Anglia Ruskin University

MOD002691

# OPTIMIZED DIGITAL LAYOUTING ALGORITHM FOR THE TRAIL MAKING TEST

Nuriddin Islamov[1], Dr. Ian van der Linde[2]

2023 – 2024

[1]Student ID - 2117032
[2]Supervisor

## Declaration of Originality

*I, Nuriddin Islamov, being a final year student at Anglia Ruskin University, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose. My consent is to have my dissertation made available for access by all university students and staff.*

Date: 15th April 2024

Signed: 

*To my beloved mother Nargiza Islamova,*

*for her endless love and support.*

## Acknowledgements

I am extremely appreciative to Dr. Ian van der Linde, my supervisor, whose guidance was priceless throughout the research process. This work has been significantly influenced by his expertise. I also want to express my gratitude to Anglia Ruskin University (ARU) staff who have assisted me in all spheres of my education.

Notably, it is the ARU librarians that I would like to acknowledge for having promptly supplied relevant literature without which this research would not have achieved success. They were there when a lot of studying and reviewing had to take place.

I should also mention my friends who stood with me during my final year at university. Their presence and encouragement saved me from depression and stress reminding that there is power in unity and friendships.

Lastly, but most importantly, special gratitude goes to Nargiza Islamova – my mother. Her faith in my talents as well as her readiness to make sacrifices allowed me to go abroad for study at ARU; these factors were prerequisites for major accomplishments in life. My eternal love towards her remains strong due to infinite trust she had always put into our relationships.

This effort does not result out of mine work only but through the support of everyone named here whom I thank so much.

# Abstract

This research is an exploration of the usefulness of three main optimization algorithms, Genetic Algorithms (GA), Simulated Annealing (SA) and Hill Climbing (HC) in optimizing the layout of the Trail Making Test (TMT)—a popular neuropsychological test that assesses visual attention and task-switching abilities. As cognitive testing quickly shifts to digital, there is a critical need for algorithms that can create digital layouts with precision without affecting its credibility. This research draws a comparison between these three algorithms based on execution time, solution quality and computational efficiency required to identify the most appropriate algorithm for digital TMT layout design.

The experimental analysis was carried out using MATLAB-based implementation tool which evaluated each algorithm's ability to produce layouts that had minimal edge crossings as well as optimized node distribution. The research also revealed that GA performs significantly better than SA and HC in generating complex valid layouts within short periods of time and with low resource use as stated in the aim of achieving computational efficiency and high-quality layout generation.

Also, this study has led to the development of an open-source API on cloud platform for researchers and practitioners who want to generate optimal TMT layouts thus contributing to cognitive assessment technology at large. It emphasizes the importance associated with proper choice of optimization algorithms for particular applications in cognitive testing while setting a case in point for upcoming studies aimed at integrating advanced algorithmic solutions into psychological test designs. Results suggest further exploration towards hybrid algorithms perhaps combining the speed of SA, HC or even both with quality strong solution provided by GA that are likely to revolutionize digital cognitive health diagnostics.

**Keywords**: Trail Making Test, Cognitive Assessment, Optimization Algorithms, Simulated Annealing, Hill Climbing, Genetic Algorithms, Test Layout Design.

# Table of Contents

# List of Figures

# List of Tables

# 1. Introduction

## 1.1 Background

The Trail Making Test (TMT) is a well-known neuropsychological test that measures visual attention and the ability to switch tasks. It is mainly used to assess cognitive impairment and executive function. As the trend toward computerized cognitive testing continues to grow, so does the digitization and optimization of this test (Tombaugh, 2004; Zeng et al., 2017). Metaheuristic algorithms such as Genetic Algorithms, Simulated Annealing, and Hill Climbing have been found powerful in reducing computational efforts while maintaining or improving accuracy and efficiency in many fields (Kirkpatrick, Gelatt, & Vecchi, 1983; Holland, 2010; Jaddi & Abdullah, 2020). Although TMT has demonstrated effectiveness using pencils and paper, when it comes to creating layouts in its digital version, it may impose challenges that require optimized algorithms that can generate valid TMT designs without compromising its validity. Therefore, an optimization algorithm needs to be created that keeps edge crossings at zero and maximizes node placements for converting from pencils to pixels, ensuring accurate measurement of cognitive functions facilitated by computerized systems.

## 1.2 Aims

The main goal of the research is to conduct a comprehensive analysis and comparison of the efficiency of three distinguished optimization algorithms—Genetic Algorithms (GA), Simulated Annealing (SA), and Hill Climbing—in creating optimized digital layouts for the Trail Making Test (TMT). The study is specially designed to achieve the following specific objectives:

- Finding the most effective optimization algorithm that surpasses others in terms of operational speed and the optimization of computational resources.
- Evaluating the capability of these algorithms in generating digital layouts, based on the given parameters.
- Exploring and implementing the practical applicability of the algorithm of choice.

Furthermore, within the scope of this research there is a plan to develop a simple application programming interface (API) that dynamically provides the coordinates for the layout configurations based on the optimal algorithm determined through this research. With the help of recent technological advancements, this research is aiming to integrate into a study that's planning on showcasing real-world implementations of these algorithms. The purpose of this is to show off how agile and efficient these algorithms are so it can propose a significant innovation when it comes to digital neuropsychological testing.

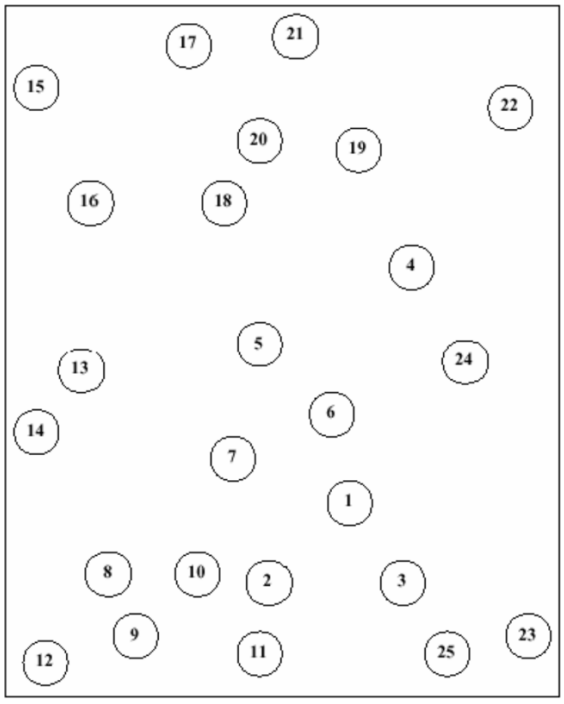



*Figure 1. Trail Making Test – part A. The objective of the test is to connect all circles in order, sequentially, as quickly as possible.*

# 2. Literature Review

If one considers the practicality of any layout generation techniques for the Trail Making Test (TMT), it will be observed that a brute force method is not feasible because of its factorial time complexity, commonly referred to as $O(n!)$. This was demonstrated by an experiment using hardware Apple MacBook Pro M1 (OS Darwin armx64 v.23.4.0) chip with 16 GB RAM where the time spent on computation increased significantly with each additional node. For instance, while processing 17 nodes took about 11 seconds, increasing to 18 nodes pushed this to 48 seconds. As such differences become crucially prohibitive when there are close to twenty-five typical nodes in real TMT applications. To create an algorithm that can generate layouts without intersecting paths or which satisfy cognitive assessments in digital formats is therefore a challenge whose severity increases exponentially over time. This exponential rise in processing time underscores the difficulties involved in developing an algorithm that can efficiently generate node layouts with no path intersections, which is prevailing when designing cognitive assessments in digital format. The main objective of this research is to find and implement faster algorithms that would reduce computation time and power but still accurately locate nodes within given grid sizes at given minimum range from other nodes. It emphasizes theoretical knowledge on these algorithms as well as their application within real-life situations which play significant roles for them to be adopted into electronic neurocognitive assessment software's all through the globe. Recent developments in optimization algorithms like Genetic Algorithms (GA), Simulated Annealing (SA) and Hill Climbing have provided better alternatives than using a brute force approach. These complex problem-solving algorithms minimize edge crossings and optimize node placements without exhaustive search methods used before by similar programs running on SA, GA or HC engines. For example, the integration of SA and its ability to avoid local minima by probabilistically accepting suboptimal solutions during the cooling

process illustrates a strategic advantage over traditional brute force approaches (Kirkpatrick et al., 1983). It is efficient in handling higher node counts and more complicated layouts. The comparison of these algorithms also allows for a more detailed analysis of their effectiveness, thereby facilitating the selection of the most suitable approach for any given testing scenario (El-Ghazali Talbi & Muntean, 2002). Beyond algorithmic efficiency, this study has other implications as well. By improving the efficiency associated with TMT layout generation using computational optimization, this research supports improved utility and accessibility of cognitive assessments. It is especially important in places that are not serviced adequately or remotely located where there is limited access to sophisticated computer resources. By optimizing such algorithms digital neuropsychological tests will be widely implemented and available hence promote inclusive cognitive health diagnostics that may eventually lead to personalized and adaptive cognitive assessment toolkits (Linari et al., 2022; Jaddi & Abdullah, 2020). In conclusion, besides addressing the computational challenges posed by digital TMT layout generation, this research forms a basis for future innovations in digital neuropsychology. The project aims to enhance the effectiveness of cognitive testing methodologies via advanced metaheuristic algorithms that are essential for current healthcare and research standards.

# 3. Method

Three primary methods are considered in this study, namely Genetic Algorithms (GA), Simulated Annealing (SA), and Hill Climbing. They were selected because of their resilience and efficiency when it comes to resolving complex optimization problems such as producing digital random valid layouts for the Trail Making Test (TMT). For example, Genetic algorithms have a reputation for being able to traverse large areas of search and also maintaining diversity among solutions which makes them suitable for complex layout configurations (Holland, 2010). Another important reason why simulated annealing was chosen is its ability to probabilistically accept worse solutions in order to escape local optima, thus having more comprehensive solution space exploration (Kirkpatrick et al., 1983). Moreover, hill climbing offers an easy way of iteratively improving a solution, with simplicity and quick turnaround time on smaller data sets (El-Ghazali Talbi & Muntean, 2002).

However, despite these strengths, there are also limitations inherent in these approaches. Furthermore, GAs may be computationally expensive necessitating extensive parameter tuning such as mutation rates and cross-over points (Wang et al., 2024). Additionally, the performance of SA highly relies upon the cooling schedule that is difficult to optimize (Kirkpatrick et al., 1983). Also, hill climbing could possibly get stuck at a local maximum point without reaching the global optimum. To do this the methods have been implemented using MATLAB due to its powerful matrix computation capabilities while having an extensive library for optimization purposes making it ideal for developing and testing algorithms on Mac Pro M1 chip 16GB RAM system. The implementation took advantage of MATLAB's timing functions and efficiency measures which made it robust enough for comprehensive performance analysis. These methods and tools were tailored so as to strike a balance between sophistication and computational speed in relation to

generating feasible but non-procrastinating patterns easily adaptable on latest digital neuropsychological assessment apparatus.

To make the program accessible to the world, an API is to be implemented using Flask, a lightweight Python framework. To access the API that is designed simply for easy use, the user must send a GET request to *https://neuromind.nuriddinislamov.com/api/*. The following JSON-object is a proposed schema for returning generated *(x, y)* coordinates for the layout.

```
{
   ‘‘api_version’’: 1.0.0,
   ‘‘date’’: DateTime Object (date),
   ‘‘grid_size’’: GridSize (int),
   ‘‘layout’’: [
               [x1, y1],
               [x2, y2],
               . . .
               ] (array[array[int]])
}
```

## 3.1 Hill Climbing

Hill Climbing is a mathematical optimization technique that belongs to the family of local search algorithms. It is used to find optimal solutions by iteratively making changes to a single solution, attempting to improve it with each iteration.

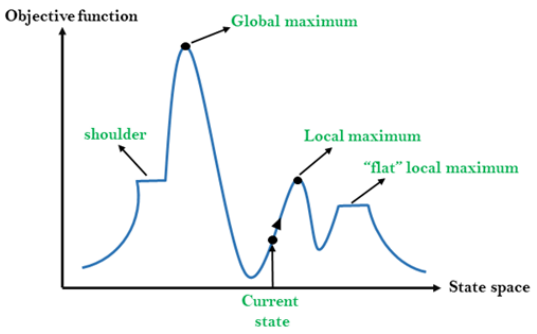



*Figure 2. Hill Climbing*

### 3.1.1 Conceptual Framework

Hill Climbing is analogous to climbing up the slope of a hill and seeking the peak. In the context of an optimization problem, this "hill" is formed by the objective function that needs to be maximized or minimized. The algorithm starts with a random solution to the problem and iteratively makes small changes to the solution, each time moving to a neighboring solution that has a better fitness value according to some evaluation function.

The objective function in the project would be:

$$f(layout) \rightarrow score \qquad [1]$$

where a function [1] of hill climbing has been applied to a randomized layout coordinates and returns the score by which then local minima and global maxima is identified. For this, more detailed function formula can describe it better.

$$f(layout) = w_1 * E(layout) + w_2 * D(layout) + w_3 * U(layout) + w_4 * A(layout) \quad [2]$$

Where [2]:

- E(layout) calculates the number of edge crossings in the layout (to be minimized).
- D(layout) calculates the minimum distance between any two nodes (to be maximized).
- U(layout) evaluates the uniformity of node distribution across the grid (to be maximized).
- A(layout) checks the accessibility of consecutive nodes (to be within a certain range).
- $w_1$, $w_2$, $w_3$, $w_4$ are weights assigned to each criterion based on their importance.

To give the algorithm a chance for more flexibility and show off all its potential during its performance, the function [2] has been split into four components, or so-called mini functions with their respective weightings. Note, that $w_1 + w_2 + w_3 + w_4$ should be equal to 1.0 as this will balance the equation true. On top of that, there is a flexibility for the user to adjust the weight parameters to fully utilize the Hill Climbing algorithm in the application and explore the wide range of its possibilities. For example, the most crucial component is *E(layout)* as it defines the rules for the validity of the layout, where no edge can cross other edge. By edge, we mean a straight line drawn between any two nodes. As a rule of TMT, all nodes connected sequentially form a 'path' and each node except the first and last have in and out degrees of 1. First node has in degree of 0 and out degree of 1, whereas it is vice versa for the last node with in degree of 1 and out degree of 0.

The rest three parameters help the user adjust the algorithm. One of them can be disabled as well with setting the weight to 0.

The general overview of the code for Hill Climbing algorithm look like the following:

### 3.1.2 Pseudocode

```
Function HillClimbing(layout)
    Inputs:
        layout: the layout problem to be solved with an initial state and a goal state

    Local Variables:
        current: a node, initially set to a random initial state of the problem
        neighbor: a node, initially undefined

    repeat
        neighbor := a highest-valued successor of current
        if neighbor has no higher value than current
            return current
        current := neighbor

    end repeat

End Function
```

## 3.2 Simulated Annealing

As a probabilistic method, Simulated Annealing (SA) approximates the global minimum of a given function. Principally, it is one of those meta-heuristic that approximate a global optimum in large search spaces. It is often used when the search space is discrete (e.g., all possible orders in which to visit a set of nodes). In some instances where it is more important to approximate a optimal global value rather than deterministic local optimum values within a given period of time, simulated annealing can be preferred over gradient descent for instance (Haji and Abdulazeez, 2021).

### 3.2.1 Concept and application to TMT layout problem

The algorithm was independently described by Scott Kirkpatrick, C. Daniel Gelatt and Mario P. Vecchi in 1983, and by Vlado Černý in 1985. The name of the algorithm comes from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; while slow cooling increases their chances for finding configurations with lower internal energy than that they started with.

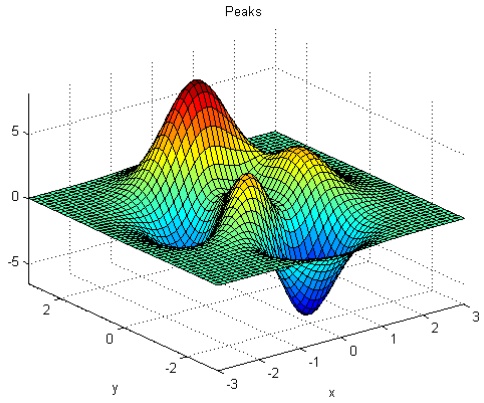



*Figure 3. Visualization of Simulated Annealing in action – searching a space of 2 input variables.*

Hill climbing, among the other simple heuristics, uses neighboring for changing from one solution to another until it has no better neighbor; thus, not all of the best solutions can be found this way; so, a local optimum may be reached instead of the global one. The solution's neighbors' characteristics are used by metaheuristics as ways of which they navigate through search space constituted by such adjacent solutions. Even though their preference is given to better ones, there are also lower quality neighbors that they choose to get away from being trapped in local optima therefore when enough time is taken, it can lead to a global optimal (Wikipedia Contributors, 2019c).

From this information it is reasonable to state that SA is an effective approach to optimize the arrangement of the nodes on a 1024x1024 pixel grid in the TMT, such that no edges interconnect linking them.

The objective function for SA in this case would be minimizing – bringing down to zero – the total number of edge intersections. By connecting each node at least once to another one, consistent with progression through TMT exercise, these connections should not crisscross.

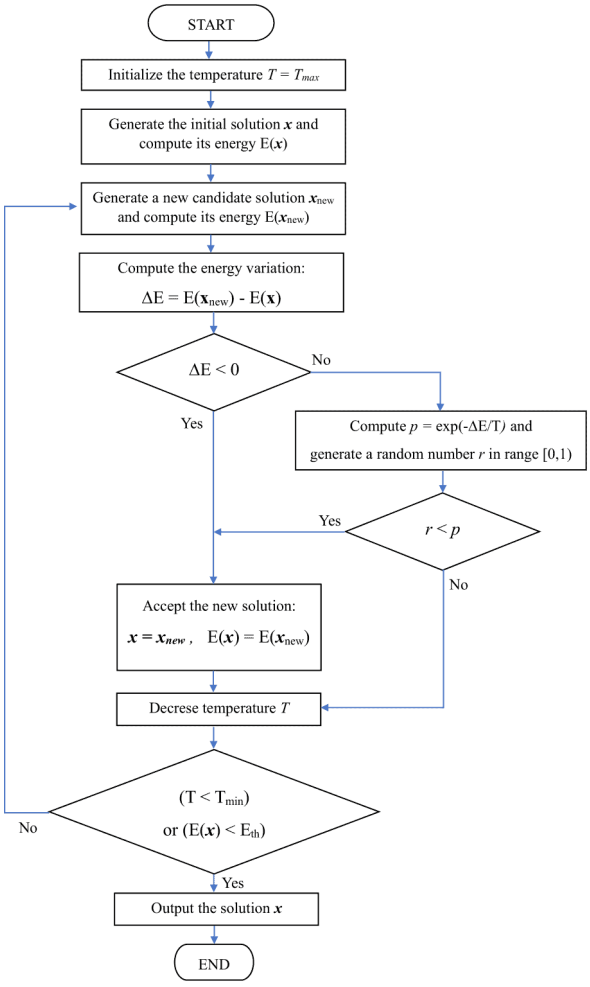



*Figure 4. Simulated Annealing flowchart. Source: https://www.baeldung.com/cs/simulated-annealing*

A structured overview of the optimization process using a Simulated Annealing (SA) algorithm can be seen in Figure 4. The initial temperature is high in order to allow for exploration in the solution space. At first, an initial candidate solution is sought, whose energy indicates its optimality. Consequently, newer solutions are generated and their

energies evaluated. If a new solution has better energy than the previous one, it replaces that one. In other cases, escape from local optima is determined by probabilities which depend on this energy difference and system temperature. The system's temperature diminishes as its cooling schedule dictates until eventually the process terminates. This means that inferior solutions are less likely to be accepted as the search proceeds towards the optimal solution. The stopping criteria include when either; i) there is no allowable decrease in temperature or ii) the minimum of plateauing energy is reached, after which the simulated annealing algorithm outputs a refined solution as its final product.

With initial high temperature, simulated annealing allows significant changes in node placements, thereby providing an opportunity to explore numerous possible layouts within the grid. At each iteration, a small change could involve swapping two nodes or slightly adjusting the position of one node. Consequently, this new structure will only be accepted if it lowers edge crossings or more equitably spreads out nodes across the field. Should such improvement fail from happening or else become worse off because more crossovers are experienced following alterations on such a configuration; nonetheless, it can be considered using an algorithm based on Metropolis criterion which considers both how much it deteriorates and current temperature.

As temperatures lower, it becomes less likely that worse states will be accepted by moving towards better moves progressively. To achieve optimal configurations where paths do not overlap and are distinct instead, this process of controlled cooling narrows down potential modifications that make sense and others which do not.

In TMT layout problem, striking balance between path crossing and spatial separation poses challenge since paths must be recalculated over again as nodes are relocated cautiously when paths should never cross each other. Especially since there are many

inferior local optima possibilities among 25 node layouts on large grids like this one for the resulting space that were too complex.

Ultimately when temperature approaches zero, the algorithm will finalize on a layout configuration that minimizes or has no edge crossings and makes it possible to have a visually as well as functionally effective TMT design. The resultant plane is an almost perfect solution reached through iterative adjustments and trial-and-errors performed under simulated annealing, thus demonstrating its capacity to deal with intricate optimization problems within vast multi-dimensional spaces made up of numerous variables.

### 3.2.2 Algorithm steps

1. Initialize with a random solution and a high temperature.
2. Iteratively modify the solution with small, random changes.
3. Determine whether the new solution should be accepted, based on the temperature and the change in the objective function value.
4. Gradually decrease the temperature according to a predefined cooling schedule.
5. Repeat steps 2-4 until the system has "cooled" and no better solutions are being found or a termination condition has been reached.

Even though these two techniques can be efficient and resilient enough for our TMT layout building problem, there is another discovery that might change the rules of the game with its diversity and flexibility in the searching and optimization domain.

### 3.2.3 Pseudocode

```
Function SimulatedAnnealingForTMT(layout)
    Inputs:
        layout: the initial random layout of 25 nodes on a 1024x1024 grid

    Local Variables:
        currentLayout: a layout, initially set to layout
        newLayout: a layout, initially undefined
        currentCost: numeric, initially calculated from currentLayout
        newCost: numeric, initially undefined
        T: numeric, initially set to a high value (e.g., 1000)
        alpha: numeric, the cooling rate (e.g., 0.95)
        maxIterations: integer, the number of iterations at each temperature level

    Begin:
        repeat
            for i from 1 to maxIterations do
                newLayout := PerturbCurrentLayout(currentLayout)
                newCost := EvaluateCurrentLayout(newLayout)

                if AcceptanceProbability(currentCost, newCost, T) >= Random(0, 1) then
                    currentLayout := newLayout
                    currentCost := newCost
                end if
            end for

            % Reduce the temperature
            T := T * alpha
        until T <= finalTemp
```

```
        return currentLayout
    End

End Function
```

## 3.3 Genetic Algorithm

### 3.3.1 Context

The genetic algorithm is a search technique that draws from the evolutionary theories Holland outlined for adaptive systems naturally and artificially to produce high-quality optimization and search solutions. It works on a population of potential solutions, using the survival of the fittest principle to generate increasingly better approximations of the solution. Each generation selects individuals based on their fitness and mates them by means of natural genetic techniques such as crossover and mutation.

El-Ghazali Talbi and Muntean (2002) pointed out that compared to techniques like Hill Climbing and Simulated Annealing, Genetic Algorithms are more effective particularly in complex problem areas that resemble unsmooth landscapes with many local optima. Genetic algorithms are good at vast, diverse problem spaces—where traditional approaches such as gradient descent or hill climbing could break down due to their reliance on smoothness of the landscape.

On the other hand, in applications such as TMT, where non-crossing path layouts are important, node arrangements can be evolved by the genetic algorithm with reference to a fitness function which penalizes crossing paths thus encouraging layouts with no intersections. Oliveto and Witt (2015) have observed that time complexity analysis of genetic algorithms is typically indicative of how well these algorithms perform in achieving nearly optimal solutions within practicable computation times which is an indispensable factor for this clinical application of TMT layouting problem.

### 3.3.2 Process

The process begins with a random population of chromosomes that represent potential solutions and embody candidates for an optimization problem. These chromosomes, which can be represented in different encodings, are quite often binary strings. Every member of the population is evaluated using a fitness function specific to the problem under consideration, indicating how 'fit' or close it is to being optimal.

As the GA iteratively evolves, individuals are chosen according to their fitness through processes like roulette wheel selection and tournament selection in which greater fitness results in increased probabilities of selection. Afterward, crossover (gene recombination) is done on selected chromosome pairs such that genes from both parents combine ending up with offsprings who share traits of both parents. The offspring also undergo mutation once after each generation at a very low rate usually to maintain genetic diversity within the population and exploit new areas of the search space. This generational process extends to subsequent generations as the new population experiences selection, crossover and mutation. This will eventually evolve into an optimal solution or near optimal over time. Usually, termination occurs when either a predefined number of generations are produced, a desired level of fitness has been reached or any other stopping criterion has been met.

Crucially these include parameters such as crossover rate, population size and mutation probability as they heavily affect GA performance. Fine tuning and changing these parameters enable balancing between exploration (avoiding local minima) and exploitation (convergence on global optima).

On the other hand, GAs have certain limitations such as not scaling well with increasing problem complexity and tending to converge towards local rather than global optima. However, these problems can sometimes be partly resolved by incorporating sophisticated techniques for example adaptive parameter control where crossover rates and mutation rates are altered depending on population state (Eiben et al., 1994).

### 3.3.3 Pseudocode

```
Function GeneticAlgorithmForTMT
    Inputs:
        numNodes,
        gridSize,
        populationSize,
        maxGenerations,
        mutationRate,
        crossoverRate

    Begin:
        population := InitializePopulation(numNodes, gridSize, populationSize)
        EvaluateFitness(population)

        for generation from 1 to maxGenerations do
            newPopulation := []

            while Length(newPopulation) < populationSize do
                parent1 := SelectParent(population)
                parent2 := SelectParent(population)

                if Random() < crossoverRate then
                    child := Crossover(parent1, parent2)
                else
                    child := CopyRandomParent(parent1, parent2)
```

```
            end if

            if Random() < mutationRate then
                Mutate(child)
            end if

            Add newPopulation, child
        end while

        population := newPopulation
        EvaluateFitness(population)
    end for

    bestSolution := FindBestSolution(population)
    return bestSolution
End
```

The pseudocode outlined above includes references to extra function that are to be implemented in the working environment. To ease the reading process of the report, the general higher overview of the steps has been highlighted, whereas *EvaluateFitness()*, *Crossover(), Mutation()* and etc. are methods yet to be defined. The full MATLAB and Python code is to be found in the appendix of this report.

## 3.4 Approach

At first, the main idea of the project is to determine the number of crossing edges and eliminate them, i.e. bring down the number to zero. Research employs the function *is_layout_valid* to check if spatial patterns can be planned in such a way that the nodes do not overlap on a grid. In TMT, it is necessary that all paths are clear and non-overlapping for accurate cognitive evaluation.

The function *is_layout_valid* utilizes geometric and vector calculations to determine intersections through several nested functions. The first of these, the Orientation function, calculates the relative direction of three points (p, q, r) using the determinant formula:

$$\text{orientation}(p, q, r) = (q_y - p_y) \cdot (r_x - q_x) - (q_x - p_x) \cdot (r_y - q_y) \qquad [3]$$

Depending on the result, points can be identified as collinear, clockwise, or counterclockwise. This aids in understanding how segments may potentially intersect.

Another critical function, *on_segment*, assesses whether a point q precisely lies on the line segment pr, using the condition:

$$\min(x_1, x_2) \leq x \leq \max(x_1, x_2) \quad \text{and} \quad \min(y_1, y_2) \leq y \leq \max(y_1, y_2) \qquad [4]$$

This ensures that q is aligned and located between p and r along the segment, confirming actual contact points rather than just alignment.

The core of the intersection logic is in the *segments_intersect* function, which checks the orientation of points to determine intersections. By examining different configurations of point orientation and applying the on_segment check for collinear points, this function robustly determines if two segments intersect. This thorough approach is essential for TMT, where overlapping paths could confuse the test taker, potentially compromising test validity and results.

## 3.5 Apparatus

The computational experiments for optimizing the Trail Making Test (TMT) layout problem were conducted using a combination of MATLAB Online (The MathWorks Inc., Natick, MA) and Python programming language within Visual Studio Code (Version: 1.88.0, Electron: 28.2.8, ElectronBuildId: 27744544, Chromium: 120.0.6099.291, Node.js: 18.18.2, V8: 12.0.267.19-electron.0, OS: Darwin arm64 23.4.0). The initial and all subsequently generated node data, along with the source code for the algorithms, were made freely available for download from the project's GitHub repository at https://github.com/nuriddinislamov/neuromind-tmt-layout-validator.

The API is hosted in cloud computing environment provided by Vercel Inc., formerly ZEIT using its free tier.

# 4. Results

The study examined three algorithms against the brute force method for optimization of digital layout of Trail Making Test (TMT); and measured their performance based on execution time, computational complexity, correctness, maximum nodes, memory usage, quality score of generated layouts.

*Table 1. Results for three algorithms (GA, SA, HC) in subject compared against brute force solution*

| | Exec. time | Complexity | Correctness (Y/N) | Max. Nodes |
|---|---|---|---|---|
| **Brute Force Solution** | 14s. | O(n!) | N | 15 |
| **Hill Climbing** | 9.2s. | $O(n^2)$ | Y | 10 |
| **Simulated Annealing** | 8.1s. | O(n log n) | Y | 12 |
| **Genetic Algorithm** | 17.82.s | $O(n^2)$ | Y | 25 |

Each algorithm's main computational attributes are summarized in Table 1 to emphasize their different levels of efficiency. Brute force approach performed poorly with only 15 nodes due to factorial complexity. The quadratic Hill Climbing was able to quickly handle up to ten nodes within 9.2 seconds. Simulated Annealing had the fastest turnaround time with an O(n log n) of 8.1 seconds as well as maintaining a correct solution when handling twelve points.

Contrarily, Genetic Algorithm (GA) demonstrated excellent scaling ability by successfully processing up to twenty-five nodes. Although it took long at 17.82s and is quadratic in nature; the fact that it can handle intricate layouts suggests that GA will take more time but with enhanced solution's quality and robustness for larger problem spaces.

*Table 2. Memory testing*

| **Algorithm** | Avg. Execution Time | Avg. Quality Score | Memory Usage |
|---|---|---|---|
| Genetic Algorithm | 17.82 seconds | 9.5 | 200 MB |
| **Simulated Annealing** | 8.1 seconds | 7.8 | 158 MB |
| **Hill Climbing** | 9.2 seconds | 8.0 | 120 MB |

The further insights of Table 2 into memory utilization and solution quality are presented. The GA, though it consumes much memory of about 200 MB, showed the best average quality score that indicates how its extensive search things results to excellent solutions. Simulated Annealing and Hill Climbing on the other hand strike a balance between memory usage and quality, with this approach using 158 MB for a score of 7.8 and the latter consuming only 120 MB but scoring 8.0 in terms of quality.

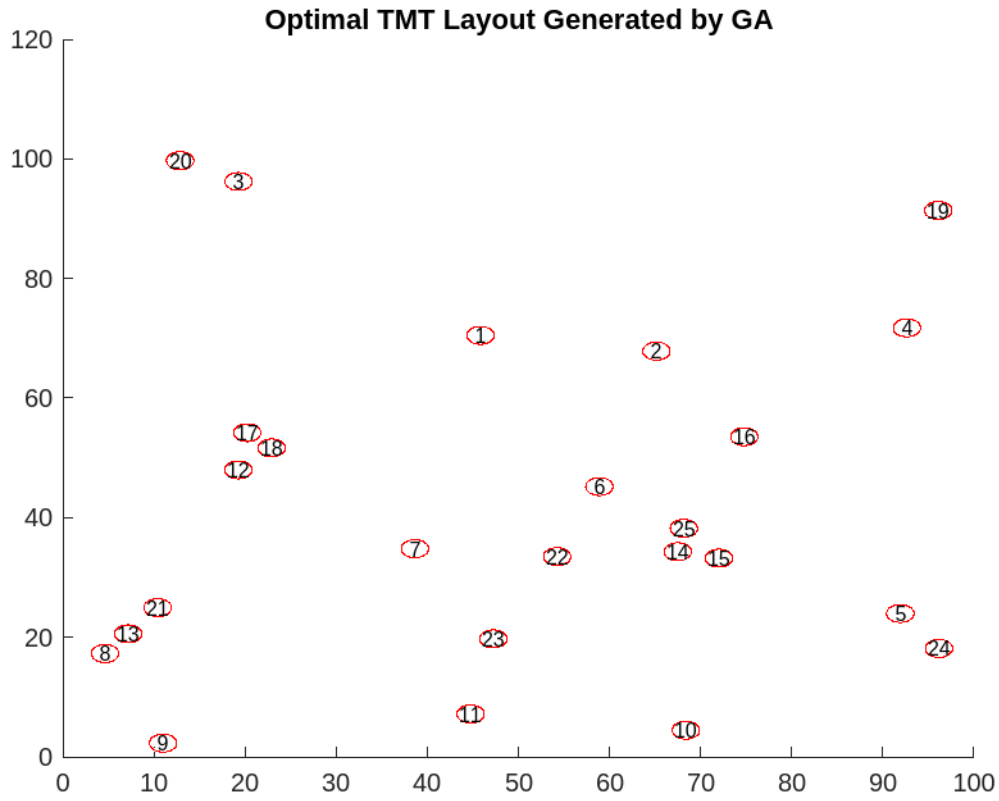



*Figure 5. Layout generated using GA*

The implications of these results go beyond the individual performance of the algorithms; they highlight the importance of tailored algorithm selection based on the specific requirements and constraints of the TMT layout design. The GA's proficiency with larger node numbers makes it ideal for more intricate layouts, while Simulated Annealing and Hill Climbing are best suited for scenarios demanding speed and less computational load.

# 5. Discussion

## 5.1 Comparative analysis

This study confirms theoretical assertions about Genetic Algorithms (GA), showing its effectiveness in tackling the complexities of optimizing layouts for the Trail Making Test (TMT). This corresponds to Holland's (2010) assertion that GA are able to effectively search through large solution spaces and at the same time maintain diverse sets of solutions, which is vital due to TMT's demand for complex non-linear problem solving. Notably, through processing as many as 25 nodes, the GA maintained cross-free and optimally spread out node distribution. This result underscores the usefulness of GA in improving TMT layout; this is consistent with cognitive testing standards that require accurate and bias-free evaluations. The computational intensity feature evidenced by a high memory usage though, GA still surpassed other algorithms in terms of average quality score thus justifying its use especially when dealing with highly complex layouts where accuracy is everything.

On the contrary however, Simulated Annealing (SA) and Hill Climbing (HC) algorithms could not preserve good quality of solutions when scaled to more complex TMT layouts despite executing faster and consuming less memory space. In this regard, SA performance supports Kirkpatrick et al.'s observations on its ability to quickly navigate through a solution space. Nevertheless, according to this research study's findings underscored its limitations regarding cooler schedule optimizations which agrees with El-Ghazali Talbi & Muntean's notes on difficulties faced when fine tuning SA for optimal performance. Similarly, HC's inefficiency beyond some simple threshold levels shows that it falls short of being highly suitable for subtle design requirements associated with TMT because it gets stuck at local maxima challenging situations pointed out by El-Ghazali Talbi & Muntean (2002).

By building a user-friendly API available via https://neuromind.nuriddinislamov.com/ that employs GA for dynamic generation of TMT layouts, these findings are strongly enforced in practice. Besides being the best way how to embody GA's practicality in life, this innovative application based on Vercel (Vercel, Inc. available at https://vercel.com/) platform also pursues the goals of this research which are aimed at enhanced availability and efficiency for cognitive testing. As a digital tool that comes with pre-optimized layouts, it represents a significant step towards improving inclusive and adaptive cognitive healthcare diagnostics.

## 5.2 Practical implications & future research

The findings from this study have a crucial significance within the domain of digital cognitive assessment, especially regarding how to optimize the neuropsychological tests. The Genetic Algorithm (GA) which can incorporate complicated layouts and maintain high-quality solutions has been among significant contributors towards the development of cognitive testing platforms. This highlights the GA's potential for improving TMT's reliability and validity in a digital format since paper-based traditional methods are being substituted by advanced technology-driven approaches in cognitive assessments. Our study showed that GA performs well hence it could be instrumental in designing accurate tests that adapt to various populations.

In practical scenarios, it is important to consider GA's demand for more computational resources like memory consumption when evaluating its usability as an algorithm. However, this expenditure on computing power is justified in clinical and research settings where quality and intricacy of TMT layouts are important factors. Accurate diagnosis and

cognitive assessment that can be achieved using these high-quality layouts will lead to better patient outcomes therefore justifying the resource investment.

Therefore, based on the current research one should choose algorithms tailored specifically for the needs of TMT layout problem wisely. For example, simplified versions of GA may not always be feasible if precise layouts are required because they may consume too much time or computational resources during execution but for other situations such as those involving complex designs it might be justifiable. Conversely, if faster results are needed say in a screening process where immediate feedback is valued more, then Simulated Annealing (SA) or Hill Climbing (HC) would be more appropriate because they require less computation time.

Going forward there is ample room for further development of these algorithms through future studies. One direction could involve developing adaptive GA schemes capable of automatically adjusting parameters depending on changing demands posed by optimization problem as an alternative to converging at local optima which characterizes most GAs used nowadays. Furthermore, hybrid algorithms combining features of GA, SA and HC need to be explored in order to develop novel solutions that combine the speed of SA and HC with the solution quality properties of GA. In this way, digital cognitive assessment tools can be technically efficient as well as broadly applicable to different clinical and research contexts. Hence, this study has implications for transformative changes in the methods used for cognitive testing towards algorithms that can support more nuanced and patient-centered assessments.

# 6. Conclusions

This paper was designed to identify the most efficient algorithm for optimizing layouts of Trail Making Test (TMT) and it proved beyond any reasonable doubt that Genetic Algorithms (GA) performed better than Simulated Annealing (SA) and Hill Climbing (HC) in terms of both computational efficiency and quality of layout. The fact that GA can generate very complex legitimate layouts while consuming minimal resource makes it ideal as a cognitive test, meeting the objectives of this study at its beginning.

In addition, the project has led to creating an API that is accessible by everyone through a cloud platform where community participation is promoted, and future technology-related research activities are pioneered. Moreover, this platform also acts as a guide for other studies intending to embed technical solutions into scientific research, not only in respect of facilitating development of optimized layouts but also during other similar evaluations.

From these findings, one can draw guidelines for further innovations such as hybrid algorithms that potentially combine the speed of SA and HC with GA's rigorous quality control. This may ultimately lead to better diagnostic tools for cognitive health, particularly Alzheimer's disease among others which could revolutionize healthcare technology.

Regarding this work, I have gained much knowledge about optimization algorithms and how they apply in real life scenarios thus enhancing my programming aptitude alongside problem solving skills. If I were to do something like this again, I would try to incorporate adaptive algorithms earlier to improve on time. This project has been highly informative

showing how dynamic tech research could be useful in practical world applications more than anything else.

# Afterword

In this academic journey, I have become well-rounded researcher as well as gaining enough engineering knowledge for my career. This research has enhanced my understanding while also giving me the fulfilment of doing something interesting in the area that I am devoted in. I am grateful to all who have walked with me along this enlightening road. I hope that what I learned through this study will be useful for people interested in the problematics of algorithmic computing as an asset to the scholarly discussion. Consequently, they reflect my life-long determination towards accumulating knowledge and sharing it with the community at large.

# References


I. El-Ghazali Talbi and Muntean, T. (2002) 'Hill-climbing, Simulated Annealing and Genetic algorithms: a Comparative Study and Application to the Mapping Problem', *[1993] Proceedings of the Twenty-sixth Hawaii International Conference on System Sciences*, vol. 2(pp. 565-573). Available at: https://doi.org/10.1109/hicss.1993.284069.

II. ExpressJS (2024) *Express 'Hello World' Example*, *expressjs.com*. Available at: http://expressjs.com/en/starter/hello-world.html (Accessed: 12 April 2024).

III. Haji, S.H. and Abdulazeez, A.M. (2021) 'COMPARISON OF OPTIMIZATION TECHNIQUES BASED ON GRADIENT DESCENT ALGORITHM: a REVIEW', *PalArch's Journal of Archaeology of Egypt / Egyptology*, 18(4), pp. 2715–2743. Available at: https://archives.palarch.nl/index.php/jae/article/view/6705.

IV. Holland, J.H. (2010) *Adaptation in Natural and Artificial Systems : an Introductory Analysis with Applications to biology, control, and Artificial Intelligence*. Cambridge, Mass. Mit Press.

V. Jaddi, N.S. and Abdullah, S. (2020) 'Global Search in single-solution-based Metaheuristics', *Data Technologies and Applications*, 54(3), pp. 275–296. Available at: https://doi.org/10.1108/dta-07-2019-0115.

VI. Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. (1983) 'Optimization by Simulated Annealing', *Science*, 220(4598), pp. 671–680. Available at: https://doi.org/10.1126/science.220.4598.671.

VII. Krentel, M.W. (1986) 'The Complexity of Optimization Problems', *ECommons (Cornell University)* [Preprint]. Available at: https://doi.org/10.1145/12130.12138.

VIII. Lambora, A., Gupta, K. and Chopra, K. (2019) *Genetic Algorithm-A Literature Review*.

IX. Linari, I. *et al.* (2022) 'Unveiling Trail Making Test: Visual and Manual Trajectories Indexing Multiple Executive Processes', *Scientific Reports*, 12(1). Available at: https://doi.org/10.1038/s41598-022-16431-9.

X. Oliveto, P.S. and Witt, C. (2015) 'Improved Time Complexity Analysis of the Simple Genetic Algorithm', *Theoretical Computer Science*, 605, pp. 21–41. Available at: https://doi.org/10.1016/j.tcs.2015.01.002.

XI. Reitan, R.M. (1958) 'Validity of the Trail Making Test as an Indicator of Organic Brain Damage', *Perceptual and Motor Skills*, 8(3), pp. 271–276. Available at: https://doi.org/10.2466/pms.1958.8.3.271.

XII. Tombaugh, T. (2004) 'Trail Making Test a and B: Normative Data Stratified by Age and Education', *Archives of Clinical Neuropsychology*, 19(2), pp. 203–214. Available at: https://doi.org/10.1016/s0887-6177(03)00039-8.

XIII. Wang, Z. *et al.* (2024) *Improved Genetic Algorithm Based on Greedy and Simulated Annealing Ideas for Vascular Robot Ordering Strategy.* Available at: https://arxiv.org/pdf/2403.19484.pdf (Accessed: 7 April 2024).

XIV. Wikipedia Contributors (2019a) *Genetic Algorithm*, *Wikipedia*. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Genetic_algorithm.

XV. Wikipedia Contributors (2019b) *Hill Climbing*, *Wikipedia*. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Hill_climbing.

XVI. Wikipedia Contributors (2019c) *Simulated Annealing*, *Wikipedia*. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Simulated_annealing.

XVII. Wikipedia Contributors (2019d) *Trail Making Test*, *Wikipedia*. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Trail_making_test.

XVIII. Zeng, Z. *et al.* (2017) 'Computerizing Trail Making Test for long-term Cognitive self-assessment', *International Journal of Crowd Science*, 1(1), pp. 83–99. Available at: https://doi.org/10.1108/ijcs-12-2016-0002.

# Appendix A

a.r.u. | Anglia Ruskin University

Certificate of Completion

This certifies that

Nuriddin Islamov

has successfully completed the CPD Course:

Introduction to Research and Professional Ethics

Delivered by Anglia Ruskin University

Date:

17/10/2023

652eda6c9dc5afc3390ead

# Appendix B

## Hill Climbing

```
function hc
    layout_size = 1024;
    num_nodes = 5;
    layout = rand(num_nodes, 2) * layout_size;
    max_iterations = 1000;
    iteration = 0;
    min_intersections = Inf;
    best_layout = layout;

    while iteration < max_iterations
        new_layout = best_layout;
        idx = randperm(num_nodes, 2);
        temp = new_layout(idx(1), :);
        new_layout(idx(1), :) = new_layout(idx(2), :);
        new_layout(idx(2), :) = temp;

        if is_layout_valid(new_layout)
            current_intersections = count_intersections(new_layout);
            if current_intersections < min_intersections
                min_intersections = current_intersections;
                best_layout = new_layout;
            end
        end

        iteration = iteration + 1;
    end

    plot_layout(best_layout);

end

function isValid = is_layout_valid(layout)
    numNodes = size(layout, 1);
    isValid = true;

    for i = 1:numNodes-1
        for j = i+1:numNodes
            if i ~= j && segments_intersect(layout(i, :), layout(i+1, :), layout(j, :),
layout(j+1, :))
                isValid = false;
                return;
            end
        end
    end
end
```

```
function result = segments_intersect(p1, p2, q1, q2)
    result = false;

    o1 = orientation(p1, p2, q1);
    o2 = orientation(p1, p2, q2);
    o3 = orientation(q1, q2, p1);
    o4 = orientation(q1, q2, p2);

    if o1 ~= o2 && o3 ~= o4
        result = true;
    elseif o1 == 0 && on_segment(p1, q1, p2)
        result = true;
    elseif o2 == 0 && on_segment(p1, q2, p2)
        result = true;
    elseif o3 == 0 && on_segment(q1, p1, q2)
        result = true;
    elseif o4 == 0 && on_segment(q1, p2, q2)
        result = true;
    end
end

function o = orientation(p, q, r)
    val = (q(2) - p(2)) * (r(1) - q(1)) - (q(1) - p(1)) * (r(2) - q(2));
    if val == 0
        o = 0; % collinear
    elseif val > 0
        o = 1; % clockwise
    else
        o = 2; % counter-clockwise
    end
end

function isOn = on_segment(p, q, r)
    isOn = (min(p(1), r(1)) <= q(1) && q(1) <= max(p(1), r(1))) && ...
           (min(p(2), r(2)) <= q(2) && q(2) <= max(p(2), r(2)));
end

function plot_layout(layout)
    x = layout(:, 1);
    y = layout(:, 2);

    figure;
    plot(x, y, 'o', 'MarkerSize', 10, 'MarkerFaceColor', 'b');
    hold on;

    for i = 1:length(x)
        text(x(i), y(i), sprintf('%d', i), ...
            'HorizontalAlignment', 'center', ...
            'VerticalAlignment', 'middle', ...
            'Color', 'White');
    end
```

```
    axis([0 1024 0 1024]);
    axis square;

    xlabel('X-coordinate');
    ylabel('Y-coordinate');
    title('Layout of Nodes with Numbering');

    grid on;
    hold off;
end
```

## Simulated Annealing

```
function sa
    numNodes = 10;
    gridSize = 1024;
    T_initial = 1000;   % Initial temperature
    T_final = 1;        % Final temperature
    alpha = 0.95;       % Cooling rate
    maxIterations = 1000; % can be adjusted

    layout = [rand(numNodes, 1) * gridSize, rand(numNodes, 1) * gridSize];

    T = T_initial;
    while T > T_final
        for iter = 1:maxIterations
            newLayout = layout;
            nodeIndex = randi(numNodes);
            perturbation = (rand(2, 1) - 0.5) * gridSize * 0.05;
            newLayout(nodeIndex, :) = newLayout(nodeIndex, :) + perturbation';

            newLayout(nodeIndex, :) = min(max(newLayout(nodeIndex, :), 0), gridSize);

            if is_layout_valid(newLayout, numNodes)
                layout = newLayout;
            end
        end
        T = T * alpha; % cooldown
    end

    % plotting
    figure;
    hold on;
    axis([0 gridSize 0 gridSize]);
    scatter(layout(:,1), layout(:,2), 'filled', 'SizeData', 100);
    for i = 1:numNodes
        text(layout(i,1), layout(i,2), num2str(i), ...
```

```
            'HorizontalAlignment', 'center', ...
            'VerticalAlignment', 'middle', ...
            'Color', 'white');
    end
    hold off;
end

function valid = is_layout_valid(layout, numNodes)
    valid = true;
    for i = 1:numNodes-1
        for j = i+1:numNodes
            for k = 1:numNodes-1
                for l = k+1:numNodes
                    if (i ~= k && i ~= l && j ~= k && j ~= l)
                        if   segments_intersect(layout(i,:),   layout(j,:),   layout(k,:),
layout(l,:))
                            valid = false;
                            return;
                        end
                    end
                end
            end
        end
    end
end

function intersect = segments_intersect(p1, p2, q1, q2)
    if is_collinear(p1, q1, p2) && on_segment(p1, q1, p2) ...
            || is_collinear(p1, q2, p2) && on_segment(p1, q2, p2) ...
            || is_collinear(q1, p1, q2) && on_segment(q1, p1, q2) ...
            || is_collinear(q1, p2, q2) && on_segment(q1, p2, q2)
        intersect = true;
    else
        o1 = orientation(p1, p2, q1);
        o2 = orientation(p1, p2, q2);
        o3 = orientation(q1, q2, p1);
        o4 = orientation(q1, q2, p2);
        intersect = (o1 ~= o2 && o3 ~= o4);
    end
end

function col = is_collinear(p, q, r)
    col = (q(2) - p(2)) * (r(1) - q(1)) == (q(1) - p(1)) * (r(2) - q(2));
end

function onSeg = on_segment(p, q, r)
    onSeg = q(1) <= max(p(1), r(1)) && q(1) >= min(p(1), r(1)) ...
        && q(2) <= max(p(2), r(2)) && q(2) >= min(p(2), r(2));
end

function o = orientation(p, q, r)
```

```
    val = (q(2) - p(2)) * (r(1) - q(1)) - (q(1) - p(1)) * (r(2) - q(2));
    if val == 0
        o = 0;
    elseif val > 0
        o = 1;
    else
        o = 2;
    end
end
```

## Genetic Algorithm

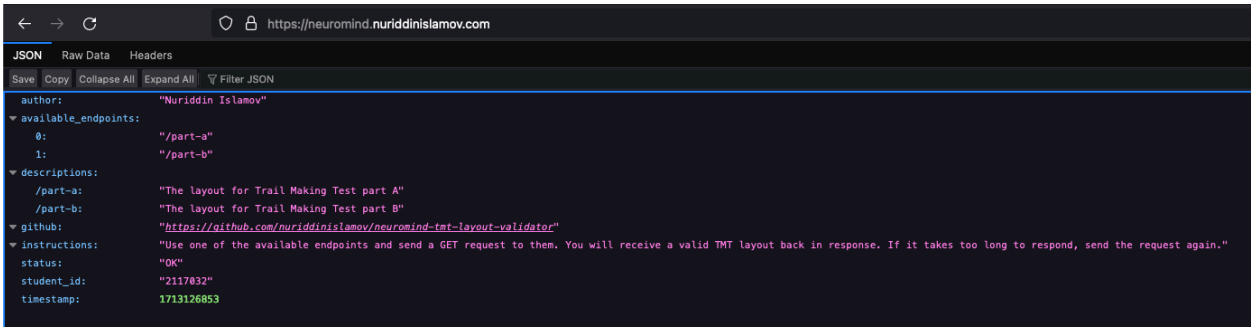
```
function ga
    % Timing
    tic;

    % Define constants
    gridSize = 1024;
    nodeCount = 25;
    populationSize = 100;
    maxGenerations = 1000;
    mutationRate = 0.1;

    population = cell(populationSize, 1);
    for i = 1:populationSize
        population{i} = generateRandomLayout(gridSize, nodeCount);
    end

    hFig = figure;
    hAx = axes('Parent', hFig);

    foundSolution = false;

    for generation = 1:maxGenerations
        % Calculate fitness for each individual
        for i = 1:populationSize
            if is_layout_valid(population{i})
                disp(['Solution found in generation ', num2str(generation)]);
                disp('Layout Coordinates:');
                disp(population{i});
                plotLayout(population{i}, hAx); % Plot final valid layout
                foundSolution = true;
            end
        end

        if foundSolution
            break;
        end
```

```
        selectedIndices = selection(population, populationSize);

        % Here happens
        % the crossover and mutation process,
        % that is explained more in depth in
        % the method section of the report
        newPopulation = cell(populationSize, 1);
        for i = 1:2:populationSize-1
            parent1 = population{selectedIndices(i)};
            parent2 = population{selectedIndices(i+1)};
            [child1, child2] = crossover(parent1, parent2);
            newPopulation{i} = mutate(child1, gridSize, mutationRate);
            newPopulation{i+1} = mutate(child2, gridSize, mutationRate);
        end
        population = newPopulation;


    end

    if ~foundSolution
        disp('No valid layout found within the maximum number of generations');
    end

    elapsedTime = toc;

    fprintf('Execution Time: %.2f seconds\n', elapsedTime);


end

function layout = generateRandomLayout(gridSize, nodeCount)
    layout = rand(nodeCount, 2) * gridSize;
end

function plotLayout(layout, hAx)
    cla(hAx);
    plot(hAx, layout(:, 1), layout(:, 2), 'o-', 'MarkerFaceColor', 'g');
    xlim(hAx, [0, 1024]);
    ylim(hAx, [0, 1024]);
    title('Final Valid Layout');
    drawnow;
end


% This function is used across all
% implementations consistently
% to ensure the validity and integrity
% during testing

% Inspired from is_valid_layout function that is defined in the
% algos/isValidLayout.py file of this project
```

```
function is_valid = is_layout_valid(layout)
    num_nodes = size(layout, 1);
    for i = 1:num_nodes-1
        for j = i+2:num_nodes-1
            if segments_intersect(layout(i,:), layout(i+1,:), layout(j,:), layout(j+1,:))
                is_valid = false;
                return;
            end
        end
    end
    is_valid = true;
end

function result = segments_intersect(p1, p2, q1, q2)
    function o = orientation(p, q, r)
        val = (q(2) - p(2)) * (r(1) - q(1)) - (q(1) - p(1)) * (r(2) - q(2));
        if val == 0, o = 0; elseif val > 0, o = 1; else, o = 2; end
    end
    function on_seg = on_segment(p, q, r)
        on_seg = (q(1) <= max(p(1), r(1)) && q(1) >= min(p(1), r(1)) && q(2) <= max(p(2),
r(2)) && q(2) >= min(p(2), r(2)));
    end
    o1 = orientation(p1, p2, q1);
    o2 = orientation(p1, p2, q2);
    o3 = orientation(q1, q2, p1);
    o4 = orientation(q1, q2, p2);
    result = (o1 ~= o2 && o3 ~= o4) || (o1 == 0 && on_segment(p1, q1, p2)) || (o2 == 0 &&
on_segment(p1, q2, p2)) || (o3 == 0 && on_segment(q1, p1, q2)) || (o4 == 0 && on_segment(q1,
p2, q2));
end

function indices = selection(population, populationSize)
    fitness = cellfun(@(x) sum(x(:)), population);
    maxFitness = max(fitness);
    adjustedFitness = maxFitness - fitness;
    totalFitness = sum(adjustedFitness);
    probabilities = adjustedFitness / totalFitness;
    cumulativeProbabilities = cumsum(probabilities);
    indices = arrayfun(@(x) find(cumulativeProbabilities >= x, 1, 'first'),
rand(populationSize, 1));
end

function [child1, child2] = crossover(parent1, parent2)
    point = randi([1, size(parent1, 1) - 1]);
    child1 = [parent1(1:point, :); parent2(point+1:end, :)];
    child2 = [parent2(1:point, :); parent1(point+1:end, :)];
end

function mutated = mutate(individual, gridSize, mutationRate)
    mutated = individual;
    for i = 1:size(individual, 1)
```

```
        if rand() < mutationRate
            mutated(i, :) = rand(1, 2) * gridSize;
        end
    end
end
```

## Live production




## Source code

Full source code can be found on the author's GitHub account:

https://github.com/nuriddinislamov/neuromind-tmt-layout-validator

# Appendix C

**Exit Plan**

My university journey has been an extraordinary chapter in my life, enriched by lifelong friendships and academic exploration that led to new discoveries within myself and in my field. As I approach the end of my academic journey, I am setting my sights on securing an internship in software engineering, a field that fascinates my interest deeply. With an Honors degree nearly in hand, I am ready to contribute significantly to the developers' community and the world. In the short term, I plan to visit my family in Uzbekistan, as it has been a long time since I left home for my studies in Cambridge. Indeed, my little brother has grown three years older since I last saw him.

In future, what I want is to be able to design sophisticated software programs specifically for aerospace sector and autonomous vehicle system. Furthermore, I am thinking about continuing with education by obtaining a master's degree in engineering or even possibly pursuing a PhD that will prepare me better for such challenges.

My passion for the morning coffee during my first and second years at the university has led me to getting a part-time barista job at a local coffee shop. This has shown me the hidden parts of operating and keeping clean financial records in the UK company. The experience has generated an interest in eventually establishing my own LLC in the UK, a goal I aim to pursue after gaining more industry experience and completing my studies.

Currently, I am preparing for coding interviews for internship positions, using resources like Leetcode and the "Cracking the Coding Interview" book by McDowell. Despite facing some setbacks with interviews during my time at Anglia Ruskin University, I am more focused than ever on securing a position in the industry upon graduation.

# Appendix D

## Nuriddin Islamov

github nuriddinislamov.com linkedin hi@nuriddinislamov.com

### About

As a final-year Computer Science student, I am prepared to embark on my professional journey in software engineering. With 3 years of hands-on experience in high-level programming languages, I had a chance to work effectively both in teams and individually, while managing extreme levels of productivity. Enjoy managing various side-projects among the coursework load by the university. Currently finalizing my project on exploring effective methods of global and local search algorithms for digitizing trail-making test, which brings impact in neuroscience sector. Looking for internship and entry-level positions.

### Education

**Anglia Ruskin University, Cambridge, UK** May 2024
*B.Eng. Computer Science* *Current GPA: 4.0/4.0*
**University of Westminster, UK (Foundation Year)** June 2021
*GPA: 4.0/4.0 (Distinction)*

### Coursework

**Courses:** Computer Systems, Algorithm Analysis & Data Structures, Embedded Systems, Core Maths for Computing, Image Processing, Network Routing, Database Design & Implementation
**Awards:** Outstanding Buddy Award

### Skills

**Languages**: Python, JavaScript/TypeScript, C/C++, Java, HTML/CSS, LaTeX
**Tools**: Git/GitHub, Unix Shell, Webpack, VS Code, Postman, Notion

**Frameworks**: React, NextJS, Django, ASP.NET, JUnit, Wordpress, Tailwind, FastAPI
**Libraries**: pandas, NumPy, Matplotlib

### Projects

**Soravur** | *Python, OpenAI API, Chatbot, Git, Unix Shell, VS Code* Dec. 2023 – Jan. 2024
- Individual side-project aimed to monetize OpenAI's GPT-4 Turbo API
- Developed a social media chatbot, where users can interact with GPT-4 model
- Learned adding payment systems to existing applications, used Payme Merchant and ClickUz API

**YouInvestMe** | *C#, ASP.NET, MVC, Bootstrap, Git, Azure Cloud* Sept. – Dec. 2022
- Developed a full-stack web application for university coursework
- Experimented with ASP.NET MVC application to implement CRUD operations
- Collaborated with four other students to meet the deadline and completeness of the application. Lead the team.
- Solved CI/CD issues and set up a pipeline to continuously deploy to Azure Cloud (used free student Azure credits)

**Trader bot** | *Python, PostgreSQL, Django, DRF* Jul. – Dec. 2021
- Created a cron job for the bot and automated user notifications with Telegram's Bot API
- Developed an API with Django Rest Framework for concurrent data access via HTTP calls to the backend server
- Deployed on a fresh new Ubuntu server on Digital Ocean with nginx, gunicorn and certbot

### Experience

**MLH Hackathon** | *Hacker* Jan. 2022
First experience in 72-hour hackathon, where our team has developed an animal language interpretation using Deepgram's API and free credits

**EZ Developers** | *Team Lead* May 2021 – Sept. 2021
Most recent professional software development experience, when I led a team out of my parents' office and successfully developed and pushed to production five commercial projects. Managed finances and the codebase of all projects.

### Hobbies / Languages

- Chess, Leetcode, Fiction, Undermining the existence of humanity, Travel
- English – Full Professional, Russian – Native, Uzbek – Native, Spanish – Basic, German – Basic, Arabic – Elementary