Algorithms Weekly Report

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

April 17, 2025

Introduction

This report summarizes the team's progress during the week of April 11–17, 2025. We tackled two primary algorithmic improvements: enhancing our incremental point-set construction via multi-threading and symmetry-based bounding, and developing a transformer-based framework for 2D point-set repair and completion in the Happy Ending problem. We also analyzed our algorithm's running time through log-transformed modeling and Monte Carlo simulations to better understand growth rates and success probabilities. Key findings and challenges from each task are highlighted below.

1 Multi-threading Problems

Last week, we tried to make incremental construction multi-threaded by dividing the generation grid into 14-point batches and dispatching them to separate threads. Each thread, however, tested validity against the *original* seed, leading to many completely invalid outputs: when two or more neighboring points in the same batch are added simultaneously, they "poison" each other. In fact, on inspecting a 30-point set we discovered that in the very first batch all 14 points were accepted at once.

Our first attempted fix was to restrict each batch to adding at most one point per patch. This makes sense on large grids—far points and near points differ only slightly in angle—yet on smaller regions (e.g. 50-point squares) the angular variation among 14 points can be large enough to admit multiple points per batch.

We then moved multi-threading down into the Graham-scan routine itself, but this spawned so many threads that VS Code crashed (we detached the debugger to recover). Even after that, CPU utilization peaked at only about 10

2 Bounding Analysis

To boost our success probability via geometric symmetry, we experimented with *bounding* the initial seed. Random generation is scale-independent, but our incremental construction runtime grows roughly quadratically with grid size. Last week we applied a translational symmetry: generate the seed in only a fraction of the full grid. In principle, this increases variation among candidate points, but in practice it proved counter-productive—seeding in a small corner makes it extremely hard to add points near the opposite edge, where angular changes are minimal (see parallax).

To overcome this, we introduced *scaling symmetry*: generate and increment on an $N \times N$ grid, then scale every point up (e.g. by a factor of two to a $2N \times 2N$ grid) and attempt one more round of insertions. Because our grid is integer-based, this creates new candidate coordinates that were previously only approximated.

We show below the set generated on a 100×100 grid (Figure 2), and the same set after scaling to 200×200, where a new point appears (Figure 3). Although promising, this method's computational cost scales poorly: at 400×400 we could not add any further points. In effect, a $16\times$ increase in grid resolution yielded only a single new point. Unless we dramatically improve our parallelism—perhaps via GPU acceleration—this approach remains impractical.

3 Transformer Implementation

This week, we focused on experimenting with transformer models for correcting and completing 2D point-set configurations within the context of the Happy Ending problem. Specifically, we explored the potential of using a transformer model to repair corrupted point locations or complete partially-obscured point sets. This approach would, in principle, lead to constant-time inference during validation, significantly improving efficiency compared to traditional brute-force search methods.

3.1 Motivation

The core motivation behind using transformers for this task stems from the computational inefficiencies of traditional point-set validation methods. In our problem setup, a valid configuration is represented as a 2D binary image where black pixels represent points in the configuration. Given the corruption of these point locations (e.g., misaligned points or missing points), a transformer model offers the possibility of efficiently repairing or completing these sets by leveraging its ability to capture global dependencies. The model's potential for constant-time inference, once trained, is crucial for reducing the computational overhead associated with point-set validation.

3.2 Problem Setup

We represent point configurations visually as $N \times N$ binary images, where each pixel is either black (value 1) or white (value 0). Black pixels correspond to points in the set, and white

pixels represent empty space. A configuration of N points on a 2D grid can thus be encoded as a binary image of size $N \times N$.

Corruption types:

- **Shifted points**: misaligned black pixels.
- **Missing points**: points entirely missing or obscured.
- **Occluded regions**: hidden parts of the grid requiring inference.

Our objective is to reconstruct the original configuration by placing points back into correct positions and restoring missing points—akin to an image inpainting problem. Currently, generating point-set images is the bottleneck ($\tilde{3}0$ valid sets/hour), so speeding up data generation or synthetic augmentation is crucial to avoid overfitting when training transformers.

3.3 Approach

Two transformer-based strategies were explored:

3.3.1 Masked Image Modeling (Inpainting)

Use a Vision Transformer (ViT) or Masked Autoencoder (MAE) to randomly mask portions of the point-set image and train the transformer to predict masked pixels. Once trained, the model infers correct locations of points from corrupted inputs in constant time.

3.3.2 Conditional Diffusion Models

Employ a transformer-based diffusion model that starts from a noisy/corrupted image and iteratively refines it to recover the true configuration. Although multi-step, inference remains polynomial in pixel count and near-constant time post-training.

3.4 Evaluation

Performance metrics:

- **Total Euclidean distance** between predicted and true point locations.
- **Binary success rate**: proportion of cases with all points matched within tolerance.

3.5 Challenges

- **Data scarcity and overfitting**: need larger datasets via faster generation or augmentation.
- **Geometric consistency**: enforcing constraints (e.g., no three collinear points).
- **Spatial encoding**: enhancing positional encodings or using hybrid CNN-transformers for better spatial reasoning.

4 Modeling Running Time Data

4.1 Log-Transformed Extrapolation of Averages

During data analysis, we applied a logarithmic transformation to stabilize growth-rate modeling. Exponential and linear fits failed on raw data, leading us to apply a quadratic fit in log-space, which better approximates the super-exponential growth observed.

4.2 Exponential and Linear Fits

Initial exponential and linear models diverged sharply or misrepresented growth trends, respectively.

4.3 Quadratic Fit in Log-Space

A quadratic model in log-space provided a significantly improved approximation, indicating the algorithm's time complexity grows super-exponentially.

5 Monte Carlo Simulation: Happy Ending Problem

We performed a Monte Carlo simulation (seed 11) to estimate the probability of forming a convex polygon (e.g., convex k-gon) as we increase the number of randomly generated points. The results are summarized in the table below.

5.1 Conclusion

The Monte Carlo simulations suggest that the probability of forming a valid set decreases significantly as the number of points increases. At $n = 17$, the success probability is only about 4.13%. Given this low likelihood, it seems highly unlikely that any 17-point set is completely void of convex polygons. Further research will extend this analysis to larger sets and refine our data generation and training techniques.

Conclusion

Over the past week, we made significant strides in parallelizing our point-set construction and in exploring transformer-based repair methods. Although multi-threading and bounding symmetry provided modest speedups, data generation remains the primary bottleneck for training deep models. Our log-space growth modeling and Monte Carlo studies clarified the super-exponential nature of our runtime and the rapidly diminishing probability of convex-free point sets as n grows. Moving forward, we will prioritize scalable data augmentation, investigate GPU-accelerated pipelines, and refine our transformer architectures to better incorporate geometric constraints.

6 Figures and Tables

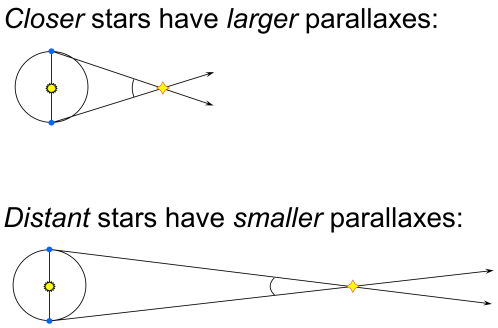

Figure 1: Small angular variations ("parallax") between distant points enable multiple simultaneous insertions.

Number of Points	**Success Rate**
12	100%
13	100%
14	94%
15	61%
16	13.5%
17	4.13% (extrapolated)

Table 1: Success rate for convex polygon formation starting from seed 11.

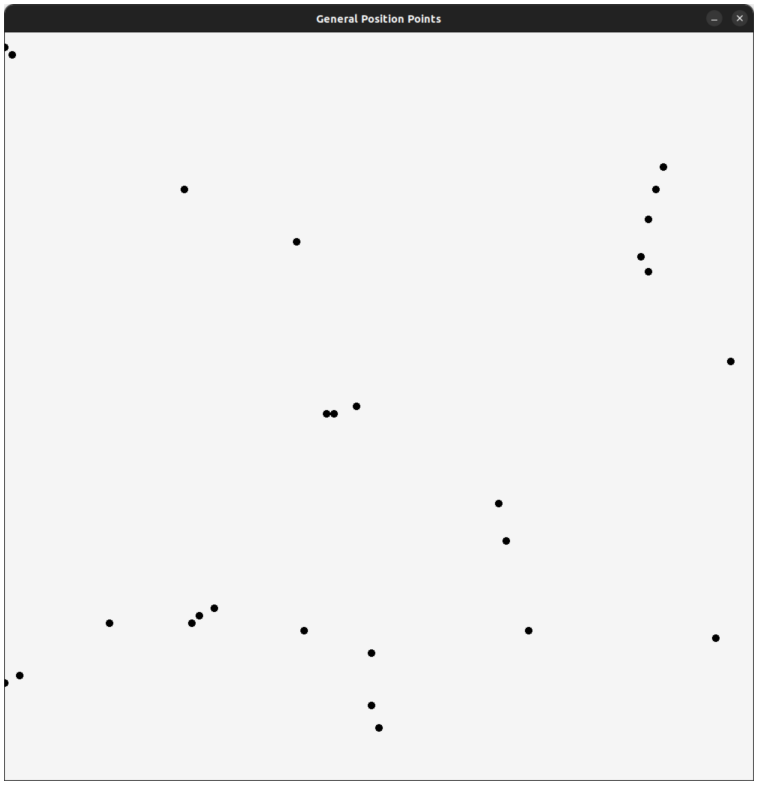

Figure 2: Set generated on a 100×100 grid.

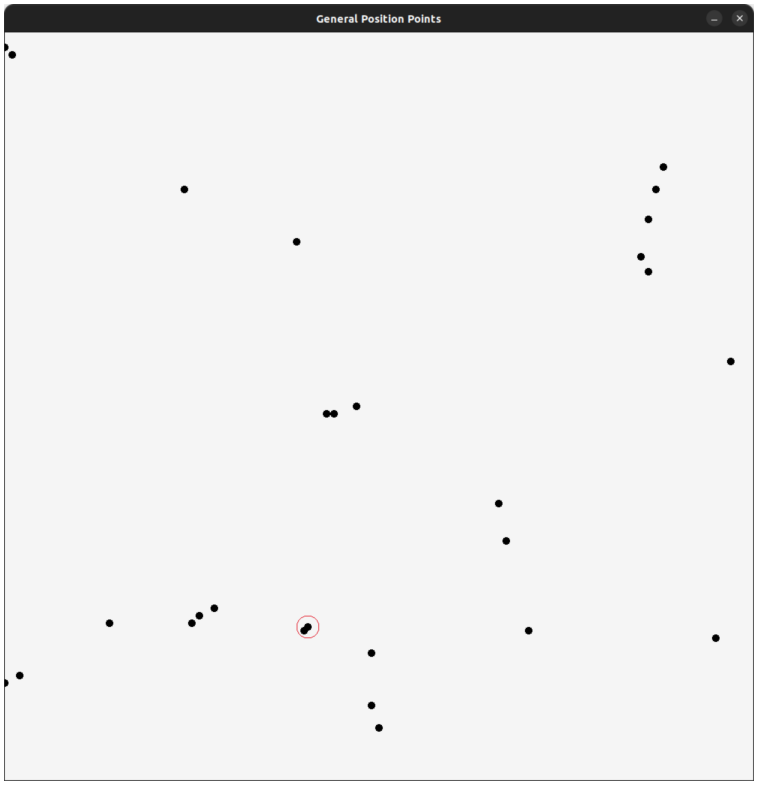

Figure 3: Same set after scaling to a 200×200 grid, showing the newly added point highlighted with the red circle.

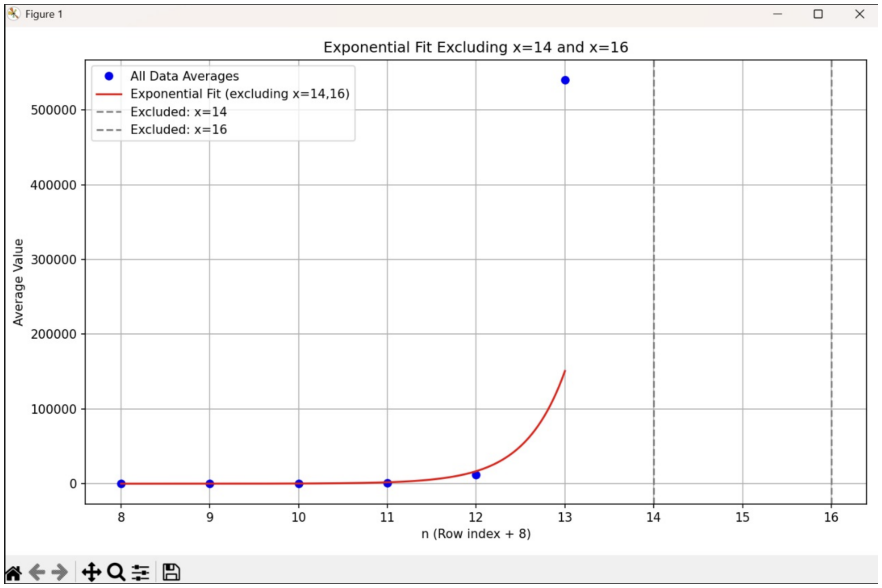

Figure 4: Exponential fit diverging from real data

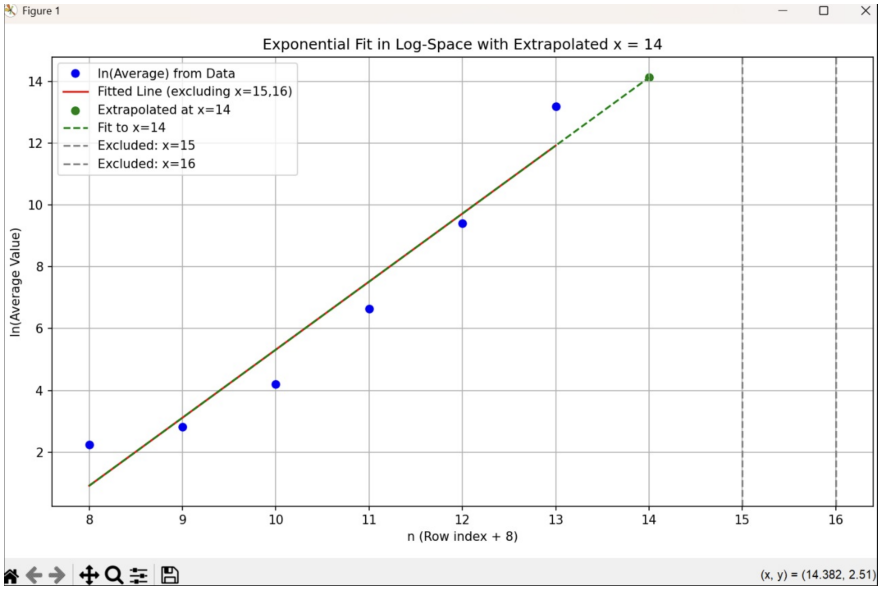

Figure 5: Linear fit misrepresenting the data structure

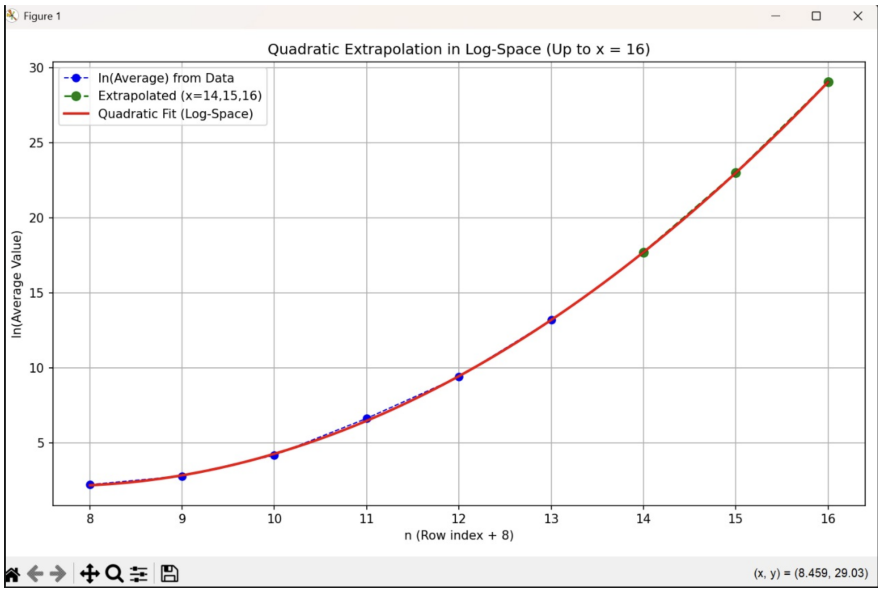

Figure 6: Quadratic fit in log-space indicating super-exponential growth

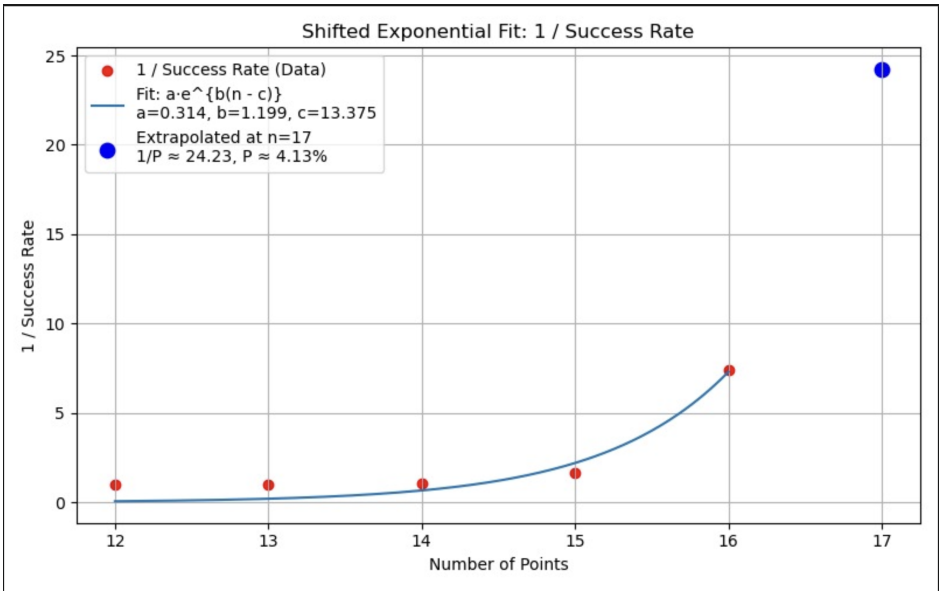

Figure 7: Shifted Exponential Fit: 1 / Success Rate