Where to build a temple, and where to dig to find one

Greg Aloupis* ‖ Jean Cardinal† ‖ Sébastien Collette‡ ‖ John Iacono§ Stefan Langerman¶ ‖

Abstract

In this paper, we analyze the time complexity of finding regular polygons in a set of n points. We use two different approaches to find regular polygons, depending on their number of edges. Those with $o(n^{0.068})$ edges are found by sweeping a line through the set of points, while the larger polygons are found by random sampling. We can find all the polygons with high probability in $O(n^{2.068+\epsilon})$ expected time for every positive ϵ. This compares well to the $O(n^{2.136+\epsilon})$ deterministic algorithm of Brass [1]. Our method can also be used to find incomplete regular polygons, where up to a constant fraction of the vertices are missing.

1 Introduction

The focus of this study is on the detection of regular structure in point sets. Our motivation comes from observations that have been published concerning extraordinary symmetries in the placements of ancient towns, temples and other important locations. Specifically, the oracle of Delphi has been measured to be the apex of isosceles triangles with at least seven pairs of ancient Greek cities[1]. The same is true for the oracle at Dodoni, while the small island of Delos is the apex of at least thirteen isosceles triangles. All three of the central locations were considered to be among the most important of places, and in fact Delphi was considered to be the navel of the world. In general, one may find seemingly countless collinearities, reflective symmetries, partial n-gons and networks of isosceles triangles when looking at the graph of cities in the ancient world, from the British Isles to the Middle East.

We will not concern ourselves further questioning whether such structures were carefully constructed or instead an expected result on large complete geometric graphs. However, the topic generates other interesting questions. If one chooses a particular location as a temple, it is not difficult to construct cities (at least on paper) so that the temple becomes the center of several symmetries. What about the opposite? Given a set of existing cities, where should one decide to place a temple? Or, to ask differently, where should one look for a hidden temple?

2 Previous Work

Given a set of n points, we wish to find the maximum subset which satisfies a specific symmetry or structure.

The algorithm by Brass [1], for finding maximum symmetric subsets in a set of points, is capable of handling reflective lines, translations, rotational symmetries and repeated sets. The time complexity is $O(n^{2.136+\epsilon})$ for every positive ϵ.

Brass noted that another result of his algorithm was to find all regular polygons contained in the set, which is remarkable because there exist sets of n points containing $O(n^2)$ regular polygons.

The bound of Brass was reached by analyzing the maximum number of possible isosceles triangles formed by a set of points in the plane. Pach and Agarwal [2] gave a simple bound of $O(n^{2+1/3})$ for that problem. This was improved informally by Pach and Tardos [3] to $O(n^{2.136+\epsilon})$ for every positive ϵ.

3 Model of Computation

We assume that all coordinates and other values are stored in a format that allows constant time equality testing and hashing. As hashing is only used to speed up one dimensional searches, it can be substituted with a comparison-based structure at a logarithmic addition cost which is absorbed into the ϵ. Furthermore, as exact computation methods are typically not used, comparison based structures can be used to allow equality tests to be substituted with proximity tests for suitably small proximities to compensate for any small discrepancies in the computation.

4 Results

We improve the time complexity for detecting maximal regular polygons in point sets, although unlike

*greg.aloupis@ulb.ac.be

†jcardin@ulb.ac.be

‡Aspirant du F.N.R.S., sebastien.collette@ulb.ac.be

§Department of Computer and Information Science, Polytechnic University, 5 MetroTech Center, Brooklyn NY 11201. Research supported in part by NSF grants CCF-0430849 and OISE-0334653. http://john.poly.edu.

¶Chercheur qualifié du F.N.R.S., stefan.langerman@ulb.ac.be

‖Computer Science Department, Université Libre de Bruxelles, CP212, Boulevard du Triomphe, 1050 Bruxelles, Belgium.

[1]For an informal, illustrative and detailed account, see `http://www.geocities.com/sfetel/en/geometry.htm.`

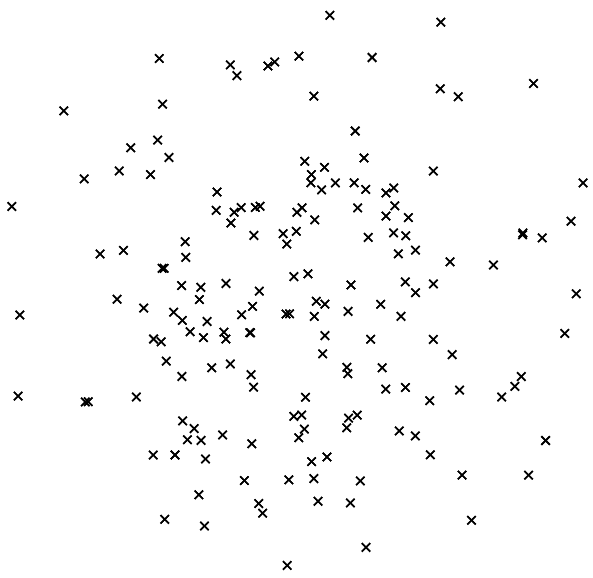
Figure 1: Can you find all of the regular k-gons in this figure? Solution is in Figure 2.

the algorithm by Brass, our algorithm is randomized.

Our new bound is $O(n^{2.068+\epsilon})$. Notice that the fractional component in the exponent of n has been halved. This is no coincidence. Our algorithm is designed to reduce this fraction by a factor of 2. Thus, any improvement over the result of Pach and Tardos will be directly reflected in our algorithm.

Our main result appears as Theorem 7 at the end of this section. The remainder of the section is devoted to presenting lemmata that guide us to the Theorem.

Lemma 1 *In time $O(n^{2.068}\log n)$ we can find all $\leq n^{0.068}$-gons in a set S of n points.*

Proof. First, we compute all line segments defined by pairs of points in S, and we view this as an embedded graph. We compute a hash table at each vertex containing incident edges which are stored by key value and length.

Let $\phi_i = \pi - \frac{2\pi}{i}$, and $\Phi = \{\phi_3, \phi_4, \ldots \phi_{n^{0.068}}\}$. Thus, Φ is the set of all angles formed by three adjacent vertices in a regular $\leq n^{0.068}$-gon. Thus, for any edge, it is possible to determine using the set Φ and the hash tables at its incident vertices if there is a possible neighboring edge that could be in a $\leq n^{0.068}$-gon. This can be done in $O(n^{0.068})$ table lookups, and therefore time.

The algorithm is a line sweep. As we sweep we keep the messages of the following form on the edges that intersect the sweep line: "possible k-gon above/below." Edges could carry several messages at one time, and edges deliver their messages to the right endpoint of the edge when the sweep line arrives there.

The sweep proceeds in the normal manner, from left to right, where at vertices we process several types of events:

- Origination Event. This event detects the possible leftmost point of a k-gon. If two edges of the same length are to the right of the active vertex and are at angle ϕ_k, then we give a "possible k-gon below" signal to the upper edge and a "possible k-gon above signal to the lower edge.
- Propagation Event. This event traces a possible k-gon though one edge on its upper or lower portion. If a "possible k-gon above/below" signal is received from an edge, and if there is a right facing edge of the same length and angle ϕ_k away from the edge delivering the message, the message is propagated to this right facing edge. (The orientation of the angle is determined by whether the signal is an above or below signal). If there is no such right-facing edge, the message is discarded, as what we thought might be a k-gon is not.
- Termination Event. This event occurs when we detect the rightmost vertex of a k-gon. If a vertex receives a "possible k-gon above" and a "possible k-gon" below signals from edges on the left of the same length, angle ϕ_k apart, and in the proper orientation, then we have finished the process of discovering a complete k-gon. We output the description of the k-gon (the center, rotation, and gonality can be easily determined from the information at hand), and we do not propagate the two incoming signals.

Hopefully, the correctness of the method is clear from the above description. We now focus on the runtime. It takes $O(n^2 \log n)$ time to perform a line sweep. (The astute reader will realize that a topological sweep could be used instead at a cost of $O(n^2)$, but as all logs get absorbed in the ϵ, this observation is not critical). As there are only $2n^{0.068}$ possible signals, if all of them appeared on all $O(n^2)$ edges, this would create a total of $O(n^{2.068})$ signals to handle over the entire sweep. Propagation and termination events are done with table lookup and take constant time for each event. Generating all origination events takes $O(n^{0.068})$ table lookups for each incident edge to the right, as all angles in Φ are searched. Thus, the total runtime is $O(n^{2.068})$ if we allow hashing, and $O(n^{2.068}\log n)$ in the model of computation described above. $\square$

Definition 1 *A* prime-skip triangle *of a k-gon is an isosceles triangle formed by three vertices of the k-gon where the two non-apex vertices defining the equal sides of the isosceles triangle are at a prime number of vertices away from each other.*

Observation 1 *Given a prime-skip triangle of a k-gon G, k and thus G can be uniquely determined in logarithmic time.*

The simplest way to determine which one it is in constant time is to build a lookup table of size $O(n^2)$. If we do not allow hashing, a logarithmic factor is added as we have to look in a binary search tree. The following follows directly from the prime number theorem:

Observation 2 *Given a k-gon G, a random isosceles triangle is prime-skip with probability $\frac{1}{\log n}$.*

Lemma 2 *The sum of complexities of all $\geq n^{0.068}$-gons is at most $n^{2.068+\epsilon}$.*

Proof. Let κ_i be the number of i-gons in a fixed set S of n points. The sum of the complexities of all $\geq n^{0.068}$-gons is $\sum_{i=n^{0.068}}^{n} i\kappa_i$. Any k-gon generates at most $O(k^2)$ isosceles triangles of which at most $O(\frac{k^2}{\log n})$ are prime-skip. Thus there are at most $\sum_{i=n^{0.068}}^{n} \frac{i^2\kappa_i}{\log i}$ prime-skip, and therefore distinct, isosceles triangles. We know from [3] that there are at most $O(n^{2.136+\epsilon})$ distinct isosceles triangles and thus:

$$\sum_{i=n^{0.068}}^{n} \frac{i^2\kappa_i}{\log i} = O(n^{2.136+\epsilon})$$

which since $n^{0.068} \leq i \leq n$ gives

$$\frac{n^{0.068}}{\log n} \sum_{i=n^{0.068}}^{n} i\kappa_i = O(n^{2.136+\epsilon})$$

and dividing and absorbing the log into the ϵ gives

$$\sum_{i=n^{0.068}}^{n} i\kappa_i = O(n^{2.068+\epsilon})$$

This last equation is exactly the statement of the lemma. □

Definition 2 *A* more-than-half-full *k-gon is a subset of the vertices of a regular k-gon containing at least $k/2$ points.*

Corollary 3 *The sum of complexities of all more-than-half-full $\geq n^{0.068}$-gons in a set S of n points is at most $n^{2.068+\epsilon}$*

Proof. This is an easy variant of Lemma 2, as the more-than-half-full condition does not change anything but constants buried in the big-O. □

Lemma 4 *Given an isosceles triangle T, we can determine one of the following:*

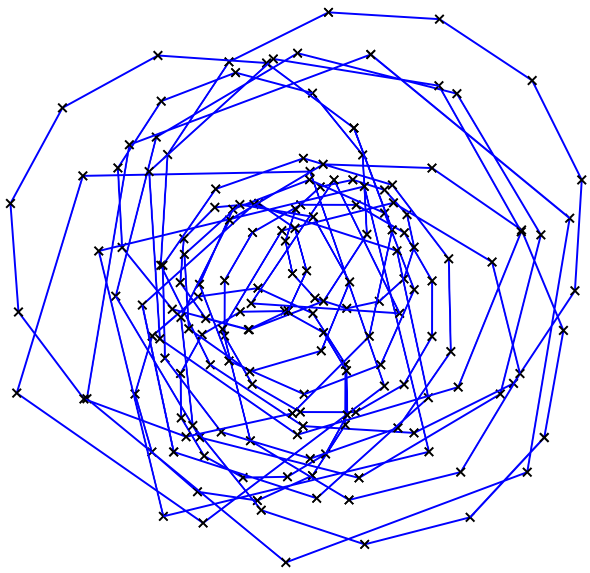

Figure 2: Here we show all of the regular k-gons in the point set illustrated in Figure 1.

- *if the candidate k-gon containing T is less-than-half-full, then we can answer "T is not part of a k-gon" in $O(1)$ time;*
- *otherwise we can decide in $O(k)$ time whether T is part of a k-gon or not.*

Proof. Given an isosceles triangle T, the center of the candidate polygon comes from the circumcenter which can be found in constant time. From the apex angle of the triangle, we can determine what is the smallest value of k such that T could be three vertices of a k-gon. This could be done by expressing the angle as a fraction of 2π in lowest terms, or alternately via a table lookup. Since there are at most n^2 possible apex angles in triangles with vertices in $\leq n-$gons, such a table could be precomputed in time $O(n^2)$, which does not asymptotically contribute to the overall runtime.

Given T, the center of the polygon, and the value of k, one knows the location of all vertices on the candidate polygon. We then check to see if all of the vertices in the candidate polygon are present. We check all $k-3$ of them in random order. If any one vertex is not present we terminate the check and output nothing. If all vertices are present, we output the candidate k-gon.

As we always spend at most $O(k)$ time checking each candidate k-gon, if the candidate k-gon is more-than-half-full, we trivially spend $O(k)$ time. If the candidate k-gon is not more-than-half-full, over half of the $k-3$ tests will fail. Since the tests are ordered randomly, we expect to perform at most 2 tests, and thus expect to spend only $O(1)$ time performing the tests in this case. □

Lemma 5 *For any $\geq n^{0.068}$-gon G, at least $\frac{1}{n^2}$ of the isosceles triangles are in G.*

Proof. According to a personal communication of Pach and Tardos cited in [1], the maximum number of isosceles triangles among n points in the plane is $O(n^{2.136+\epsilon})$. Any $\geq n^{0.068}$-gon G defines $O(n^{0.136})$ isosceles triangles. Thus, $O(1/n^2)$ of the isosceles triangles have all three points in G. $\square$

Corollary 6 *For any $\geq n^{0.068}$-gon G, $\frac{1}{n^2 \log n}$ of the isosceles triangles are prime-skip triangles in G.*

Proof. Follows from Lemma 5 and Observation 2. $\square$

Theorem 7 *With high probability, we can find all regular polygons in a set S of n points in the plane in expected time $O(n^{2.068+\epsilon})$.*

Proof. Using Lemma 1, all regular $\leq n^{0.068}$-gons can be found in time $O(n^{2.068})$. We thus focus on the case of large polygons, that is, $\geq n^{0.068}$-gons.

The algorithm proceeds as follows: we pick $O(n^2 \log^2 n)$ random isosceles triangles formed from the vertices of S. Corollary 6 and the solution to the coupon collector problem indicates that we will now have with positive constant probability at least one prime-skip isosceles triangle from every $\geq n^{0.068}$-gon. Lemma 4 explains how we can determine which isosceles triangles come from k-gons and which do not. Since $O(1)$ expected time is spent checking those that are not part of a k-gon, we spend $O(n^2 \log^2 n)$ time checking all such triangles. In order to check those that are part of a k-gon, or at least a partially full k-gon, we spend time linear in the gonality for each check. Corollary 3 states that the sum of the gonalities of all more-than-half-full $\geq n^{0.068}$-gons is $O(n^{2.068})$ thus giving this same bound on the time to perform this class of checks. As we picked $O(n^2 \log^2 n)$ triangles, the total complexity is $O(n^{2.068+\epsilon})$.

$\square$

5 Variants

While we have focused on the case of finding regular k-gons, our algorithm can be used to find all k-gons where a constant fraction of the points in the k-gon are missing, by using a Monte Carlo version of Lemma 4. The identification of such almost-complete symmetric sets of historical sites can aid the computational archaeologist identify possible locations for exploratory missions in search of buried ruins. This variant has the same runtime as the original algorithm.

Also, by processing all k-gons with the same center, more complex types of symmetry can be easily detected.

6 Acknowledgments

We would like to thank Boris Aronov, Jean Chapelle and Erik Demaine for interesting discussions about chords, polygons, and Euclid in general.

References

[1] P. Brass. On finding maximum-cardinality symmetric subsets. *Computational Geometry - Theory and Applications*, 24(1):19–25, 2003.

[2] J. Pach and P. K. Agarwal. *Combinatorial geometry.* Wiley-Interscience, 1995.

[3] J. Pach and G. Tardos. Personal communication cited in [1].