Friedrich-Alexander University
Faculty of Natural Sciences

Modelling Seminar

Solving the Tantrix puzzle using Linear Programming

- Seminar Report -

Submitted by

ABHIRAMI BINDHU - 23279832

JUSTINS PULPAKUNNEL VARKEY - 23140028

Under the Guidance of

Prof. Dr. Michael Hartisch

Department of Data Science

- Chair of Analytics & Mixed-Integer Optimization -

Friedrich-Alexander University

Erlangen-Nuremberg

Presentation Date: 07 August 2024

Abstract

The goal of this project is to develop a computer program that can solve the problem known as Tantrix, which involves joining lines painted on hexagonal tiles to form a loop. In order to do this, we provide a Tantrix problem setting that involves placing tiles in the proper order and simultaneously creating a loop inside a specified hexagonal lattice board. The Tantrix rules are then expressed as its constraints, and we structure it as an integer program. To find a solution, we use a mathematical programming solver. Consequently, we develop a formulation for solving Tantrix of moderate sizes, and we succeed in doing so by adding more constraints and creating an artificial objective function, even in cases when the solutions are invalid due to simple restrictions.

Setting for the Modelling and Optimization

Modelling and Optimization in this report are done using Gurobi, a library for optimization problems in the Python environment. For creating hexagonal grids, we used hexalattice. For visualization with the help of the library matplotlib . In total, we got a realistic solution of the optimized model.

Contents

1 Introduction

The puzzle known as Tantrix was created in 1988 by New Zealander Mike McManaway. Of the commercial Tantrix products that have been offered thus far, ”Tantrix Discovery” is the name of a solitaire version. In this study, we solely discuss Tantrix Discovery, the solitaire version, which we will refer to as ”Tantrix” throughout. Ten different kinds of identical-sized hexagonal tiles are used in the game Tantrix. There are two surfaces on a tile, which we refer to as the top and the back. Three lines are painted in the colors red, blue, and yellow on the top surface, while one of the numerals from 1 to 10 is drawn in one of the three colors on the rear surface. Each of the ten distinct line patterns painted on the tops is unique.

Description of the tile:

Top Surface: Every tile has a different pattern of colorful lines on top of it.
Back Surface: Each tile has a number between 1 and 10 displayed on the back in one of the three available colors.

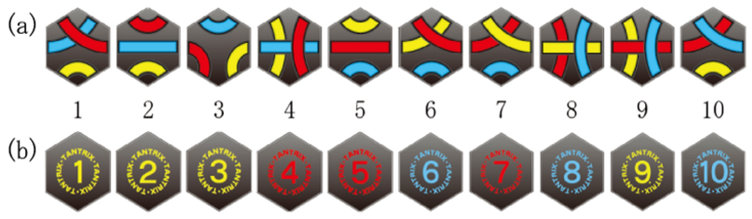

Figure 1: (a) Tops of 10 sorts of tiles (with their orientations 1), and (b) their corresponding backs.

Rules behind Tantrix

Tantrix is played by arranging tiles in the form of a hexagonal lattice. The goal is to make a loop in one designated color according to the following rules:

Determine the Challenge Number:

Choose the number of tiles for the challenge (the challenge number), which must be greater than 2.

Prepare Tiles:

Prepare as many tiles as the challenge number by starting with the tile numbered 1 and consecutively up to that number. If the number exceeds 10, restart from 1 again.

Select the Designated Color:

The designated color for the challenge is the color of the lowest digit on the back of the tile.

Form a Loop:

Connect all the lines of the designated color (drawn on the prepared tiles) so that they form a single loop.

Match Other Colors:

Connect the lines of the other colors to match their touching colors.

Additional Rules:

The arrangement of tiles must not have any holes (places without tiles surrounded by 6 tiles or more). Any tile line of the designated color must be part of the loop. Completing a loop according to the rules signifies clearing the challenge.

2 Problem Setting and Terminology for Formulations

Our goal is to utilize an integer programming (IP) framework, to solve the Tantrix puzzle. For this, we establish a board where tiles will be arranged to create workable Tantrix solutions. A tile can be placed in any hexagon inside the infinite plane (Crosswise Lattice Plane). A limited number of spaces are placed in predetermined configurations (spiral, for example). Tiles must be arranged so that the loop is completed in a "round" shape and without any holes. Board size greater than the challenge number in order to allow for the forms of valid solutions. There are Board Categories, Type A and Type B symmetric forms (sizes 7, 19, 37, etc.) ready for Tantrix solution.

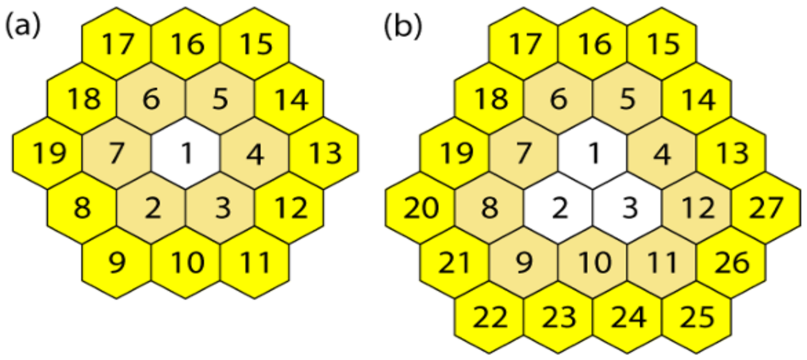

Figure 2: A numbering of places on the infinite hexagonal plane, and boards of sizes (a) 19 of type A and (b) 27 of type B with their place numbers.

In our project we used the board category Type A. So initially we created a plot of a 5x5 lattice of hexagons using the hexalattice library in Python. And doing so, we get a lattice, the one below:

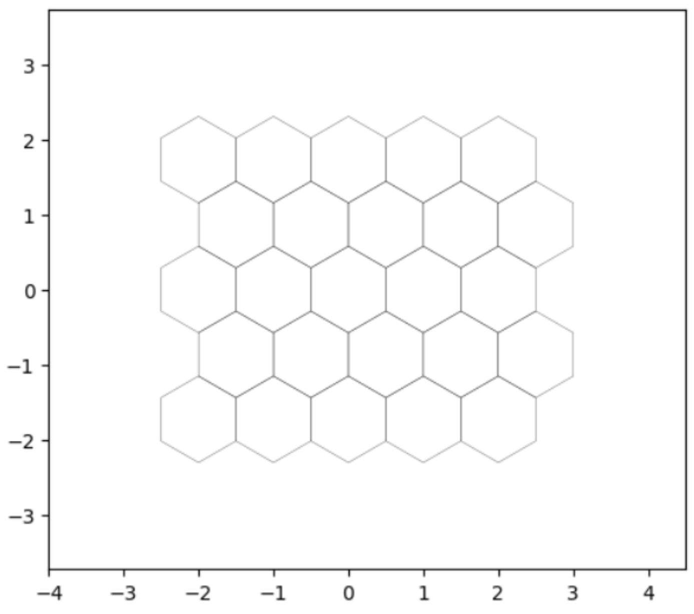

Figure 3: Hexalattice

Before placing the tiles into the lattice cells, the hexagonal lattice must be trimmed into a circular shape, and the cells need to be numbered according to the configuration shown below:

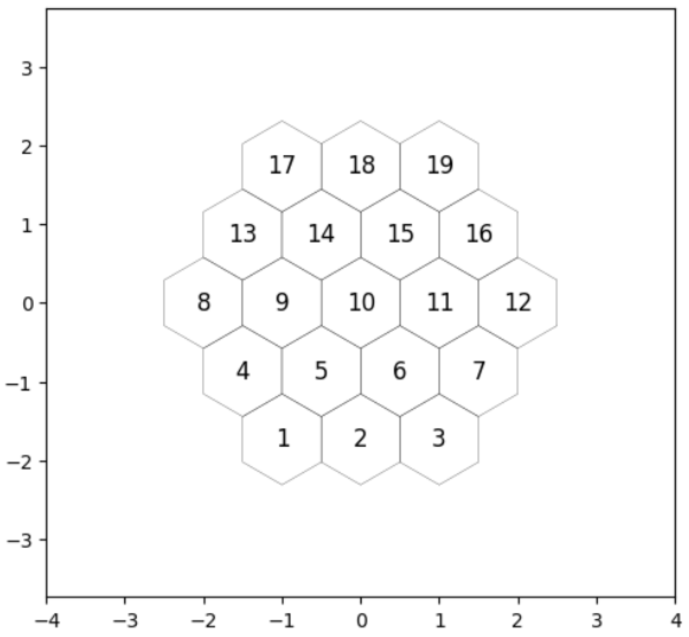

Figure 4: Circularly cropped - Hexalattice

The hexagonal lattice is numbered sequentially starting from the bottom left. The cell on the bottom left is marked as the first cell. Moving to the right, the next cell is marked as the second cell, continuing this pattern until the end of the row is reached. When there is no other cell to the right, numbering continues by moving to the top left of the current row. The next number is placed in the cell to the left of this new row, continuing the numbering to the right again. This pattern of moving right and then shifting to the top left continues until all the cells in the hexagonal lattice are numbered.

Formulation Specifics:

A tile with number i on its back, *tile i*. The angle of a line is its central angle when considered as part of a circle. Straight line: 0°, Other angles: 60°, 120°. Places on an input board are numbered from 1 to m.

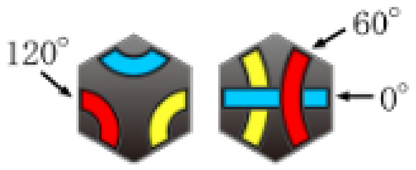

Figure 5: Angle of lines

Edges of each place are numbered counterclockwise from 1 to 6.

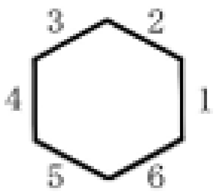

Figure 6: Edge numbering

Each tile has orientations numbered from 1 to 6.

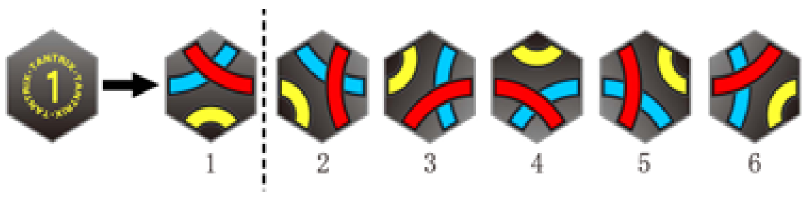

Figure 7: Tile orientation

Function $a(j,l)$ determines adjacency, that is it returns the number of the place adjacent to place j at edge l. Returns 0 if there is no adjacent place. Tiles can be identified by their placement and orientation. For example, adjacent tiles can be referred to by their colors or edges. A place is considered empty if no tile is placed on it.

3 Integer Programming Model

This section describes an Integer Programming (IP) formulation for solving the problem of placing tiles on a board, for the puzzle, Tantrix. The aim is to place a given number of tiles on a board in a manner that adheres to specific constraints.

3.1 Variables

Tile Placement Variable x_{ijk}: Binary variable that indicates whether tile i (where $i \in \{1, \ldots, 10\}$) is placed on place j (where $j \in \{1, \ldots, m\}$) with orientation k (where $k \in \{1, \ldots, 6\}$). $x_{ijk} = 1$ if tile i is placed on place j with orientation k. $x_{ijk} = 0$ otherwise.

Edge Color Variable y_{jl}: Variable that expresses the color of the edge l (where $l \in \{1, \ldots, 6\}$) of place j (where $j \in \{1, \ldots, m\}$). y_{jl} can take on the following values:

0: No color (for an empty place or edge).

1: The color is not the designated color nor color 2.

2: The color is not the designated color nor color 1.

3: The color is the designated color.

Helper Functions and Variables

Function $c(i,k,l)$: Returns the color of the line corresponding to edge l of a given tile i placed in orientation k.

Aggregate Variable: $u_{jj'} = 1$ if a tile is placed on either of the places j and j'. $u_{jj'} = 0$ otherwise.

3.2 Constraints

This section explains the constraints introduced in the integer programming formulation to solve the Tantrix puzzle, ensuring the solution's validity according to the game's rules.

Constraint 1 (C1): At most one tile is placed on each place.

This constraint ensures that no place on the board has more than one tile. It's mathematically represented by:

$$\sum_{i=1}^{10} \sum_{k=1}^{6} x_{ijk} \leq 1, \forall j \in \{1, 2, \ldots, m\}$$

where x_{ijk} is a binary variable indicating if tile i is placed on place j with orientation k. So, in general, this constraint ensures that no place j on the board has more than one tile, preventing overlaps and ensuring each position is uniquely occupied.

Constraint 2 (C2): The number of places each tile is placed equals the challenge number.

This constraint ensures that exactly the number of tiles corresponding to the challenge number n are used. It's formulated as:

$$\sum_{i=1}^{10} \sum_{j=1}^{m} \sum_{k=1}^{6} x_{ijk} = n$$

i.e., this constraint ensures that the number of tiles used matches the challenge number, ensuring the puzzle is set up correctly.

Constraint 3 (C3): Each tile is used the number of times defined by the challenge number.

This constraint ensures that each tile is used a number of times proportional to the challenge number:

$$\sum_{j=1}^{m} \sum_{k=1}^{6} x_{ijk} = \left\lfloor \frac{n+1-i}{10} \right\rfloor, \forall i \in \{1, \ldots, 10\}$$

The ceiling term ensures that each tile type i is used a specific number of times proportional to the challenge number n.

This constraint distributes the tiles according to the challenge number, ensuring a balanced and fair setup.

Constraint 4 (C4): The color of a line of an edge that is adjacent to no other tile is not the designated color.

This constraint ensures that edges not adjacent to another tile cannot have the designated color. This avoids incomplete loops or unconnected edges with incorrect colors.

Constraint 5 (C5): The colors of lines whose corresponding edges are touching each other have to match.
This ensures that adjacent tiles have matching edge colors. Mathematically, for any two adjacent places j and $j^{'}$, and for corresponding edges l and $l^{'}$:

$$y_{j\ell} = y_{j'\ell'}$$

This gives adjacent tiles matching edge colors, maintaining the puzzle's integrity.
Combined Constraints for Adjacency (C4+C5):
To handle the detailed adjacency constraints, we combine C4 and C5 into a unified constraint. This helps ensure that the colors match across adjacent edges and prevents the forming of invalid loops or disconnected paths:

$$-2u_{jj'} \leq y_{j\ell} - y_{j'\ell'} \leq 2u_{jj'}$$

Here, $u_{jj'}$ is a binary variable that is 1 if tiles are placed on both places j and j', and 0 otherwise. This combined constraint ensures that if tiles are placed, their colors must match.

3.3 Objective Function

An objective function is not absolutely necessary. In our case we are not looking for an optimal solution but for a valid solution, so an objective function enough to be virtual. Here, we take $x_{1,1,1} \rightarrow$ min., for a descriptive purpose.

4 Visualization and Analysis of the Solution

This chapter begins by detailing the methods for visualizing Tantrix solutions obtained through Gurobi. It will proceed to demonstrate various solutions, altering parameters such as the challenge number, board size, tiles, and game colors.

The solutions generated by the Gurobi model are represented by the binary values of the decision variable x_{ijk}. For example, if we consider a board with 19 places and a challenge number of 10, there are 1,140 possible x_{ijk} values. To display the results, these 10 x_{ijk} values must be identified, as they specify which tile is placed on which spot and in which rotation. The .png images of the tiles, stored in the Tiles list, can then be accurately rotated and placed on the board. Matplotlib is used to create an "AnnotationBbox" for each tile, positioning it correctly on the board. The final step involves plotting the board along with the tiles. The following sections will present the solutions found in various configurations.

Visualizing these solutions not only helps in understanding the optimal tile placements but also aids in identifying any potential improvements in the algorithm. The variations in challenge numbers and board sizes provide insight into the scalability of the solution approach. Additionally, different game colors allow for the examination of color-specific strategies and their effectiveness. By exploring these multiple configurations, a comprehensive analysis of the solution space is achieved.

By setting up the problem with an initial configuration of 3 tiles, which are numbered according to the scheme outlined in the introduction, and by choosing yellow as the designated color for this scenario, we obtain the following result:

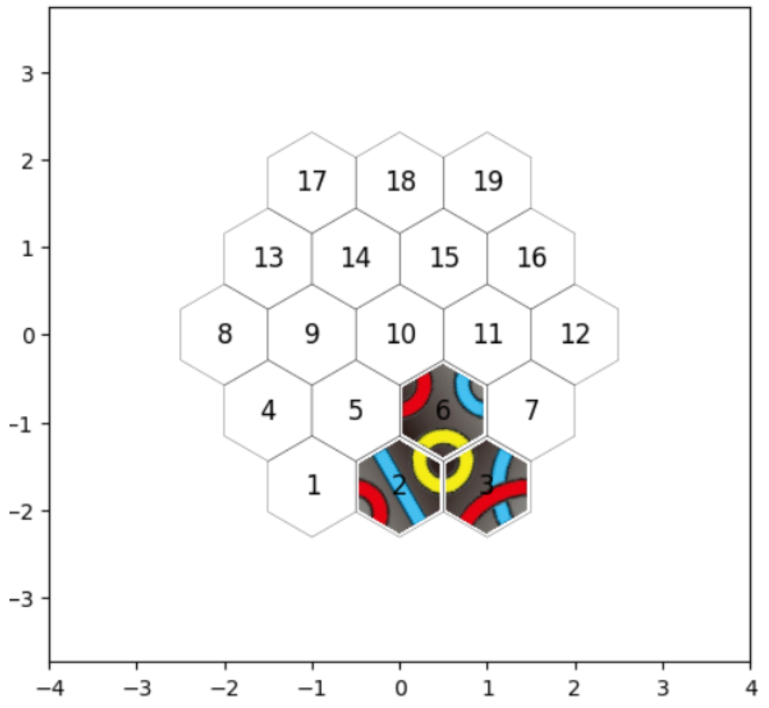

Figure 8: Designated color: Yellow, n=3

We expand the complexity of the problem by increasing the number of tiles involved: For a challenge number set to 10, the resulting solution of the problem is as follows:

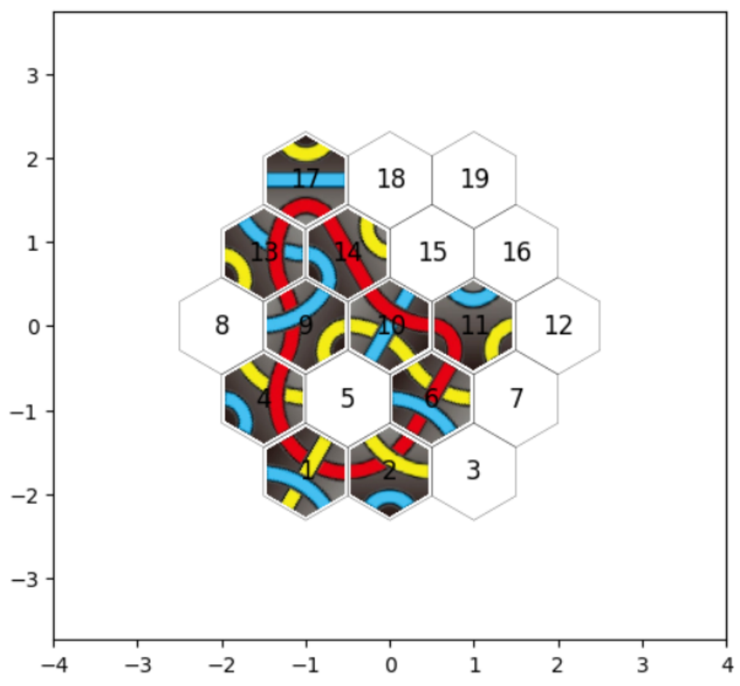

Figure 9: Designated color: Red, n=10

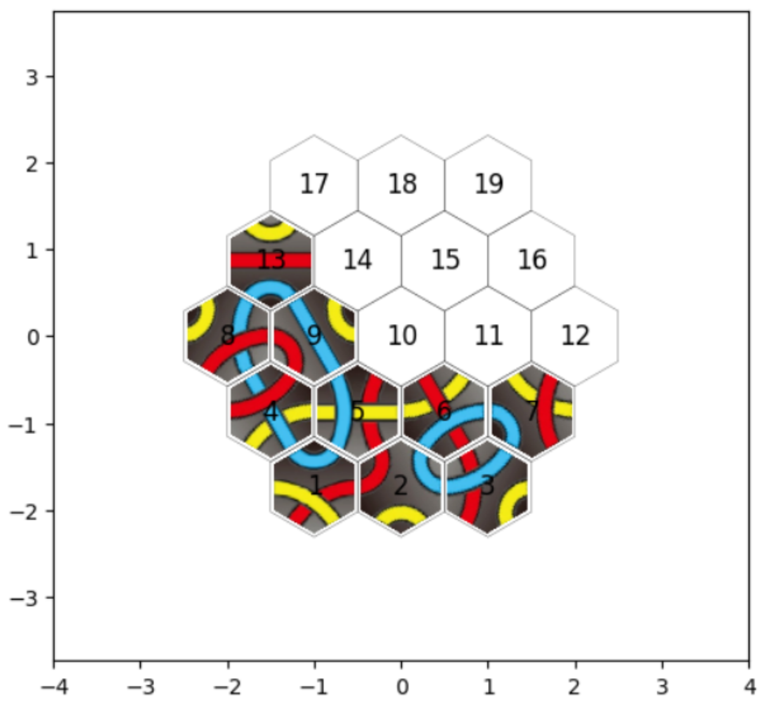

Figure 10: Designated color: Blue, n=10

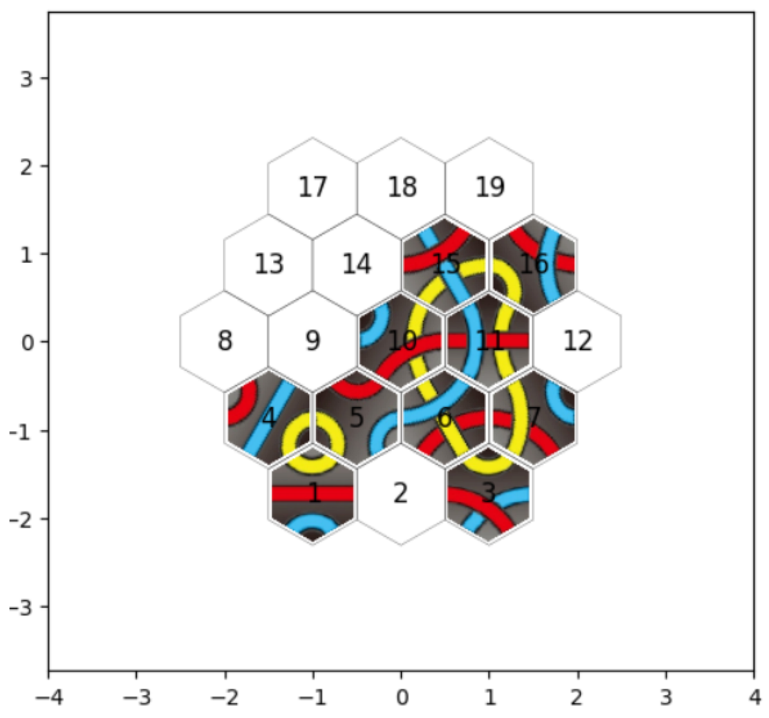

Figure 11: Designated color: Yellow, n=10

Additionally, we can apply the optimization problem to a scenario with a challenge number of 15 as well.

For instance, if we designate Red as the selected color for this challenge, the resulting solution will be as follows:

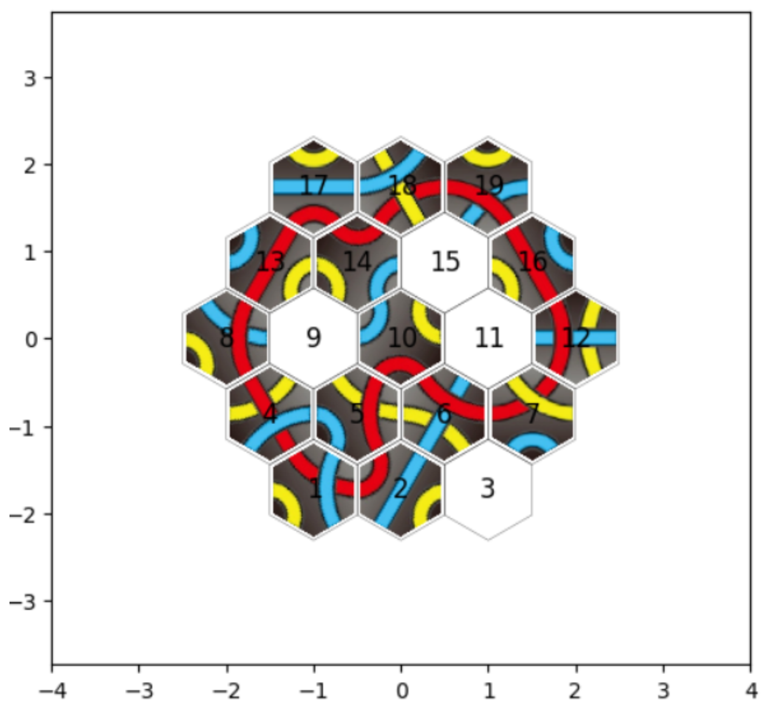

Figure 12: Designated color: Red, n=15

5 Improvement of Formulations

The IP formulations previously described can struggle with relatively small challenge numbers because they may produce solutions with holes and subloops. This section addresses these issues by proposing additional constraints to ensure the solution shapes are round, which helps avoid holes and eliminate subloops in the solutions.

The rules of Tantrix implicitly require the solution shapes to be round, which helps to prevent holes. Holes are problematic as they imply gaps in the board, which are not valid in Tantrix solutions. Here, new constraints are introduced to maintain this round and holeless property.

5.1 Restrict the Number of Adjacent Tiles

A hole is defined as a set of adjacent empty places surrounded by 6 or more tiles. To avoid holes, the following constraints are introduced:

Constraint 6a (C6a): At most 5 tiles are placed on adjacent places to an empty place.
Constraint 6b (C6b): At most 4 tiles are placed on adjacent places to an empty place.
Constraint 6c (C6c): At most 3 tiles are placed on adjacent places to an empty place.

These constraints are formulated as:

$$\text{C6a:} \quad \sum_{i=1}^{10}\sum_{k=1}^{6} x_{i,j',k} \leq \sum_{i=1}^{10}\sum_{k=1}^{6} x_{ijk} + 5, \quad \forall j = 1, \ldots, m$$

$$\text{C6b:} \quad \sum_{i=1}^{10}\sum_{k=1}^{6} x_{i,j',k} \leq \sum_{i=1}^{10}\sum_{k=1}^{6} x_{ijk} + 4, \quad \forall j = 1, \ldots, m$$

$$\text{C6c:} \quad \sum_{i=1}^{10}\sum_{k=1}^{6} x_{i,j',k} \leq \sum_{i=1}^{10}\sum_{k=1}^{6} x_{ijk} + 3, \quad \forall j = 1, \ldots, m$$

where x_{ijk} represents whether tile i is placed at place j with orientation k. The left-hand side of the formulas counts the number of tiles placed on adjacent places to a place j. If j is not occupied by a tile, the right-hand side adjusts the maximum allowed adjacent tiles.

Using Constraints Effectively

Since C6c implies C6b, and C6b implies C6a, we only need to use one of these constraints along with C1-C5. Computational results show that using C1-C5 with one of C6a, C6b, or C6c gives better solutions. Specifically:

C6a allows up to 5 adjacent tiles, resulting in more flexible but potentially holey solutions. C6b and C6c are stricter, preventing holes more effectively.

5.2 Adjusting the Objective Function

To further improve solution shapes, the objective function is adjusted to prefer round shapes. A new weight function is defined where each place is weighted based on its distance to the center of the board:

$$w_j = 1 - \frac{\|p_j - p_c\|}{R}$$

where p_j is the position of place j, p_c is the center of the board, R is the radius of the board.

The new objective function aims to maximize the sum of these weights for the placed tiles, encouraging a round shape:

$$\sum_{i=1}^{10} \sum_{j=1}^{m} \sum_{k=1}^{6} w_j x_{ijk} \to \max$$

Following the above two sections 5.1 and 5.2, we can avoid the holes and subloops occurring in the solution and thereby resulting in a valid Tantrix puzzle. [1]

6 Conclusion

In this report, we have explored the application of linear programming to solve the Tantrix puzzle, a challenging and engaging problem involving the arrangement of hexagonal tiles. By formulating the puzzle as an integer programming (IP) problem, we were able to leverage mathematical programming solvers, specifically Gurobi, to find solutions for various configurations of the Tantrix puzzle.

We began by introducing the Tantrix puzzle, its rules, and the specific constraints that must be met to solve it. Following this, we detailed the problem setting and the necessary terminology for formulating the puzzle within an IP framework. This included defining variables, constraints, and a virtual objective function to guide the solver towards valid solutions.

Through our computational experiments, we successfully solved Tantrix puzzles of different sizes, demonstrating the effectiveness of our approach. Visualizations of the solutions provided insights into how the solver arranged the tiles to form valid loops, adhering to the rules of Tantrix.

To enhance the robustness of our solutions, we can introduced additional constraints aimed at avoiding holes and subloops, ensuring that the solution shapes remained round and valid. Adjusting the objective function to favor round shapes further improved the quality of the solutions.

Overall, our approach successfully addressed the Tantrix puzzle for moderate sizes, providing a clear methodology for tackling similar combinatorial optimization problems using linear programming techniques. The implementation of this project, including code and visualizations, is available on GitHub, offering a valuable resource for further research and development in this area.

Future work could explore scaling the methodology to handle larger puzzle sizes, incorporating more sophisticated constraints, or applying similar techniques to other types of tiling and combinatorial puzzles. Our findings underscore the potential of mathematical programming in solving complex problems and pave the way for more advanced applications in optimization and game theory.

- Link to the GitHub:
  'Tantrix modeling'.

References

[1] Fumika Kino and Yushi Uno. “Solving Tantrix via Integer Programming”. In: *Fun with Algorithms*. Ed. by Evangelos Kranakis, Danny Krizanc, and Flaminia Luccio. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 245–257. ISBN: 978-3-642-30347-0.