all realizations up to similarity.

It is essential to distinguish pattern equivalence from similarity. For example, the six multipliers for C_0 give the six possible separations of two vertices deleted from R_{13} . They share a distance pattern although their point sets need not be similar. For the extension argument, all are covered by allowing the indicated regular-polygon deletion family.

Lemma 6 (Computer-assisted classification). *Every eleven-point planar set with at most six distances is similar either to one of the fifty-two lattice sets in Table 2, or to one of the circular realizations specified by Table 3 and its multipliers.*

Proof. Every such set has a pattern in the published candidate list. The sixty-two reconstructed inequivalent realizable patterns exhaust that list. The exact metric checks described above exhaust the realizations of each pattern. For the fifty-two lattice patterns, the eliminated equations are linear and fix the displayed palettes. For the ten circular patterns, the triangular equations have final degrees 6, 2, 2, 2, 2, 2, 2, 5, 5, 2 in the order of Table 3. The listed multipliers give that many distinct solutions in every case. The verification computes the bases over $\mathbb{Q}$ from the principal minors and verifies the cyclotomic substitutions exactly. $\square$

8 Diameter extensions of eleven-point cores

We next give a finite extension argument for the configurations in Lemma 6. Throughout this section a palette consists of *squared* distances. This convention does not change the number of distinct distances.

Lemma 7. *Let $A \subset X \subset \mathbb{R}^2$, where $|A| = 11$, A determines at most six distances, and X determines at most seven distances. If $\text{diam}(A) < \text{diam}(X)$, then $|X| \leq 16$.*

Proof. Apply the classification in Lemma 6. We treat its fifty-two lattice configurations and ten circular pattern families below. The lattice calculations are unchanged by similarity, so use the coordinates of Tables 1 and 2. Write $D = \text{diam}(X)$ and let S be the squared-distance palette of A . If X has at most six distances, then the published bound $g_2(6) = 13$ suffices. We may therefore assume that X has exactly seven distances.

Nonextreme anchors

Let I be the points of A which are not extreme vertices of its convex hull. No point of I can be an endpoint of a diameter of X . Indeed, if $p = \sum_i \lambda_i a_i$ is a nontrivial convex combination of distinct points of A , strict convexity gives, for every $q \in X$,

$$\|q - p\|^2 < \sum_i \lambda_i \|q - a_i\|^2 \leq D^2.$$

This includes points in the relative interiors of hull sides.

The forty-nine six-distance lattice cores

Here $|S| = 6$, and the sole additional squared distance is $D^2 > \max S$. Every added point $q \in X \setminus A$ must therefore satisfy $\|q - p\|^2 \in S$ for every $p \in I$.

For each of the forty-nine cores the verifier finds three noncollinear points A_0, B_0, C_0 in I . In triangular coordinates put $Q(x, y) = x^2 + xy + y^2$. An assignment

$$Q(q - A_0) = a, \quad Q(q - B_0) = b, \quad Q(q - C_0) = c, \quad (a, b, c) \in S^3,$$

determines q uniquely. To see this, put $w = B_0 - A_0$ and $v = C_0 - A_0$ and subtract the first equation from the other two. The resulting system is

$$\begin{pmatrix} 2w_x + w_y & w_x + 2w_y \\ 2v_x + v_y & v_x + 2v_y \end{pmatrix} \begin{pmatrix} q_x \\ q_y \end{pmatrix} = \begin{pmatrix} Q(B_0) - Q(A_0) + a - b \\ Q(C_0) - Q(A_0) + a - c \end{pmatrix}.$$

Its determinant is nonzero by noncollinearity. Solve it over $\mathbb{Q}$ and retain the solution only if the first original equation holds. The other two then hold as well. Exhausting S^3 thus produces every possible real added point; rationality follows from the equations and is not imposed on X .

Filter the candidates against all of I and A . A retained point either has only old distances to A , or has just one additional squared distance $T > \max S$. The new diameter must occur between a candidate and a core point, or between two candidates. Enumerate these possible values of T , and, for each, keep only candidates compatible with the palette $S \cup \{T\}$. Join two candidates when their mutual squared distance belongs to that palette. The largest admissible extension is a largest clique in this finite graph. Checking the old palette alone also covers extensions with no additional distance.

The three five-distance lattice cores

These are $L_0 \setminus \{v_3\}$, $L_4 \setminus \{v_0\}$, and $L_4 \setminus \{v_4\}$, with the zero-based vertex indices of Table 2. Each has exactly five nonextreme points. Now the palette of X has the form

$$S \cup \{T, D^2\}, \quad D^2 > \max(S \cup \{T\}).$$

For an added point q , its squared distances to I lie in $S \cup \{T\}$. If at most two of these distances equal T , at least three are old, so there are two distinct old-distance anchors. If at least three equal T , the corresponding three points lie on a circle centered at q . They are noncollinear, and q is their circumcenter. Both alternatives give finite lists without first specifying T .

For the first alternative choose distinct $A_0, B_0 \in I$, assign $a, b \in S$, and put

$$w = B_0 - A_0, \quad L = Q(w), \quad t = \frac{a - b + L}{2L}, \quad k = \frac{a - t^2 L}{3L}.$$

If $k < 0$ there is no real solution. Otherwise the two circle intersections are exactly

$$q = A_0 + tw \pm \sqrt{k}(-w_x - 2w_y, 2w_x + w_y). \quad (16)$$

The last vector is perpendicular to w in the triangular-coordinate metric and has squared norm $3L$, which proves the formula. Enumerate (16) for every anchor pair and every assignment in S^2 , and add the circumcenters of every noncollinear triple of I .

The calculation at this step permits irrational points. Write $\sqrt{k} = f\sqrt{d}$, with $f \in \mathbb{Q}$ and d a positive squarefree integer, treating $k = 0$ separately and absorbing $d = 1$ into the rational coordinates. Candidate coordinates then have the form

$$(a_x + b_x\sqrt{d}, a_y + b_y\sqrt{d}), \quad a_x, b_x, a_y, b_y \in \mathbb{Q}.$$

Every squared distance to a core point is represented exactly as $u + v\sqrt{d}$. If $d > 1$, this belongs to $\mathbb{Q}$ precisely when $v = 0$. This representation permits exact comparisons, including equality of two unknown new distances.

Remove points of A and retain a candidate only when it has at most one distance outside S toward I and at most two outside S toward A . The exhaustive calculation verifies that all survivors are rational. Their numbers, in the order above, are 10, 12, 11. Take the pool of squared distances outside S occurring from survivors to A or between survivors. For every pair $a < b$ in this pool with $b > \max S$, use palette $S \cup \{a, b\}$ and discard candidates at squared distance b from any point of I . The remaining compatibility graph again yields the extension maximum by a largest-clique calculation. This covers the actual pair $\{T, D^2\}$ even when one or both values occur only between added points.

Exact calculation and lattice results

The lattice verifier uses rational arithmetic, squarefree extraction of positive integers, and an exact include/exclude clique search. Its search invariant is that every retained candidate is compatible with all points already selected. At each recursive call, branching on one remaining candidate covers both its inclusion and its exclusion; a branch is pruned only when the selected cardinality plus the number of remaining candidates cannot improve the best clique already found. The entire computation can be summarized as follows.

1. Reconstruct each coordinate core from the deletion table and verify that its squared-distance equality pattern is the specified colored matrix.
2. Compute its convex hull and nonextreme set I with integer orientation tests. Form the complete candidate list by the appropriate anchor procedure above, using exact arithmetic.
3. Form the complete pool of possible additional distance values, filter for the largest-distance restriction, and exhaust the resulting compatibility graphs.

The results are

Core type	Number of cores	Upper bound for $ X $
Six distances, first group	32	12
Six distances, second group	17	16
Five distances	3	16

For each five-distance core, cardinality sixteen is attained only in the computed extra-value pair $\{12, 13\}$. The certificate records the palette, nonextreme points, candidate list, extension palettes, and largest-clique witnesses for each core. It is reproduced by

```
python3 core_extensions/eleven_lattice_extensions.py
```

Its final check is that all fifty-two bounds are at most sixteen.

The ten circular cores

The circular configurations in Table 3 are covered by four containment assertions:

- (a) A seven-distance set containing eleven vertices of R_{13} is contained in R_{13} together with its center.
- (b) A seven-distance set containing at least ten vertices of R_{12} is contained in R_{12} together with its center.
- (c) A seven-distance set containing at least ten vertices of R_{11} is contained in R_{11} together with its center.
- (d) A seven-distance set containing R_{10} and its center has no additional points.

Here the regular polygons have circumradius one. Deleting the stated numbers of vertices leaves every chord length of the corresponding polygon: each nondiameter chord length occurs on n pairs and each vertex belongs to two of them, whereas a diameter occurs on $n/2$ disjoint pairs. The removals cannot eliminate all pairs of any length.

We verify the assertions by one common exact procedure. For a noncentral point q compatible with at most seven distances, the eight or more retained polygon vertices cannot all have different

distances from q . An equal-distance pair places q on a reflection axis of the full regular polygon. After rotation, write $q = te$, where e is a unit vector along such an axis and $t \in \mathbb{R} \setminus \{0\}$. Odd polygons have one axis type; even polygons have vertex and edge-midpoint axes. If C_j is twice the projection of a polygon vertex onto e , the distinct squared radii from q to the full polygon are

$$f_j(t) = t^2 + 1 - tC_j.$$

For $t \neq 0$ distinct projections give distinct radii.

For R_{13} there are seven radii, one occurring once and six occurring twice. Removing two vertices leaves at least six radii. As the old palette has six values, at least five retained radii must be old. For R_{12} , vertex axes have seven radii, of which two occur once; edge axes have six radii, all occurring twice. Removing at most two vertices leaves at least five radii, and hence at least four old matches. For R_{11} there are six radii, only one occurring once. Removing at most one vertex leaves at least five, and the five old chord lengths force at least three old matches. Finally, R_{10} has six radii on a vertex axis and five on an edge axis. The core R_{10} together with its center has six old squared distances, including 1, so the respective requirements are five and four old matches.

In every case at least two distinct radii must match two distinct old values. If $f_j(t) = a$ and $f_k(t) = b$, subtraction gives the finite candidate set

$$t = \frac{b - a}{C_j - C_k} \quad (j < k, a \neq b). \quad (17)$$

The denominator is nonzero. All quantities in (17) belong to the corresponding real cyclotomic field. Work in its power basis $1, u, \dots, u^{r-1}$ with the following defining polynomial:

Polygon	u	Minimal polynomial
R_{13}	$2 \cos(2\pi/13)$	$u^6 + u^5 - 5u^4 - 4u^3 + 6u^2 + 3u - 1$
R_{12}	$2 \cos(\pi/12)$	$u^4 - 4u^2 + 1$
R_{11}	$2 \cos(2\pi/11)$	$u^5 + u^4 - 4u^3 - 3u^2 + 3u + 1$
R_{10}	$2 \cos(\pi/10)$	$u^4 - 5u^2 + 5$

These polynomials follow from the respective cyclotomic polynomials by pairing inverse roots; their degrees are half the relevant cyclotomic field degrees. The values $2 \cos(j\theta)$ are generated by $C_0 = 2$, $C_1 = u$, $C_{j+1} = uC_j - C_{j-1}$. For even polygons use even j for vertex axes and odd j for edge axes. Multiplication is polynomial multiplication followed by reduction modulo the displayed polynomial; inverses are computed by exact Gaussian elimination on multiplication matrices. Thus no choice of an approximate real embedding enters an equality check.

Exhausting (17) and counting old-radius matches gives the following complete histograms. An entry $h : c$ means that exactly c parameters match exactly h old values. The old palette for R_{10} includes the center distance; the other rows use the polygon's chord palette.

Polygon	Axis	Parameters	Match histogram
R_{13}	vertex	362	0 : 355, 1 : 6, 6 : 1
R_{12}	vertex	120	0 : 108, 1 : 2, 2 : 8, 6 : 2
R_{12}	edge	92	0 : 86, 3 : 6
R_{11}	vertex	162	0 : 156, 1 : 5, 5 : 1
R_{10}	vertex	122	0 : 110, 1 : 2, 2 : 4, 3 : 4, 5 : 2
R_{10}	edge	72	0 : 68, 1 : 4

The only parameters meeting the required thresholds are $t = 1$ for odd polygons and $t = \pm 1$ for even polygons, all existing vertices of the full polygon. The only possible nonvertex addition is its center,

which was handled separately. This proves assertions (a)–(d), giving upper bounds 14, 13, 12, 11, respectively. The exact computations are reproduced by

`python3 verify_circle_extensions.py`

The verifier uses the retained-subset thresholds justified above. Combining these circular bounds with the fifty-two lattice bounds proves Lemma 7. $\square$

9 Proof of the main theorem

Proof of Theorem 1, upper bound. Suppose X has seventeen points and at most seven distances. It has exactly seven distances because $g_2(6) = 13$. By Lemma 2, $m = |X_D| \leq 13$. If the graph on X_D is not C_m , it has an independent set of size at least $\lceil m/2 \rceil$. Together with the $17 - m$ isolated vertices, this gives an independent set of size at least

$$17 - m + \lceil m/2 \rceil = 17 - \lfloor m/2 \rfloor \geq 11.$$

If the graph on X_D is C_m with $m \leq 11$, then m is odd and the corresponding bound is

$$17 - m + (m - 1)/2 = 17 - (m + 1)/2 \geq 11.$$

In either case choose eleven independent points A . Their distances omit the global diameter, so $|\Delta(A)| \leq 6$ and $\text{diam } A < \text{diam } X$. Lemma 7 gives $|X| \leq 16$, a contradiction.

The only remaining case is $m = 13$ with endpoint graph C_{13} . Lemma 4 makes these points a regular thirteen-gon. Lemma 3 then gives $|X| \leq 14$, again a contradiction. Every larger set contains a seventeen-point subset, so all cardinalities above sixteen are excluded. $\square$

10 An attaining configuration

In triangular coordinates, with $Q(a, b) = a^2 + ab + b^2$, let

$$H = \{(a, b) \in \mathbb{Z}^2 : Q(a, b) \in \{0, 1, 3\}\}, \quad X = H \cup \{(-2, 0), (0, 2), (2, -2)\}.$$

There are sixteen points and exactly seven squared distances,

$$\{1, 3, 4, 7, 9, 12, 13\}.$$

Their multiplicities are 33, 24, 18, 24, 9, 6, 6, respectively, summing to $\binom{16}{2} = 120$. Direct integer calculation verifies these claims. This supplies the lower bound and completes Theorem 1.

Computational provenance and reproduction

The argument depends on the published candidate count in [3] and on the exact computations described here. The new computations were performed and audited using independently written verification routines. In particular, all sixty-two metric elimination bases were recomputed from the original Gram minors in two runs and compared exactly; all lattice compatibility maxima were also independently checked by exhaustive subset enumeration. This is not a formal verification in a proof assistant.

This work is jointly authored by Alon Amit and GPT-6 Astra, an OpenAI AI model. Alon Amit proposed the problem, guided the research, and supplied Budianto's thesis. GPT-6 Astra, working

through OpenAI Codex with parallel research and audit agents, contributed to the literature analysis, mathematical arguments, verification code, computational checks, and manuscript preparation. The computational audits described above were carried out in separate AI-agent sessions; they do not constitute external human peer review. No human review beyond that actually performed is implied by these audits.

The accompanying verification package contains the input coordinates, pattern reconstruction, exact metric and extension checks, cycle verifier, and independent audit records. From its root, with Python 3 and SymPy installed, run

```
python3 verify_all.py
```

Run without optimization: assertions are part of the certificate. The metric stage recomputes the algebra from the Gram minors rather than trusting cached bases. Its expected result is fifty-two forced lattice metrics and ten exhausted circular pattern families. The extension stage checks all fifty-two lattice cores and all circular families. The cycle stage checks all 8190 nonconstant words with 8459 accepted intervals and maximum subdivision depth three. The reference code and input data are printed in Appendix A to preserve reproducibility of the mathematical record without depending on a mutable external URL.

References

- [1] P. Erdős and P. Fishburn, Maximum planar sets that determine k distances, *Discrete Mathematics* **160** (1996), 115–125. doi:10.1016/0012-365X(95)00153-N.
- [2] X. Wei, A proof of Erdős–Fishburn’s conjecture for $g(6) = 13$, *Electronic Journal of Combinatorics* **19**(4) (2012), P38. doi:10.37236/2917.
- [3] F. Szöllősi and P. R. J. Östergård, Constructions of maximum few-distance sets in Euclidean spaces, *Electronic Journal of Combinatorics* **27**(1) (2020), P1.23. doi:10.37236/8565.
- [4] W. T. Budianto, *An Observation on 7-distance Set in Euclidean Plane*, master’s thesis, Department of Mathematics, National Central University, Taiwan, June 2022. <https://ir.lib.ncu.edu.tw/handle/987654321/88394>.
- [5] L.-R. Bao and W.-H. Yu, Constructions of Large m -Distance Sets on Triangular Lattice, arXiv:2509.00880v1 (2025). <https://arxiv.org/abs/2509.00880>.

A Reference verification code and data

The following files form the reference verification package. File names at the beginning of each listing give the paths relative to its root. The input data are explicit finite mathematical objects; all proof checks are recomputed from them. The source distribution also includes independent audit implementations and their records.

`verify_all.py`

```
1 """Recompute the certificates for the planar seven-distance bound."""
2 import sys, subprocess, json
3 from pathlib import Path
4 if not __debug__: raise RuntimeError("Run without -O: assertions are proof checks")
```

```

5 import sympy
6 print("Python",sys.version,"SymPy",sympy.__version__,flush=True)
7 steps=[
8     "computational_strategy/twelve_core_patterns.py",
9     "seventeen_attack/reconstruct_eleven.py",
10    "core_realizations/audit_eleven.py",
11    "core_realizations/verify_eleven_cyclic.py",
12    "core_extensions/eleven_lattice_extensions.py",
13    "verify_circle_extensions.py",
14 ]
15 for name in steps:
16     print("VERIFY",name,flush=True)
17     subprocess.run([sys.executable,name],check=True)
18 subprocess.run([sys.executable,"algebraic_bound/certify_cycle13.py","--all-words"],check=True)
19 c=json.loads(Path("algebraic_bound/cycle13_allwords_certificate.json").read_text())
20 assert (c["words"],c["total_intervals"],c["max_depth"])==(8190,8459,3)
21 Q=lambda a,b:a*a+a*b+b*b
22 H=[(a,b) for a in range(-3,4) for b in range(-3,4) if Q(a,b) in (0,1,3)]
23 X=H+[(-2,0),(0,2),(2,-2)]
24 assert len(set(X))==16
25 from collections import Counter
26 counts=Counter(Q(a-c,b-d) for i,(a,b) in enumerate(X) for c,d in X[:i])
27 assert dict(counts)=={1:33,3:24,4:18,7:24,9:9,12:6,13:6}
28 print("PASS: all computational certificates for g_2(7)=16",flush=True)

```

computational_strategy/twelve_lattice_cores.json

```

1 [
2 {"index":0,"points":[[0,0],[0,1],[0,2],[0,3],[1,-1],[1,0],[1,1],[1,2],[2,0],[2,1],[3,-1],[3,0]],"
   squared_distances":[1,3,4,7,9,13]},
3 {"index":1,"points":[[0,0],[0,1],[0,2],[0,3],[1,0],[1,1],[1,2],[2,-1],[2,0],[2,1],[2,2],[3,0]],"
   squared_distances":[1,3,4,7,9,12]},
4 {"index":2,"points":[[0,0],[0,1],[0,2],[0,3],[1,0],[1,1],[1,2],[2,0],[2,1],[3,-1],[3,0],[3,1]],"
   squared_distances":[1,3,4,7,9,13]},
5 {"index":3,"points":[[0,0],[0,1],[0,2],[1,-2],[1,-1],[1,0],[1,1],[2,-1],[2,0],[2,1],[3,-2],[3,-1]],"
   squared_distances":[1,3,4,7,9,13]},
6 {"index":4,"points":[[0,0],[0,1],[0,2],[1,-1],[1,0],[1,1],[1,2],[2,-1],[2,0],[2,1],[3,-1],[3,0]],"
   squared_distances":[1,3,4,7,9]},
7 {"index":5,"points":[[0,0],[0,1],[0,2],[1,-1],[1,0],[1,1],[2,-2],[2,-1],[2,0],[2,1],[3,-1],[3,0]],"
   squared_distances":[1,3,4,7,9,12]},
8 {"index":6,"points":[[0,0],[0,1],[0,2],[1,-1],[1,0],[1,1],[2,-1],[2,0],[2,1],[3,-2],[3,-1],[3,0]],"
   squared_distances":[1,3,4,7,9,13]},
9 {"index":7,"points":[[0,0],[0,2],[1,-2],[1,-1],[1,0],[1,1],[2,-1],[2,0],[2,1],[3,-2],[3,-1],[4,-1]],"
   squared_distances":[1,3,4,7,9,13]},
10 {"index":8,"points":[[0,0],[0,2],[1,0],[1,1],[2,-1],[2,0],[2,1],[3,-2],[3,-1],[3,0],[3,1],[4,-1]],"
   squared_distances":[1,3,4,7,9,13]},
11 {"index":9,"points":[[0,0],[1,-2],[1,-1],[1,0],[1,1],[2,-2],[2,-1],[2,0],[3,-3],[3,-2],[3,0],[4,-2]],"
   squared_distances":[1,3,4,7,9,12]},
12 {"index":10,"points":[[0,0],[1,-2],[1,-1],[1,0],[1,1],[2,-2],[2,0],[3,-3],[3,-2],[3,-1],[3,0],[4,-2]],"
   squared_distances":[1,3,4,7,9,12]}
13 ]

```

computational_strategy/twelve_core_patterns.py

```

1 import json
2 from collections import Counter
3 from itertools import combinations
4
5 rows=json.load(open('computational_strategy/twelve_lattice_cores.json'))
6 patterns=[]
7 for row in rows:
8     pts=row['points']; D=row['squared_distances'];lookup={d:i+1 for i,d in enumerate(D)}
9     M=[[0]*12 for _ in range(12)]

```

```

10 for i in range(12):
11     for j in range(i):
12         a=pts[i][0]-pts[j][0];b=pts[i][1]-pts[j][1]
13         M[i][j]=M[j][i]=lookup[a*a+b*b]
14     patterns.append(dict(name='L'+str(row['index']),matrix=M,points=pts,palette=D))
15 for name,n,verts,center in [('R13minus1',13,list(range(12)),False),('R12',12,list(range(12)),False),('
    R12minus1_plus0',12,list(range(11)),True),('R11plus0',11,list(range(11)),True)]:
16     M=[[0]*12 for _ in range(12)]
17     for i in range(len(verts)):
18         for j in range(i):
19             delta=abs(verts[i]-verts[j]);M[i][j]=M[j][i]=min(delta,n-delta)
20     if center:
21         color=2 if n==12 else 6
22         for i in range(11):M[i][11]=M[11][i]=color
23     patterns.append(dict(name=name,matrix=M))
24
25 for pattern in patterns:
26     M=pattern['matrix'];colorcounts=Counter(M[i][j] for i in range(12) for j in range(i))
27     signature=tuple(sorted(colorcounts.values()))
28     vertexsig=[]
29     for i in range(12):
30         vc=Counter(M[i][j] for j in range(12) if i!=j)
31         vertexsig.append(tuple(sorted((colorcounts[c],vc[c]) for c in colorcounts)))
32     invariant=(signature,tuple(sorted(vertexsig)))
33     pattern['edge_count_signature']=signature
34     pattern['vertex_signature']=sorted(vertexsig)
35     print(pattern['name'],signature,'vertexsig distinct',len(set(vertexsig)))
36     # Invariants are unchanged by both vertex permutations and arbitrary color relabeling.
37     invariants=[(p['edge_count_signature'],tuple(p['vertex_signature'])) for p in patterns]
38     assert len(patterns)==15 and len(set(invariants))==15
39     assert all(len(set(p.get('palette',[])))<=6 for p in patterns)
40     print('Distinct simple invariants:',len(set(invariants)), 'outof', len(patterns))
41     for i,j in combinations(range(len(patterns)),2):
42         if invariants[i]==invariants[j]:print('Need fuller isomorphism check',patterns[i]['name'],patterns[j]['name']
            ])
43 with open('computational_strategy/twelve_core_patterns.json','w') as f:json.dump(patterns,f,indent=2)

```

seventeen_attack/reconstruct_eleven.py

```

1 """Reconstruct62pairwise inequivalent realized distance patterns on11points.
2 Only existence and distinctness are proved here. Completeness uses the published
3 Table9candidatecount; metricrealization rigidity is a separate calculation.
4 """
5 import json
6 from collections import Counter
7 from pathlib import Path
8 if not __debug__:raise RuntimeError('Run without -O')
9 rows=json.loads(Path('computational_strategy/twelve_core_patterns.json').read_text())
10
11 def invariant(M):
12     colors=set(sum(M,[]))-{0}
13     ec=Counter(M[i][j] for i in range(len(M)) for j in range(i))
14     # The unordered globalcolorfrequencies and unordered vertexcolorfrequencies
15     # are preserved by arbitrary vertex and color relabeling.
16     vs=[tuple(sorted((ec[c],sum(x==c for x in row)) for c in colors)) for row in M]
17     return tuple(sorted(ec.values()),tuple(sorted(vs)))
18
19 patterns={}
20 for row in rows:
21     M0=row['matrix']
22     if 'points' in row:
23         pts=row['points'];palette=row['palette']
24         for i in range(12):
25             for j in range(i):
26                 a,b=pts[i][0]-pts[j][0],pts[i][1]-pts[j][1]
27                 assert a*a+b*b==palette[M0[i][j]-1]

```

```

28     for removed in range(12):
29         keep=[i for i in range(12) if i!=removed]
30         M=[[M0[i][j] for j in keep] for i in keep]
31         key=invariant(M)
32         if key not in patterns:
33             item=dict(name=row['name']+'minus'+str(removed),matrix=M,source=row['name'],deleted=removed)
34             if 'points' in row:item['points']=[row['points'][i] for i in keep];item['source_palette']=row['
palette']
35             patterns[key]=item
36 assert len(patterns)==61
37 M=[[0]*11 for _ in range(11)]
38 for i in range(10):
39     for j in range(i):M[i][j]=M[j][i]=min(i-j,10-i+j)
40     M[i][10]=M[10][i]=6
41 assert invariant(M) not in patterns
42 patterns[invariant(M)]=dict(name='R10plus0',matrix=M)
43 assert len(patterns)==62
44 out=list(patterns.values())
45 assert sum('points' in r for r in out)==52
46 Path('seventeen_attack/eleven_patterns.json').write_text(json.dumps(out,indent=2))
47 print('PASS:62distinct realized coloredpatterns,52lattice and10circular')

```

core_realizations/audit_eleven.py

```

1  """Fresh exact audit of all eleven-point pattern realizations.
2  Uses explicit 3x3 determinant formulas, not saved bases.
3  """
4  import sympy as s,json,time
5  from itertools import combinations
6  from collections import Counter
7  from pathlib import Path
8  if not __debug__: raise RuntimeError('Run without -O')
9  rows=json.load(open('seventeen_attack/eleven_patterns.json'))
10 assert len(rows)==62 and sum('points' in r for r in rows)==52
11 assert sum('points' not in r for r in rows)==10
12 signatures=[]
13 for row in rows:
14     M=row['matrix'];assert len(M)==11 and all(len(r)==11 for r in M)
15     colors=set(sum(M,[]))-{0};assert len(colors)<=6
16     assert all(M[i][j]==M[j][i] and (M[i][j]==0)==(i==j) for i in range(11) for j in range(11))
17     ec=Counter(M[i][j] for i,j in combinations(range(11),2))
18     signatures.append((tuple(sorted(ec.values()))),tuple(sorted((ec[c],sum(x==c for x in r)) for c in
        colors)) for r in M)))
19 if 'points' in row:
20     pts=row['points'];palette=row['source_palette'];assert len(pts)==11 and len(set(map(tuple,pts)))==11
21     for i,j in combinations(range(11),2):
22         a,b=pts[i];c,d=pts[j]
23         assert (a-c)**2+(a-c)*(b-d)+(b-d)**2==palette[M[i][j]-1]
24 assert len(set(signatures))==62
25 print('PASS 62 distinct patterns and 52 exact lattice realizations',flush=True)
26 outdir=Path('core_realizations/eleven');outdir.mkdir(exist_ok=True)
27 completed=0
28 for ix,row in enumerate(rows):
29     name=row['name'];out=outdir/(name+'.json')
30     M=row['matrix'];colors=sorted(set(sum(M,[]))-{0});k=len(colors)
31     xs=s.symbols('x2:'+str(k+1));ds=dict(zip(colors,[s.Integer(1)]+list(xs)));ds[0]=s.Integer(0)
32     G=[[ds[M[0][i+1]]+ds[M[0][j+1]]-ds[M[i+1][j+1]] for j in range(10)] for i in range(10)]
33     polys=[]
34     for i,j,l in combinations(range(10),3):
35         a,b,c=G[i][i],G[j][j],G[l][l];d,e,f=G[i][j],G[i][l],G[j][l]
36         polys.append(s.expand(a*b*c+2*d*e*f-a*f*f-b*e*e-c*d*d))
37     ps=sorted(set(polys))-{0},key=str
38     start=time.time();print(ix+1,name,'BEGIN',len(ps),flush=True)
39     gb=s.groebner(ps,*xs,order='grevlex')
40     u=s.Symbol('u');sat=s.groebner([p.as_expr() for p in gb.polys]+[u*s.prod(xs)-1],u,*xs,order='lex')
41     eliminated=[p.as_expr() for p in sat.polys if not p.as_expr().has(u)]

```

```

42 output={'name':name,'variables':[str(x) for x in xs],'original_colors':colors,'saturated':[str(p) for p in
    eliminated]}
43 out.write_text(json.dumps(output,indent=2))
44 if 'points' in row:
45     old=row['source_palette'];palette=[s.Rational(old[c-1],old[colors[0]-1]) for c in colors]
46     target=[v-q for v,q in zip(xs,palette[1:])]
47     if len(eliminated)!=len(target) or any(s.expand(p-q)!=0 for p,q in zip(eliminated,target)):
48         raise RuntimeError(('NONLINEAR OR ALTERNATIVE LATTICE SOLUTIONS',name,eliminated))
49     else:print('UNIQUE LATTICE',name,flush=True)
50     completed+=1
51     print(ix+1,name,'DONE',round(time.time()-start,2),flush=True)
52
53 assert completed==62
54 print('PASS all 62 fresh exact Gram-minor computations',flush=True)

```

core_realizations/verify_eleven_cyclic.py

```

1 """Exhaust all metric solutions of the ten circular eleven-point patterns."""
2 import sympy as s,json,math
3 from pathlib import Path
4 if not __debug__:raise RuntimeError('Run without -O')
5 z=s.Symbol('z')
6 rows=json.load(open('seventeen_attack/eleven_patterns.json'))
7 for row in rows:
8     if 'points' in row:continue
9     name=row['name'];n=int(name[1:3]);center=(n==10 or n==11 and name!='R11plus0minus11')
10    phi=s.Poly(s.cyclotomic_poly(n,z),z,domain=s.QQ)
11    def red(x):return s.rem(s.Poly(x,z,domain=s.QQ),phi).as_expr()
12    def chord(k):return red(2-z**(k%n)-z**((-k)%n))
13    d=json.loads((Path('core_realizations/eleven')/(name+'.json')).read_text())
14    colors=d['original_colors'];xs=s.symbols(' '.join(d['variables']));polys=[s.Poly(s.sympify(p),*xs) for p in d
        ['saturated']]
15    assert colors[0]==1
16    # Triangular equations ensure exactly one full distance assignment per final root.
17    assert len(polys)==len(xs)
18    t=xs[-1]
19    for v,p in zip(xs[:-1],polys[:-1]):
20        e=p.as_expr();assert e.free_symbols<=set([v,t]) and s.diff(e,v).is_Rational and s.diff(e,v)!=0
21    assert polys[-1].as_expr().free_symbols=={t}
22    # Reconstruct the actual retained polygon labels, with None for the center.
23    if n==13: labels=list(range(12))
24    elif n==12 and row.get('source')== 'R12': labels=list(range(12))
25    elif n==12: labels=list(range(11))+[None]
26    elif n==11: labels=list(range(11))+[None]
27    else: labels=list(range(10))+[None]
28    if 'deleted' in row: labels.pop(row['deleted'])
29    assert len(labels)==11
30    js=[j for j in range(1,n//2+1) if math.gcd(j,n)==1];seen=[]
31    for j in js:
32        inv=s.invert(chord(j),phi.as_expr(),z)
33        vals=[red(inv if center and c==6 else chord(c*j)*inv) for c in colors[1:]]
34        # Every exhibited tuple realizes the original colored distance matrix.
35        values={0:s.Integer(0),colors[0]:s.Integer(1)}
36        values.update(zip(colors[1:],vals))
37        for a in range(11):
38            for b in range(a):
39                raw=s.Integer(1) if labels[a] is None or labels[b] is None else chord((labels[a]-labels[b])*j)
40                assert red(raw*inv-values[row['matrix'][a][b]])==0,(name,j,a,b)
41        for p in polys:
42            result=0
43            for powers,coeff in p.terms():
44                term=coeff
45                for val,power in zip(vals,powers):term=red(term*red(val**power))
46            result=red(result+term)
47            assert result==0,(name,j,p,result)
48        seen.append(vals[-1])

```

```

49 assert len(set(seen))==len(js)
50 assert s.degree(polys[-1].as_expr(),t)==len(js)
51 print(name,': exactly',len(js),'solutions; all relabeled/scaled regular-polygon subsets',flush=True)
52 print('PASS all ten circular eleven-point metric classifications')

```

core_extensions/lattice_diameter_extensions.py

```

1  #!/usr/bin/env python3
2  """Exact exhaustive one-new-largest-distance extensions of ten twelve-point cores.
3  All coordinates are triangular lattice coordinates, but candidate coordinates
4  are arbitrary rational solutions obtained from distances to three nonextreme
5  anchors. A geometric proof guarantees these distances belong to the old palette.
6  """
7  from fractions import Fraction as F
8  from itertools import combinations,product
9  import json
10 from pathlib import Path
11 if not __debug__: raise RuntimeError('Assertions required; run without -O.')
12 def Q(p):return p[0]*p[0]+p[0]*p[1]+p[1]*p[1]
13 def d(p,q):return Q((p[0]-q[0],p[1]-q[1]))
14 def cross(a,b,c):return(b[0]-a[0])*(c[1]-a[1])-(b[1]-a[1])*(c[0]-a[0])
15 def hull(P):
16     H=[]
17     for p in sorted(P):
18         while len(H)>1 and cross(H[-2],H[-1],p)<=0:H.pop()
19     H.append(p)
20     L=len(H)
21     for p in sorted(P,reverse=True)[1:]:
22         while len(H)>L and cross(H[-2],H[-1],p)<=0:H.pop()
23     H.append(p)
24     return H[:-1]
25 def anchored(A,B,C,S):
26     l1=2*(B[0]-A[0])+B[1]-A[1];m1=B[0]-A[0]+2*(B[1]-A[1])
27     l2=2*(C[0]-A[0])+C[1]-A[1];m2=C[0]-A[0]+2*(C[1]-A[1])
28     det=l1*m2-l2*m1;assert det
29     out=set()
30     for a,b,c in product(S,repeat=3):
31         r1=Q(B)-Q(A)+a-b;r2=Q(C)-Q(A)+a-c
32         p=(F(r1*m2-r2*m1,det),F(l1*r2-l2*r1,det))
33         if d(p,A)==a:
34             assert d(p,B)==b and d(p,C)==c
35             out.add(p)
36     return out
37 def clique(P,S):
38     best=[]
39     def go(picked,remaining):
40         nonlocal best
41         if len(picked)+len(remaining)<=len(best):return
42         if len(picked)>len(best):best=picked
43         while remaining:
44             p=remaining.pop()
45             go(picked+[p],[q for q in remaining if d(p,q) in S])
46     go([],P[:]);return best
47 rows=json.loads(Path('computational_strategy/twelve_lattice_cores.json').read_text())
48 cert=[]
49 for row in rows:
50     S=set(row['squared_distances'])
51     if len(S)!=6:continue
52     P=list(map(tuple,row['points']));H=hull(P);I=[p for p in P if p not in H]
53     anchors=next(t for t in combinations(I,3) if cross(*t))
54     C=anchored(*anchors,S)-set(P)
55     C={p for p in C if all(d(p,a) in S for a in I)}
56     new={p:{d(p,a) for a in P}-S for p in C}
57     C=sorted(p for p in C if len(new[p])<=1 and all(t>max(S) for t in new[p]))
58     ts={t for p in C for t in new[p]}|{d(p,q) for p,q in combinations(C,2) if d(p,q)>max(S)}
59     # Include old-only extension case as an independent guard.

```

```

60 cases=[]
61 for t in [None]+sorted(ts):
62     T=S if t is None else S[{t}
63     V=[p for p in C if all(d(p,a) in T for a in P)]
64     best=clique(V,T)
65     cases.append(dict(T=str(t),vertices=[[str(x) for x in p] for p in V],max_added=len(best),witness=[[str(x)
        for x in p] for p in best]))
66 maxn=12+max(c['max_added'] for c in cases)
67 print(row['index'], 'anchors', anchors, 'candidates', len(C), 'Ts', list(map(str, sorted(ts))), 'max_total', maxn,
        flush=True)
68 cert.append(dict(index=row['index'], palette=sorted(S), hull=H, nonextreme=I, anchors=anchors, cases=cases,
        max_total=maxn))
69 assert maxn<=16
70 Path('core_extensions/lattice_diameter_extensions.json').write_text(json.dumps(cert, indent=2))
71 print('PASS: all ten six-distance lattice cores have at most sixteen points in a one-new-largest-distance
        extension.')

```

core_extensions/eleven_lattice_extensions.py

```

1  #!/usr/bin/env python3
2  """Exact largest-diameter extensions of the 52 reconstructed eleven-point lattice cores.
3  Five-distance cores use exact quadratic circle intersections, not assumed rationality.
4  """
5  from fractions import Fraction as F
6  from itertools import combinations, product
7  from pathlib import Path
8  import json
9  if not __debug__: raise RuntimeError('Assertions required')
10 exec(Path('core_extensions/lattice_diameter_extensions.py').read_text().split('rows=json.loads')[0])
11 #  $q=(ax+bx\sqrt{D}, ay+by\sqrt{D})$ ;  $D$  is zero or positive squarefree integer.
12 def sqrt_normalize(k):
13     assert k>=0
14     if not k: return F(0), 0
15     n=k.numerator*k.denominator; den=k.denominator; f=1; D=1; p=2
16     while p*p<=n:
17         e=0
18         while n%p==0: e+=1; n//=p
19         f*=p**(e//2)
20         if e%2: D*=p
21         p+=1
22     if n>1: D*=n
23     return F(f, den), D
24
25 def pq(x,y): return (F(x), F(0), F(y), F(0), 0)
26 def qdist(q,p):
27     ax,bx,ay,by,D=q; ax-=p[0]; ay-=p[1]
28     u=ax*ax+ax*ay+ay*ay+D*(bx*bx+bx*by+by*by)
29     v=2*ax*bx+ax*by+ay*bx+2*ay*by
30     return (u,v,D if v else 0)
31 def old(v,S): return v[1]==0 and v[0] in S
32 def pair_candidates(A,B,S):
33     w=(B[0]-A[0], B[1]-A[1]); L=Q(w); out=set()
34     for a,b in product(S, repeat=2):
35         t=F(a-b+L, 2*L); k=F(a-t*t*L, 3*L)
36         if k<0: continue
37         f,D=sqrt_normalize(k)
38         for sig in [-1,1]:
39             D=D0
40             ax=F(A[0])+t*w[0]; ay=F(A[1])+t*w[1]
41             bx=sig*f*(-w[0]-2*w[1]); by=sig*f*(2*w[0]+w[1])
42             if D==1: ax+=bx; ay+=by; bx=by=F(0); D=0
43             q=(ax,bx,ay,by,D)
44             assert qdist(q,A)==(F(a), F(0), 0) and qdist(q,B)==(F(b), F(0), 0)
45             out.add(q)
46     return out
47

```

```

48 def circumcenter(A,B,C):
49     l1=2*(B[0]-A[0])+B[1]-A[1];m1=B[0]-A[0]+2*(B[1]-A[1])
50     l2=2*(C[0]-A[0])+C[1]-A[1];m2=C[0]-A[0]+2*(C[1]-A[1]);det=l1*m2-l2*m1
51     assert det
52     r1=Q(B)-Q(A);r2=Q(C)-Q(A)
53     return pq(F(r1*m2-r2*m1,det),F(l1*r2-l2*r1,det))
54
55 base={x['name']:x for x in json.loads(Path('computational_strategy/twelve_core_patterns.json').read_text())}
56 results=[]
57 for row in json.loads(Path('seventeen_attack/eleven_patterns.json').read_text()):
58     if not row['name'].startswith('L'):continue
59     P=[tuple(p) for i,p in enumerate(base[row['source']]['points']) if i!=row['deleted']]
60     S={d(a,b) for a,b in combinations(P,2)};I=sorted(set(P)-set(hull(P)))
61     assert len(P)==11 and len(S) in [5,6]
62     label_to_distance={}
63     for i,j in combinations(range(11),2):
64         label=row['matrix'][i][j];value=d(P[i],P[j])
65         assert label not in label_to_distance or label_to_distance[label]==value
66         label_to_distance[label]=value
67     assert len(set(label_to_distance.values()))==len(label_to_distance)==len(S)
68     if len(S)==6:
69         anchors=next(t for t in combinations(I,3) if cross(*t))
70         C=anchored(*anchors,S)-set(P)
71         C={p for p in C if all(d(p,a) in S for a in I)}
72         new={p:{d(p,a) for a in P}-S for p in C}
73         C=sorted(p for p in C if len(new[p])<=1 and all(t>max(S) for t in new[p]))
74         ts={t for p in C for t in new[p]}|{d(p,q) for p,q in combinations(C,2) if d(p,q)>max(S)}
75         cases=[]
76         for t in [None]+sorted(ts):
77             T=S if t is None else S|{t}
78             V=[p for p in C if all(d(p,a) in T for a in P)]
79             best=clique(V,T)
80             cases.append(dict(extras=[str(t)],max_total=11+len(best),witness=[[str(x) for x in p] for p in best]))
81     else:
82         assert len(I)>=5
83         C=set()
84         for A,B in combinations(I,2):C|=pair_candidates(A,B,S)
85         for t in combinations(I,3):
86             if cross(*t):C.add(circumcenter(*t))
87         C-={pq(*p) for p in P}
88         C={q for q in C if len({v for p in I if not old(v:=qdist(q,p),S)})<=1 and len({v for p in P if not old(v:=
            qdist(q,p),S)})<=2}
89         # Verify, rather than assume, all surviving candidates are rational.
90         assert all(q[1]==q[3]==0 for q in C),(row['name'],'nonrational survivors',[q for q in C if q[1] or q[3]])
91         C=sorted((q[0],q[2]) for q in C)
92         values={d(q,p) for q in C for p in P}-S
93         values|={d(a,b) for a,b in combinations(C,2)}-S
94         cases=[]
95         for a,b in combinations(sorted(values),2):
96             if b<=max(S):continue
97             T=S|{a,b}
98             V=[q for q in C if all(d(q,p) in T for p in P) and all(d(q,p)!=b for p in I)]
99             best=clique(V,T)
100             cases.append(dict(extras=[str(a),str(b)],max_total=11+len(best),witness=[[str(x) for x in q] for q in best])
101             )
102             # Extensions with fewer than 7 distances have at most 13 points by published theorem.
103             cases.append(dict(extras=['at most one'],max_total=13,witness=[]))
104             maximum=max(c['max_total'] for c in cases)
105             print(row['name'],'old',len(S),'interior',len(I),'candidates',len(C),'max',maximum,flush=True)
106             results.append(dict(name=row['name'],points=P,palette=sorted(S),nonextreme=I,candidates=[[str(x) for x in p]
107                 for p in C],max_total=maximum,cases=cases))
108             assert maximum<=16,(row['name'],maximum)
109 Path('core_extensions/eleven_lattice_extensions.json').write_text(json.dumps(results,indent=2))
110 print('PASS: all',len(results),'eleven-point lattice cores extend to at most 16 points.')

```

verify_circle_extensions.py

```

1 """Exact axis certificates for all circular eleven-point core families."""
2 from fractions import Fraction as F
3 from itertools import combinations, permutations
4 from collections import Counter
5 if not __debug__: raise RuntimeError('Run without -O')
6 class Field:
7     def __init__(self, mod):
8         self.mod=tuple(map(F, mod)); self.r=len(mod)-1
9         self.zero=(F(0),)*self.r; self.one=self.scalar(1)
10        self.u=(F(0), F(1))+(F(0),)*(self.r-2)
11    def scalar(self, a): return (F(a),)*(self.r-1)
12    def add(self, a, b): return tuple(x+y for x, y in zip(a, b))
13    def sub(self, a, b): return tuple(x-y for x, y in zip(a, b))
14    def neg(self, a): return tuple(-x for x in a)
15    def mul(self, a, b):
16        c=[F(0)]*(2*self.r-1)
17        for i, x in enumerate(a):
18            for j, y in enumerate(b): c[i+j]+=x*y
19        for i in range(len(c)-1, self.r-1, -1):
20            for j in range(self.r): c[i-self.r+j]-=c[i]*self.mod[j]
21        return tuple(c[:self.r])
22    def inverse(self, a):
23        assert a!=self.zero
24        columns=[self.mul(a, tuple(F(i==j) for i in range(self.r))) for j in range(self.r)]
25        M=[[columns[j][i] for j in range(self.r)]+[F(i==0)] for i in range(self.r)]
26        for j in range(self.r):
27            pivot=next(i for i in range(j, self.r) if M[i][j]); M[j], M[pivot]=M[pivot], M[j]
28            d=M[j][j]; M[j]=[x/d for x in M[j]]
29            for i in range(self.r):
30                if i!=j:
31                    d=M[i][j]; M[i]=[x-d*y for x, y in zip(M[i], M[j])]
32        result=tuple(M[i][-1] for i in range(self.r))
33        assert self.mul(a, result)==self.one
34        return result
35    # Coefficients of the monic minimal polynomial, constant term first.
36    cases=[(13, (-1, 3, 6, -4, -5, 1, 1)), (12, (1, 0, -4, 0, 1)),
37           (11, (1, 3, -3, -4, 1, 1)), (10, (5, 0, -5, 0, 1))]
38    expected={ (13, 'vertex'): {0: 355, 1: 6, 6: 1},
39              (12, 'vertex'): {0: 108, 1: 2, 2: 8, 6: 2}, (12, 'edge'): {0: 86, 3: 6},
40              (11, 'vertex'): {0: 156, 1: 5, 5: 1},
41              (10, 'vertex'): {0: 110, 1: 2, 2: 4, 3: 4, 5: 2}, (10, 'edge'): {0: 68, 1: 4}}
42    for n, mod in cases:
43        K=Field(mod); period=n if n%2 else 2*n
44        C=[K.scalar(2), K.u]
45        for j in range(2, period+1): C.append(K.sub(K.mul(K.u, C[-1]), C[-2]))
46        assert C[period]==K.scalar(2)
47        old_indices=range(1, n//2+1) if n%2 else range(2, n+1, 2)
48        OLD=[K.sub(K.scalar(2), C[j]) for j in old_indices]
49        if n==10: OLD.add(K.one)
50        axes=[('vertex', list(range(n//2+1)))] if n%2 else [
51            ('vertex', list(range(0, n+1, 2))), ('edge', list(range(1, n, 2)))]
52        for axis, indices in axes:
53            parameters=set()
54            for j, k in combinations(indices, 2):
55                inv=K.inverse(K.sub(C[j], C[k]))
56                for a, b in permutations(OLD, 2): parameters.add(K.mul(K.sub(b, a), inv))
57            assert K.zero not in parameters
58            threshold=5 if n==13 or (n==10 and axis=='vertex') else (3 if n==11 else 4)
59            histogram=Counter(); survivors=set()
60            for t in parameters:
61                radii=[K.sub(K.add(K.mul(t, t), K.one), K.mul(t, C[j])) for j in indices]
62                assert len(radii)==len(indices)
63                overlap=len(radii&OLD); histogram[overlap]+=1
64                if overlap>=threshold: survivors.add(t)
65            assert dict(histogram)==expected[n, axis]

```

```

66     allowed=set() if axis=='edge' else ({K.one} if n%2 else {K.one,K.neg(K.one)})
67     assert survivors==allowed
68     print('PASS R'+str(n),axis,'parameters',len(parameters),'histogram',dict(sorted(histogram.items())))
69 print('PASS: all circular core families have only polygon vertices or center as possible points')

```

algebraic_bound/certify_cycle13.py

```

1  #!/usr/bin/env python3
2  """Exact integer-arithmetic certificate: nonconstant binary-angle diameter C13 cannot close.
3  No external dependencies. Fixed-point intervals round every product/division outward.
4  """
5  from fractions import Fraction
6  from math import factorial
7  from pathlib import Path
8  from functools import lru_cache
9  import json,time,sys
10 if not __debug__:
11     raise RuntimeError("Run without -O: proof checks require assertions.")
12 N=13
13 S=10**50
14 assert 4**121*S < factorial(121)
15
16 def trim(a):
17     while len(a)>1 and a[-1]==0:a.pop()
18     return a
19
20 def idiv(a,b):
21     a=a[:];q=[0]*max(1,len(a)-len(b)+1)
22     for k in range(len(a)-len(b),-1,-1):
23         assert b[-1]==1
24         q[k]=a[k+len(b)-1]
25         for j,x in enumerate(b):a[k+j]-=q[k]*x
26     assert all(x==0 for x in a)
27     return trim(q)
28 cyclo={}
29 for n in range(1,25):
30     p=[-1]+[0]*(n-1)+[1]
31     for d in range(1,n):
32         if n%d==0:p=idiv(p,cyclo[d])
33     cyclo[n]=p
34
35 def atan_bounds(inv, terms=100):
36     p=Fraction(0)
37     for k in range(terms):p+=Fraction((-1)**k,(2*k+1)*inv**(2*k+1))
38     e=Fraction(1,(2*terms+1)*inv**(2*terms+1))
39     return (p,p+e) if terms%2==0 else (p-e,p)
40 a,b=atan_bounds(5);c,d=atan_bounds(239)
41 pi_lo,pi_hi=16*a-4*d,16*b-4*c
42 PI=(pi_lo.numerator*S//pi_lo.denominator,-((-pi_hi.numerator*S)//pi_hi.denominator))
43 assert 3*S<PI[0]<=PI[1]<4*S
44
45 def add(a,b):return(a[0]+b[0],a[1]+b[1])
46 def neg(a):return(-a[1],-a[0])
47 def sub(a,b):return add(a,neg(b))
48 def mul(a,b):
49     p=[x*y for x in a for y in b]
50     return(min(p)//S,-((-max(p))//S))
51 def divi(a,n):return(a[0]//n,-((-a[1])//n))
52 def scale(a,n):return(a[0]*n,a[1]*n) if n>=0 else(a[1]*n,a[0]*n)
53
54 @lru_cache(None)
55 def trig_pi(num,den):
56     # Both sine and cosine are enclosed by 61 Taylor terms at x in [0,pi].
57     # Remainder <=4^121/121! <1e-120 < 1/S.
58     x=divi(scale(PI,num),den);assert 0<=x[0]<=x[1]<=4*S
59     xx=mul(x,x);st=x;ct=(S,S);sn=x;cs=ct

```

```

60 for k in range(1,61):
61     st=divi(mul(st,xx),(2*k)*(2*k+1))
62     ct=divi(mul(ct,xx),(2*k-1)*(2*k))
63     sn=add(sn,st if k%2==0 else neg(st))
64     cs=add(cs,ct if k%2==0 else neg(ct))
65     return((sn[0]-1,sn[1]+1),(cs[0]-1,cs[1]+1))
66
67 def cmul(a,b):
68     return(sub(mul(a[0],b[0]),mul(a[1],b[1])),add(mul(a[0],b[1]),mul(a[1],b[0])))
69 def cadd(a,b):return(add(a[0],b[0]),add(a[1],b[1]))
70 def zero_excluded(a):return a[0][0]>0 or a[0][1]<0 or a[1][0]>0 or a[1][1]<0
71
72 class Ring:
73     def __init__(self,v):
74         self.v=v;self.mod=cyclo[2*v];self.d=len(self.mod)-1;self.zero=(0,)*self.d;self.one=(1,)+(0,)*(self.d-1)
75         self.omega=self.reduce([0,1]);self.powers=[self.one]
76         for k in range(1,2*v):self.powers.append(self.mul(self.powers[-1],self.omega))
77     def reduce(self,a):
78         a=list(a)+[0]*max(0,self.d-len(a))
79         for k in range(len(a)-1,self.d-1,-1):
80             c=a[k]
81             for j in range(self.d+1):a[k-self.d+j]-=c*self.mod[j]
82         return tuple(a[:self.d])
83     def add(self,a,b):return tuple(x+y for x,y in zip(a,b))
84     def neg(self,a):return tuple(-x for x in a)
85     def mul(self,a,b):
86         out=[0]*(2*self.d-1)
87         for i,x in enumerate(a):
88             for j,y in enumerate(b):out[i+j]+=x*y
89         return self.reduce(out)
90     def interval(self,a):
91         re=(0,0);im=(0,0)
92         for j,x in enumerate(a):
93             if x:
94                 sn,cs=trig_pi(j,self.v)
95                 re=add(re,scale(cs,x));im=sub(im,scale(sn,x))
96         return(re,im)
97
98 def canon(s):return min(s[i:]+s[:i] for s in(s,s[::-1]) for i in range(N))
99 FULL = '--all-words' in sys.argv
100 words=([format(x,'013b') for x in range(1,2**N-1)] if FULL else
101         sorted({canon(format(x,'013b')} for x in range(1,2**N-1)}))
102 assert len(words)==(8190 if FULL else 378)
103 allcert=[];start=time.time()
104 for iw,w in enumerate(words):
105     v=w.count('1');ring=Ring(v);B=[0]
106     for b in w[:-1]:B.append(B[-1]+int(b))
107     exps=[v*k-N*B[k] for k in range(N)];lo=min(exps);hi=max(exps)
108     P=[ring.zero]*(hi-lo+1)
109     for k,e in enumerate(exps):
110         c=ring.powers[B[k]%(2*v)]
111         if k%2:c=ring.neg(c)
112         P[e-lo]=ring.add(P[e-lo],c)
113     r=pow(13,-1,2*v)
114     F=[ring.powers[(r*(12-j))%(2*v)] for j in range(13)]
115     # Exact monic division by regular-configuration factor.
116     rem=P[:];Q=[ring.zero]*(len(P)-12)
117     for k in range(len(P)-13,-1,-1):
118         c=rem[k+12];Q[k]=c
119         for j,f in enumerate(F):rem[k+j]=ring.add(rem[k+j],ring.neg(ring.mul(c,f)))
120     assert all(a==ring.zero for a in rem),(w,'regular factor failure')
121     # Remove all exact roots z=1 (collapsed configurations).
122     endpoint_order=0
123     while True:
124         val=ring.zero
125         for a in Q:val=ring.add(val,a)
126         if val!=ring.zero:break
127     assert len(Q)>1

```

```

128 q=[ring.zero]*(len(Q)-1);q[-1]=Q[-1]
129 for k in range(len(q)-2,-1,-1):q[k]=ring.add(Q[k+1],q[k+1])
130 assert ring.add(Q[0],q[0])==ring.zero
131 Q=q;endpoint_order+=1
132 coeff=[ring.interval(a) for a in Q]
133 stack=[(0,1,0)];accepted=[];maxdepth=0
134 while stack:
135     left,right,depth=stack.pop();den=2**depth
136     # theta in [pi*left/(13*v*den),pi*right/(13*v*den)].
137     sl,cl=trig_pi(left,13*v*den);sr,cr=trig_pi(right,13*v*den)
138     z=((cr[0],cl[1]),(-sr[1],-sl[0]))
139     val=coeff[-1]
140     for c in reversed(coeff[:-1]):val=cadd(cmul(val,z),c)
141     if zero_excluded(val):
142         accepted.append((left,right,depth));maxdepth=max(maxdepth,depth)
143     else:
144         if depth>=30:raise RuntimeError((w,'inconclusive',left,right,depth,val))
145         stack.append((left*2+1,right*2,depth+1));stack.append((left*2,left*2+1,depth+1))
146 cert={'word':w,'v':v,'degree_P':len(P)-1,'degree_Q':len(Q)-1,'removed_z1':endpoint_order,'intervals':len(
147     accepted),'max_depth':maxdepth,'partition':accepted}
148 allcert.append(cert)
149 if iw%(1000 if FULL else 25)==0:print(iw+1,w,'intervals',len(accepted),'depth',maxdepth,'seconds',round(time.
150     time()-start,2),flush=True)
151 output_name='cycle13_allwords_certificate.json' if FULL else 'cycle13_certificate.json'
152 with Path(__file__).with_name(output_name).open('w') as f:json.dump({'arithmetic':'integer fixed-point outward
153     intervals','scale':str(S),'words':len(words),'total_intervals':sum(c['intervals'] for c in allcert),'
154     max_depth':max(c['max_depth'] for c in allcert),'certificates':allcert},f,indent=2)
155 print('PROVED',len(words),'cases;',sum(c['intervals'] for c in allcert),'intervals; max depth',max(c['
156     max_depth'] for c in allcert),'elapsed',time.time()-start,flush=True)

```