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

Motivation for Transformer Models

The core motivation behind using transformers for this task stems from the computational inefficiencies of traditional point-set validation. 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 repairing or completing these sets efficiently 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.

Problem Setup

We represent the point configurations visually as $N \times N$ binary images, where each pixel is either black (value 1) or white (value 0). In these images, 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 $N \times N$ (for simplicity, let's assume square grids).

The challenge arises when the configuration is corrupted or incomplete. Several types of corruption can occur:

- **Shifted points**: The points in the configuration may be misaligned, where the black pixels are not in their exact correct positions. This could be due to minor distortions or errors during data generation. The model's task is to correct these misalignments by shifting the points back to their valid locations.

- **Missing points**: Some points may be entirely missing or obscured, making the configuration incomplete. For example, certain regions of the image could have been removed or occluded, and the model needs to infer the locations of these missing points.
- **Occluded regions**: Parts of the grid might be hidden, either randomly or systematically. These hidden portions will have white pixels, and the transformer must infer the correct positions of points in these regions.

Given such corrupted or partial point-set images, our objective is to reconstruct the original configuration by placing the points back into their correct positions, restoring any missing points, and maintaining the overall geometric structure of the set. This task is akin to an image inpainting problem, where the model must learn to generate missing information based on the observed portions of the image.

While the size of the dataset for training the model is important, we have found that generating point-set images remains a bottleneck. Currently, we can generate up to 30 valid point sets per hour, which, while substantial, is still relatively slow compared to the amount of data required to train a transformer model without overfitting. With the goal of training a model that generalizes well across different configurations, we need a much larger dataset. Therefore, finding ways to speed up data generation or to develop techniques for synthetic data augmentation will be crucial in overcoming this limitation.

Proposed Transformer-Based Approach

We explored using transformer models for point-set completion and image inpainting tasks, as these models are capable of capturing global dependencies within images, which is essential for understanding the relationships between points in a set. Two promising approaches were identified for applying transformers to this problem:

Masked Image Modeling (Inpainting)

The first approach involves using a transformer model to perform masked image modeling, which is commonly used in image inpainting tasks. Here, the model learns to reconstruct missing or corrupted pixels from the observed parts of the image. We propose to use a Vision Transformer (ViT) or Masked Autoencoder (MAE) architecture, which has been highly effective in image completion tasks.

In this setup, we randomly mask portions of the point-set image (by setting some of the black pixels to white) and train the transformer model to predict the correct positions of the masked pixels. The model learns to utilize the spatial structure of the image to infer the missing information. The key advantage of using transformers for this task is their ability to capture long-range dependencies between pixels, which is particularly useful when dealing with configurations where the correct placement of one point might be influenced by others across the image.

Once the model is trained, it can infer the correct locations of points from corrupted or incomplete point-set images in constant time. This constant-time inference is a significant improvement over traditional combinatorial methods, as the model can quickly generate a solution with a single forward pass.

Conditional Diffusion Models

The second approach we considered is based on the idea of conditional diffusion models, which have recently shown promising results in image generation and inpainting tasks. These models work by iteratively refining a noisy image into a complete image through a series of steps, conditioned on the known parts of the input. The diffusion process gradually reduces the noise, progressively improving the quality of the reconstruction.

In our context, a transformer-based diffusion model would start from a noisy or corrupted version of a point-set image and iteratively refine it to recover the true configuration. During the refinement process, the model learns to fill in missing points, correct misalignments, and restore occluded regions. By conditioning on the partially visible parts of the image, the model can efficiently restore the missing information.

While diffusion models involve multiple steps, the total computational cost remains polynomial in the number of pixels, and once trained, the model can generate point sets in near-constant time. This approach offers an additional advantage in that it may help preserve the geometric structure of the point sets more accurately during the reconstruction process.

Both of these approaches rely on transformer models that can be fine-tuned on our specific task. However, the main challenge lies in obtaining a sufficiently large dataset for training without overfitting. Given the current rate of data generation (30 sets per hour), we are limited in the amount of data available for training. This makes it difficult to train a robust model capable of generalizing across various point-set configurations. As such, accelerating data generation or implementing advanced data augmentation techniques will be crucial for the success of this approach.

Evaluation Strategy

The evaluation of the transformer-based model will primarily focus on the accuracy of the reconstructed point configurations. The key metric will be the **total Euclidean distance** between the predicted and true point locations. This metric directly measures the discrepancy between the predicted and correct positions, with a lower value indicating higher accuracy.

Additionally, we will evaluate the model's performance in terms of **binary success rate**, which will be the proportion of test cases where the model correctly matches all point locations within a small tolerance. Achieving a high success rate will indicate that the model is effectively learning the geometric structure of point configurations and is capable of accurately reconstructing the set.

Challenges and Considerations

Several challenges must be addressed in the application of transformer models to this problem:

- **Data Scarcity and Overfitting:** Generating a large labeled dataset of point configurations remains a challenge. With the current generation rate of up to 30 point sets per hour, we face limitations in training the model without risking overfitting. Expanding the dataset through synthetic augmentation or efficient data generation techniques will be crucial for training a generalizable model.
- **Geometric Consistency:** The model must adhere to the geometric constraints of valid point configurations. Point sets must be in general position (no three points are collinear) and should maintain convexity and other geometric properties. Ensuring the model's outputs respect these constraints remains a challenge. We will need to explore ways to incorporate these rules into the model, either during training or through post-processing.
- **Positional Encoding and Spatial Structure:** Transformers do not naturally encode the spatial relationships between points in the same way that convolutional networks do. We will need to experiment with positional encodings or hybrid CNN-transformer architectures to improve the model's ability to preserve spatial consistency.

Future Work

In the coming weeks, we will focus on the following directions:

- **Enhanced Dataset Generation:** We will explore ways to speed up the generation of point sets, such as parallelizing the data generation process or using procedural generation techniques to create more diverse configurations.
- **Model Architecture Exploration:** We will experiment with different transformer architectures, including hybrid CNN-transformer models, to improve the model's ability to handle spatial dependencies.
- **Geometric Constraints in the Model:** We will investigate methods for incorporating geometric constraints directly into the transformer model, such as through specialized loss functions or constraints during training.

In conclusion, the transformer-based approach has significant potential to improve the speed and accuracy of point-set validation in the Happy Ending problem. While data generation remains a bottleneck, we are optimistic that the model will provide substantial improvements once we overcome these challenges.

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 1.155×10^{-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

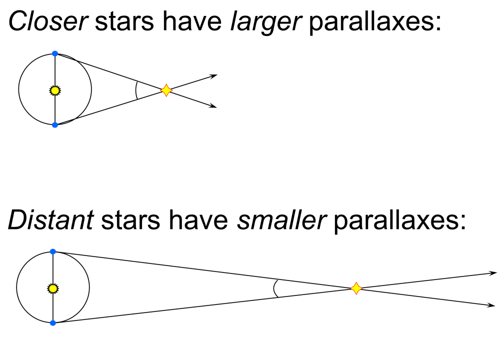
Closer stars have *larger* parallaxes:
Distant stars have *smaller* parallaxes:

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.

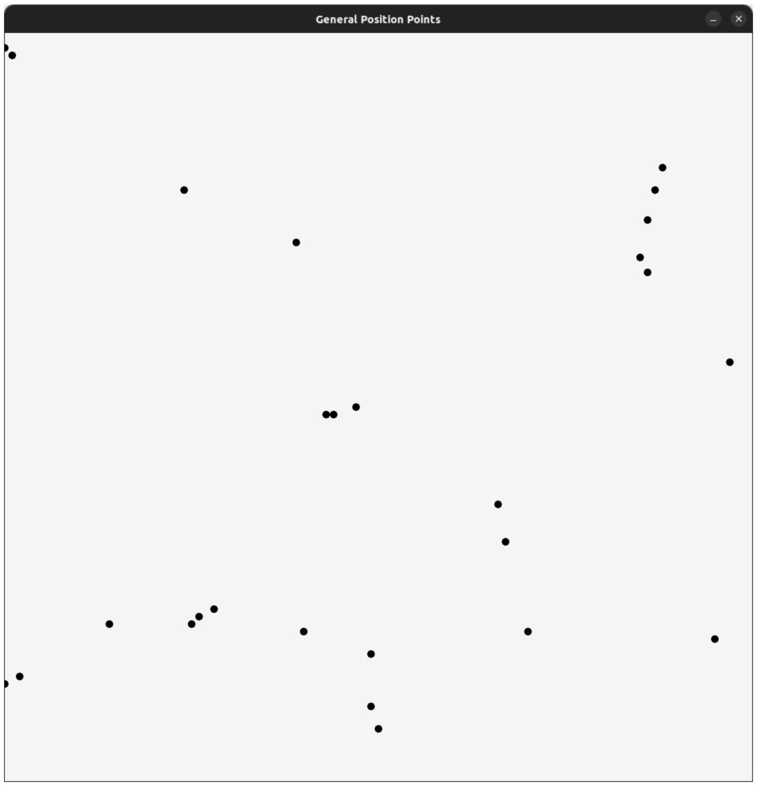

Figure 2: Set generated on a 100×100 grid.

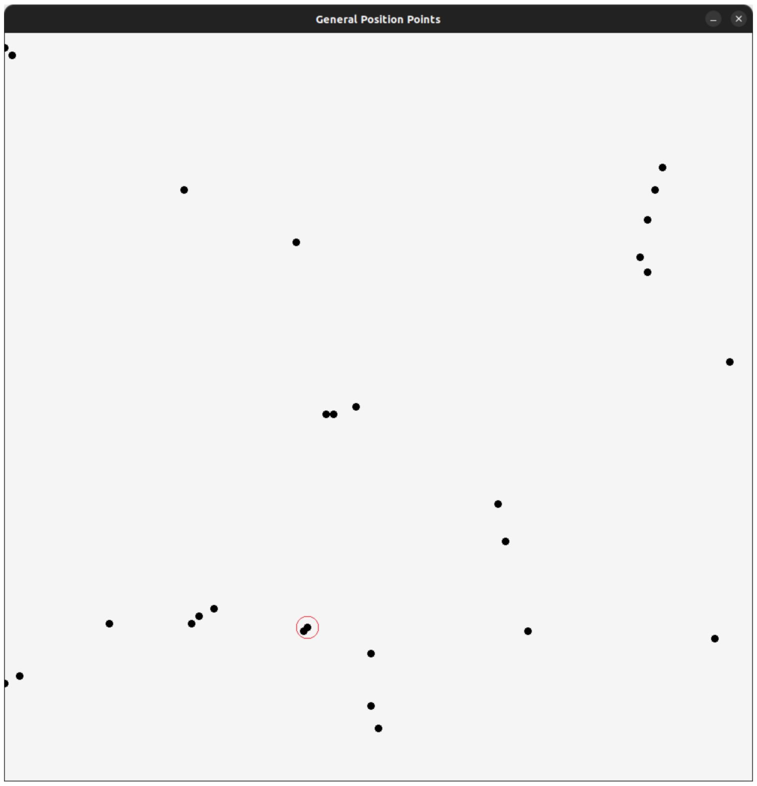

Figure 3: Same set after scaling to a 200 × 200 grid, showing the newly added point highlighted with the red circle.

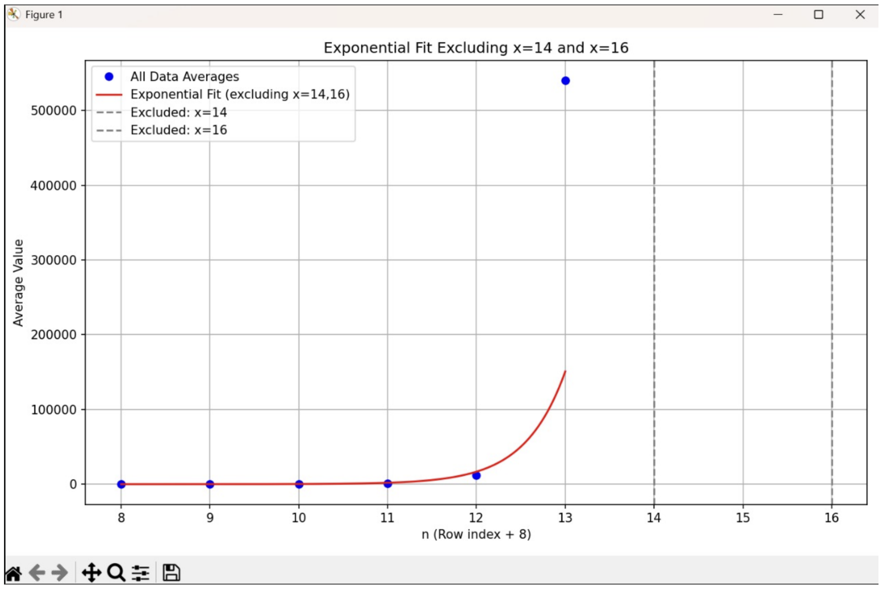

Figure 4: Exponential fit diverging from real data

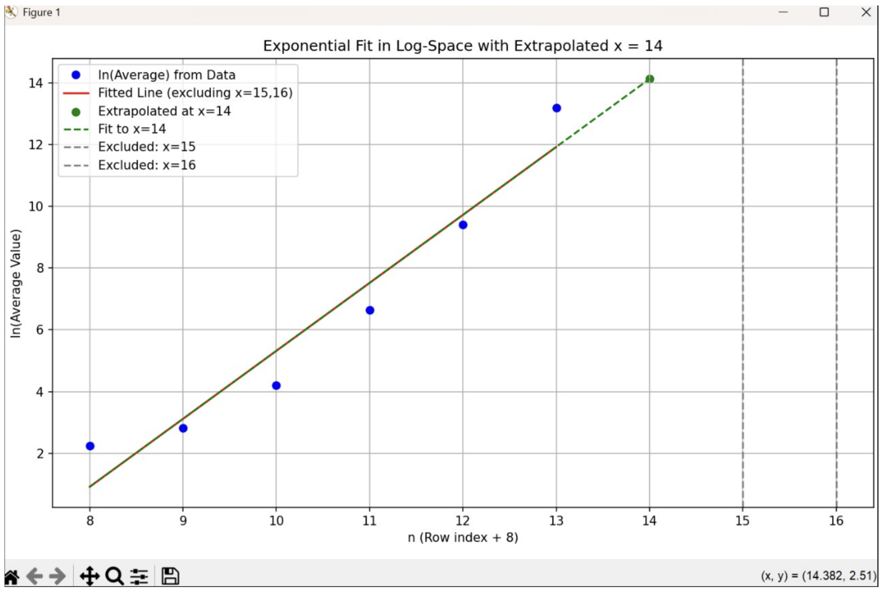

Figure 5: Linear fit misrepresenting the data structure

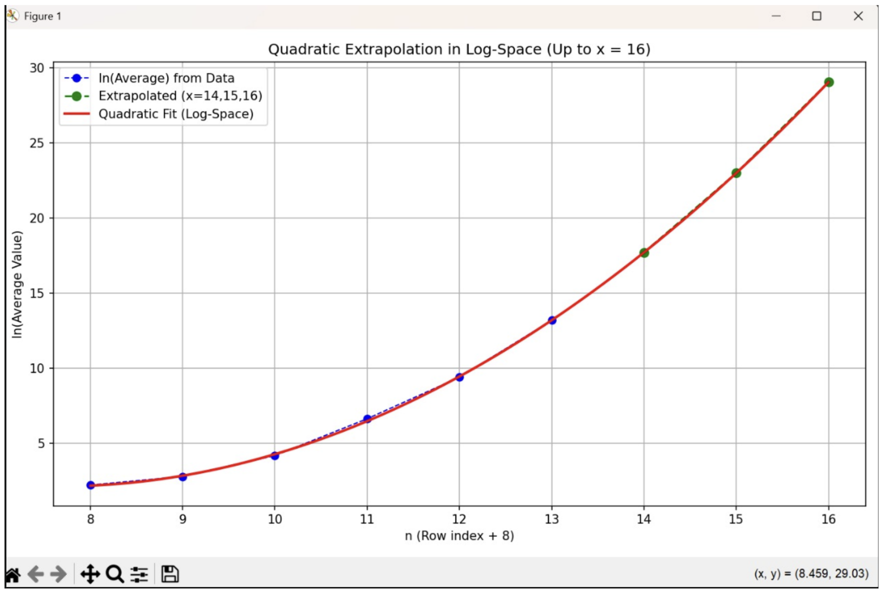

Figure 6: Quadratic fit in log-space indicating super-exponential growth

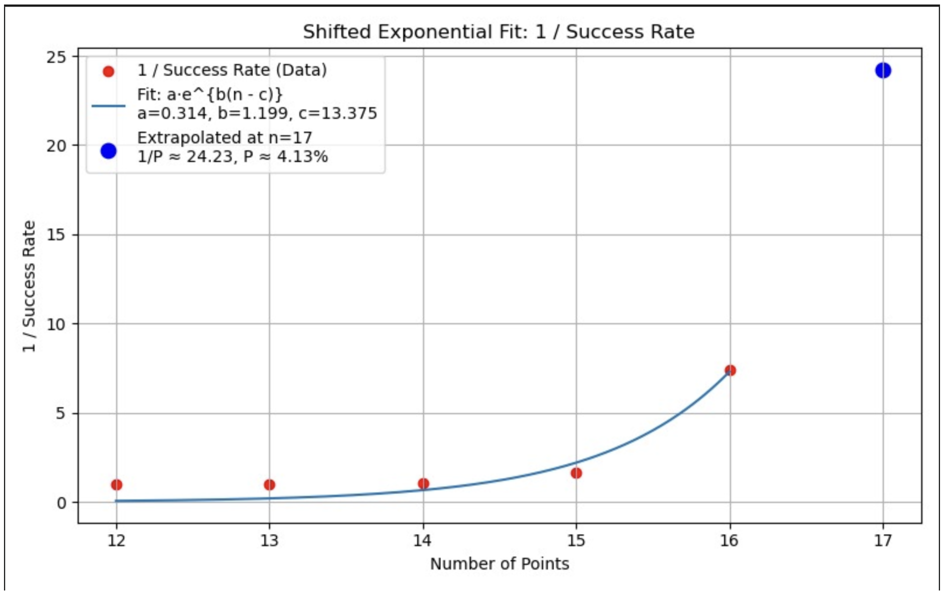

Figure 7: Shifted Exponential Fit: 1 / Success Rate