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%, of the theoretical 87.25%, (because each scan is extremely brief). Despite poor utilization, this approach still *doubled* single-threaded performance.

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 Introduction

During our data analysis, we attempted to extrapolate the performance and behavior of a computationally intensive algorithm. Initially, we worked with the raw averages of the benchmark data. However, it quickly became clear that the rate of increase in these averages was extremely high and non-linear - almost unmanageable in scale. This indicated that we were dealing with a growth rate beyond what a normal exponential or linear model could reasonably approximate.

4.2 Log Transformation and Fitting Attempts

To manage this, we applied a logarithmic transformation to the averages. This significantly smoothed the data and allowed for more stable modeling. Initially, we tried fitting an exponential function, assuming the data followed an exponential growth pattern, but this was not an appropriate fit for the transformed data.

4.3 Exponential Fit (Rejected)

As seen in Figure 4, the exponential fit quickly shoots upward and diverges sharply from the data. It was not manageable to work with due to the excessive growth and failed to represent the real progression of values effectively.

4.4 Linear Fit (Rejected)

The linear fit (Figure 5), while more manageable than the exponential model, did not pass through any real data point and hovered between them. It clearly misrepresented the structure of the data and was deemed too simplistic for the problem's nature.

4.5 Quadratic Fit in Log-Space

Eventually, we implemented a quadratic fit on the logarithm of the averages (Figure 6). This provided a significantly better approximation. A quadratic fit in log-space implies that the original (unlogged) data follows a super-exponential growth - in this case, due to the factorial time complexity of the algorithm. This confirmed our suspicion: the operation time increases more rapidly than any exponential function.

4.6 Factorial Complexity Implication

The algorithm generating the data is known to have factorial time complexity. This complexity class grows faster than exponential, indicating that the number of operations required for input size $n = 16$ is vastly greater than that for $n = 12$ or even $n = 14$, basically that means at $x = 16$, the value is greater by 10^{16} times at $x = 14$. The implication of this is

severe - running the original algorithm to full completion at $n = 16$ would require excessive wait times (10^{16} operations) and computation resources.

From our calculations, we have found that at $x = 13$, it takes 18 seconds. This would give 18×10^{16} seconds at $x = 16$, showing that it takes enormously greater time than at $x = 13$.

4.7 Constructive Algorithm and Termination Strategy

To address this, we implemented a constructive algorithm. This constructive method avoids direct computation of enormous values or permutations. Instead of recursively generating all possibilities (which would result in factorial overhead), we added a fixed upper cap on iterations - specifically 30,000 additional iterations per trial. This constraint translated to approximately a 10% increase in computation cost compared to earlier benchmarks, allowing the algorithm to terminate early while still retaining a reasonable success rate.

The result of this hybrid method yielded a success rate of approximately 30% for finding acceptable solutions within this iteration cap. These results help guide how future computational tasks of this scale can be managed efficiently with a trade-off between completeness and computation time.

5 Monte Carlo Simulation: Happy Ending Problem

Objective

To estimate how the probability of forming a convex polygon (e.g., a convex k-gon) changes as we increase the number of generated points, using a Monte Carlo method.

Method

We used Monte Carlo simulations to randomly generate sets of points in the plane and check whether they contain a convex polygon of a certain size. Each configuration was tested multiple times to estimate the success rate.

Initially, we considered using **seed = 13**, as it seemed to maximize the success probability. However, we **did not use seed 13** in our main analysis because we **lacked enough data points** for interpolation and comparison. The results from seed 13 gave near-guaranteed success rates with little variation, making it difficult to observe meaningful trends or patterns.

Experiment Focus: Seed = 11

We shifted focus to **seed = 11**, which allowed us to analyze a wider range of success probabilities as we increased the number of generated points.

interpretation

The extrapolated probability of finding a valid set at $n = 17$ is approximately 4.2%. Conversely, the probability of not finding such a set in a single trial is around 95.8%. While

this failure rate per trial appears high, the probability of not finding a valid set in **500 consecutive trials** drops drastically to approximately $1.155 \times 10^{-7}\%$.

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

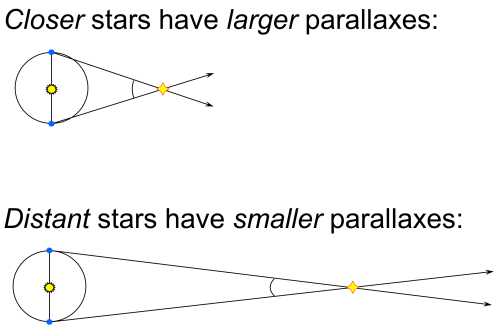

Figure 1: Small angular variations ("parallax") between distant points enable multiple simultaneous insertions.

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.

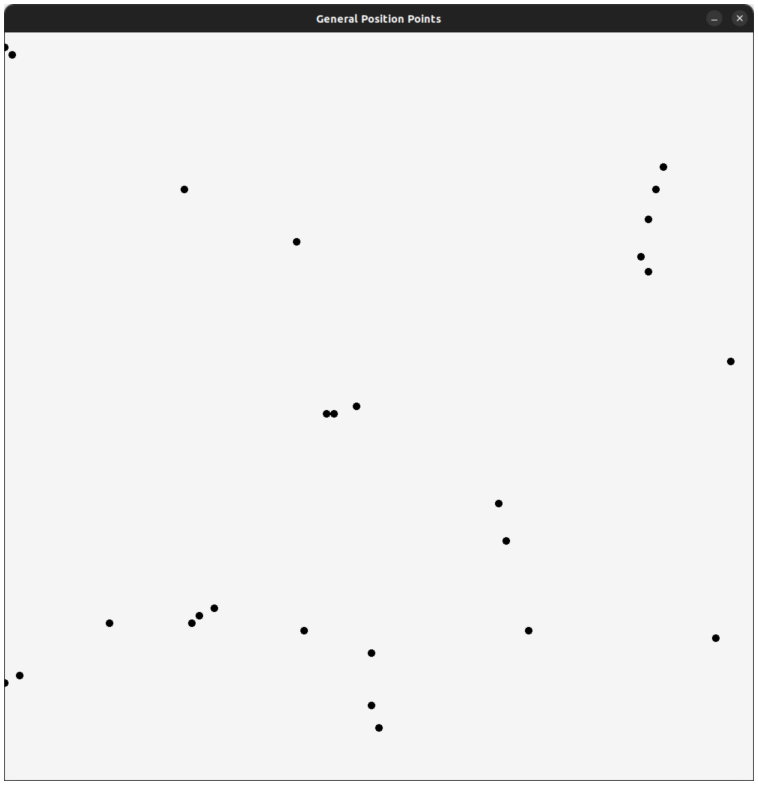

Figure 2: Set generated on a 100×100 grid.

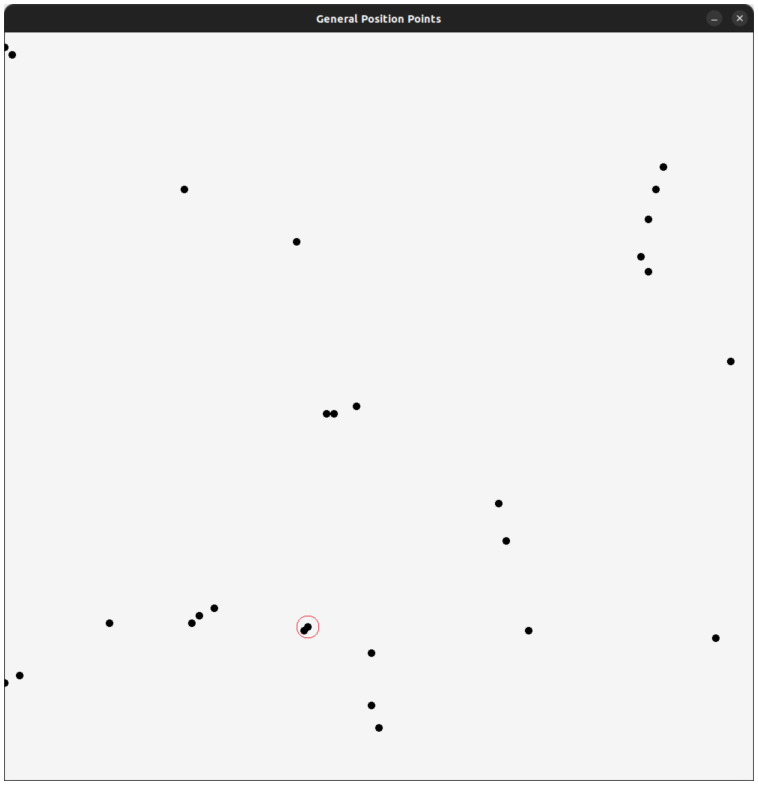

Figure 3: Same set after scaling to a 200×200 grid, showing the newly added point highlighted with the red circle.

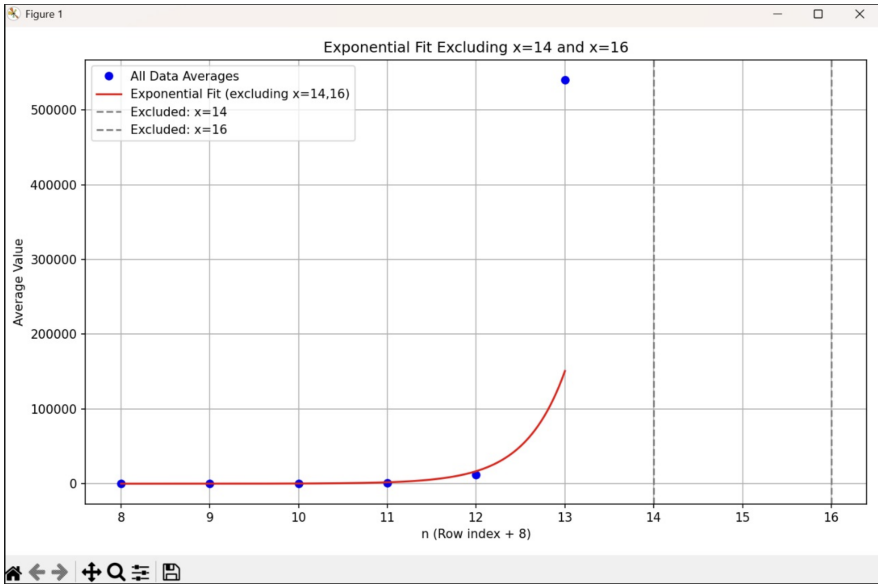

Figure 4: Exponential fit diverging from real data

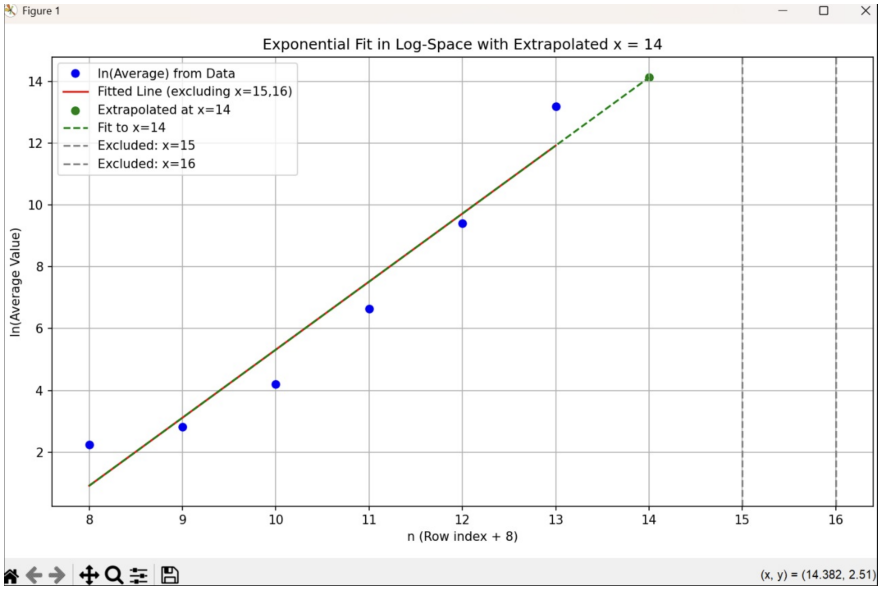

Figure 5: Linear fit misrepresenting the data structure

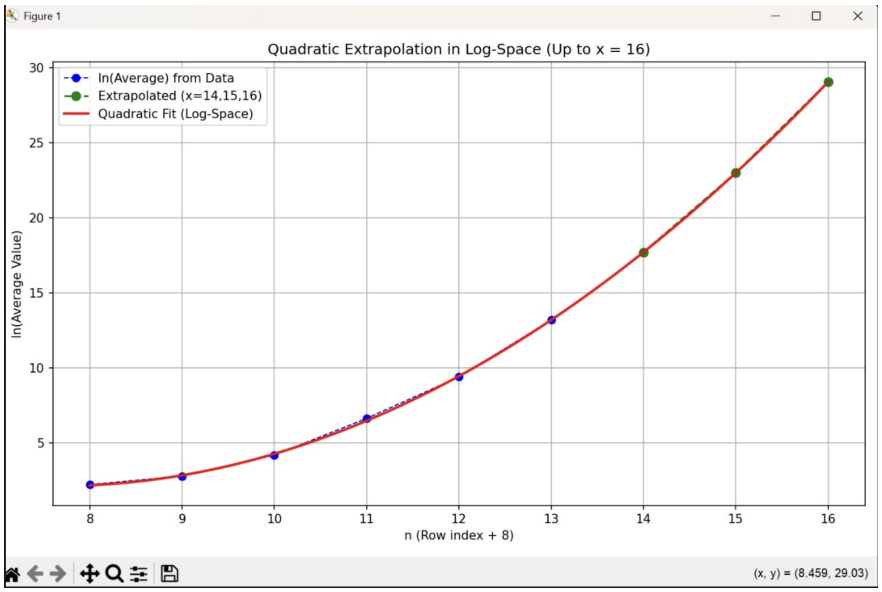

Figure 6: Quadratic fit in log-space indicating super-exponential growth

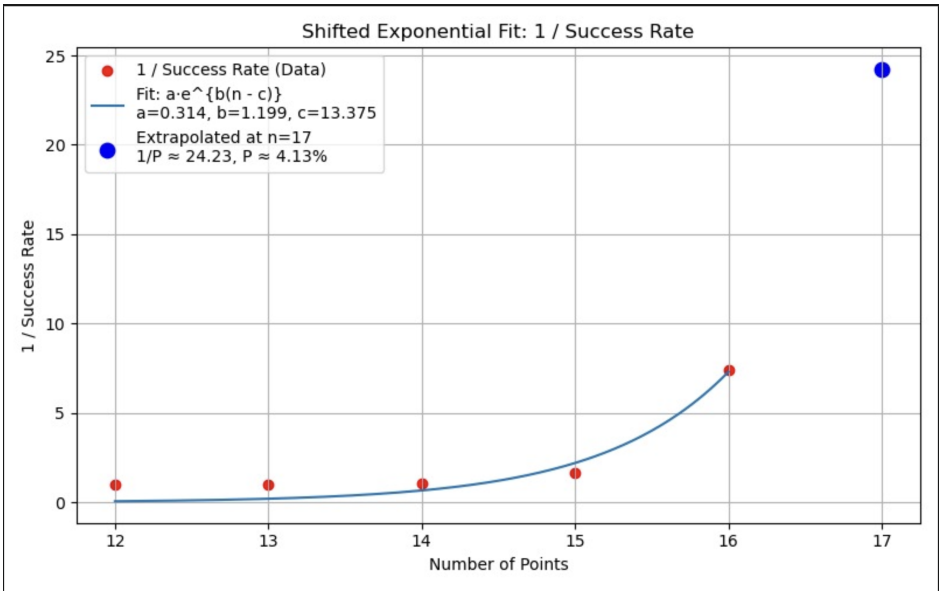

Figure 7: Shifted Exponential Fit: 1 / Success Rate