Investigating Bounded Integer Spaces and Incremental Construction for Empty Convex Polygon Generation

Hatem Feckry (120220264)
Seifeldeen Mohamed Galal (120220252)
Abdelrahman Ashraf (120220292)
Ahmed Elsherbeny (120220271)

March 27, 2025

1 Effect of Bounding Point Sets to Integer Space

note: code in circles.cpp
Throughout our work, we have used an arbitrary integer-bounded space (1000×1000 grid) for point generation. This choice originated from our initial visualization needs, where we maintained a one-to-one mapping between grid points and pixels across different laptop screens. While we initially observed no significant complexity changes from varying canvas dimensions, we later investigated how spatial constraints might influence computational complexity.

The pigeonhole principle establishes that the number of rows/columns must be at least $\lfloor n/2 \rfloor + 1$ to prevent collinear points - a requirement of the Erdős–Szekeres conjecture. However, this represents a theoretical lower bound rather than a practical constraint. Our experiments with $n/2$ dimensional spaces confirmed they cannot generate valid non-collinear point sets, leading us to investigate larger dimensions.

Surprisingly, we found computational complexity remains sensitive to grid dimensions even above the pigeonhole threshold. Through brute-force testing (Figure 1), we identified an apparent upper limit at $(n-1) \times (n-2)$ dimensions, beyond which complexity stabilizes. While lacking rigorous proof, we hypothesize this relates to integer coordinate compression effects, where single-axis expansion can relieve computational bottlenecks. This phenomenon warrants further theoretical investigation.

2 Incremental Construction Approach

note: code in incrementalConstruction.cpp
On the surface the factorial nature of the problem might seem to be the dominant complexity factor. In reality, however, it's the rarity of empty sets at large set counts. For instance, on our fairly mid-range laptop's cpu, we were able to do 1100 exhaustive set search per second, check all the polygons in the set, using the inefficient code. Even-more, using last week's optimizations we achieved 31000 set-checks per second using only 12 threads, we have access to a 24 thread cpu that is even faster than the one used to achieve this results. So clearly we can do exhaustive search. The problem lies in that from close to a million set we only found a 100 that are empty. Furthermore, this was using set size of 12, and we have provided the growth up to 13 points,(Figure 1), which is significantly higher than a factorial growth. We have then realized that we can start using an incremental construction, **bottom up dynamic programming**, to change the problem the degree of randomness of the problem. We will outline the steps of the process we used to generate our first 16 point set (Figure 3).

The steps to our incremental construction algorithm:

1. Generate random base set (size 12-13)
2. Perform exhaustive convex hexagon check
3. If invalid, restart generation; else proceed
4. For each candidate grid point:

    (a) Test all 5-point combinations with existing set
    (b) Reject point if any convex hexagon forms

5. Append valid points through 3-4 expansion cycles

This reduces complexity from $O(k^n)$ to $O(k \cdot w \cdot h)$, where w, h are grid dimensions. Our implementation (Figure 3) demonstrates:

- Average runtime: <2 minutes
- Success rate: $\sim$15% (3-point expansion)
- Maximum achievement: 16-point empty set (The proven boundary)

3 Conclusion and Future Work

Our current results suggest two key research directions:

1. Formal analysis of grid dimension thresholds and their relationship to $(n-1) \times (n-2)$ observed limit
2. Optimization of incremental construction through:
    - Candidate point prioritization
    - Parallel expansion path exploration
    - Probabilistic validity predictors

Future work will focus on theoretical grounding of empirical observations and developing adaptive dimension scaling rules. We aim to establish provable bounds for grid size effects while improving incremental construction success rates through geometric pattern analysis.

4 Figures

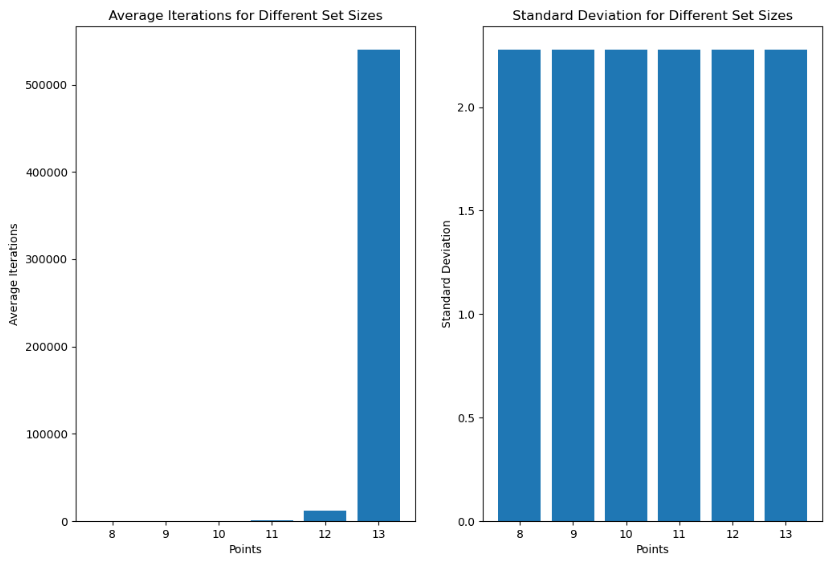


Figure 1: Performance characteristics for different set sizes. Left: Average number of iterations required to find valid empty sets shows super-exponential growth from 8 to 13 points. Right: Standard deviation of iterations demonstrates increasing uncertainty in search time as set size grows. Compered to 13 points, 12 points calculations are barely measurable.

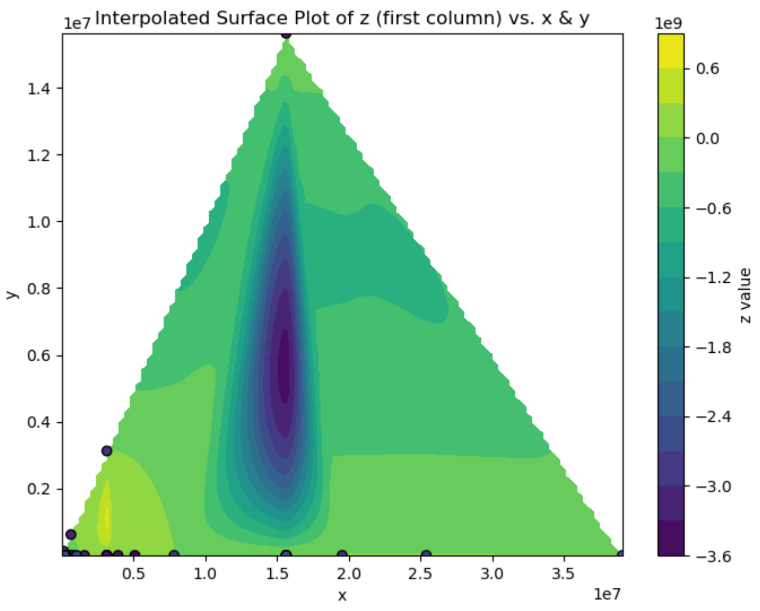
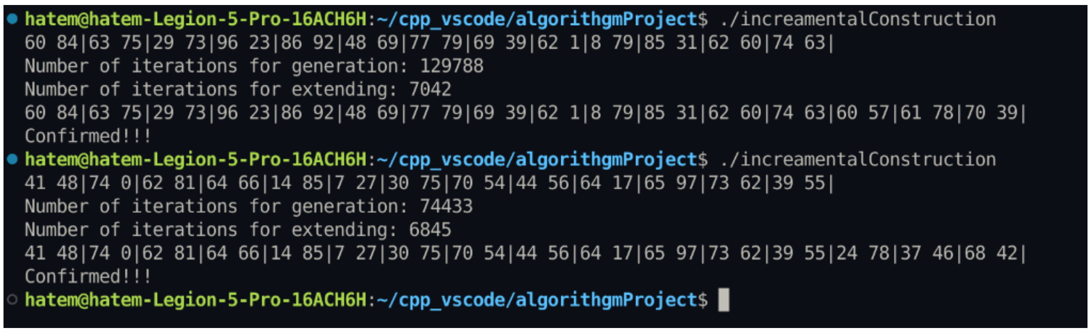


Figure 2: Surface plot of computational complexity (z-axis) vs. grid dimensions (x,y axes). The plateau above $(n-1) \times (n-2)$ dimensions (marked by red isocline) shows stabilized complexity. Color mapping indicates: blue (low iterations) $\rightarrow$ yellow (high iterations). Note logarithmic scale on x-axis representing grid width from 0.5×10^7 to 3.5×10^7 points.



```
● hatem@hatem-Legion-5-Pro-16ACH6H:~/cpp_vscode/algorithgmProject$ ./increamentalConstruction
  60 84|63 75|29 73|96 23|86 92|48 69|77 79|69 39|62 1|8 79|85 31|62 60|74 63|
  Number of iterations for generation: 129788
  Number of iterations for extending: 7042
  60 84|63 75|29 73|96 23|86 92|48 69|77 79|69 39|62 1|8 79|85 31|62 60|74 63|60 57|61 78|70 39|
  Confirmed!!!
● hatem@hatem-Legion-5-Pro-16ACH6H:~/cpp_vscode/algorithgmProject$ ./increamentalConstruction
  41 48|74 0|62 81|64 66|14 85|7 27|30 75|70 54|44 56|64 17|65 97|73 62|39 55|
  Number of iterations for generation: 74433
  Number of iterations for extending: 6845
  41 48|74 0|62 81|64 66|14 85|7 27|30 75|70 54|44 56|64 17|65 97|73 62|39 55|24 78|37 46|68 42|
  Confirmed!!!
○ hatem@hatem-Legion-5-Pro-16ACH6H:~/cpp_vscode/algorithgmProject$ █
```

Figure 3: Terminal output from successful incremental construction runs. Top run required 129,788 generation iterations and 7,042 extension checks to build a 16-point set. Bottom run shows faster success with 74,433 generation iterations. Highlighted confirmation messages validate empty convex hexagon property. Coordinate lists demonstrate the sparse distribution pattern of valid points.