Cheating on the Daily Paper: Solving Puzzles with Mathematical Optimization

Matthew Conroy, Julie Dieroff,
Tuilelaith Kehoe and Keyan Lu

Year 4 Project
School of Mathematics
University of Edinburgh
14th of March, 2025

Abstract

This report explores mathematical optimization techniques applied to the realm of deductive puzzles. These puzzles have the distinct challenge of spatial considerations in combination with logical rules. We predominantly focus on the branch of optimization that involves Integer Programming. By applying these methods to smaller scale puzzles we demonstrate how classical optimization techniques can be implemented in unconventional fields. We explore new developments in Bilevel Programming incorporated into a famous puzzle. We present several models for each puzzle and assess their comparative performance. We propose that taking a deliberate and calculated approach to a problem allows us to address particular logical constraints without the need for extra computation. From this we assert the importance of dealing with precise conditions carefully in Integer Programming formulations.

Keywords: Integer Programming, Logic Puzzles, Sudoku, Bilevel Programming, Set-partitioning Constraints, Minimum Cost Network Flow Formulation, Feasibility Problems.

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Matthew Conroy, Julie Dieroff, Tuilelaith Kehoe and Keyan Lu*)

To those who spend their mornings on sudoku.

Contents

Abstract ii

Contents vi

1 Introduction 1

1.1 Report Outline . . . 1

2 Preliminaries 2

2.1 Integer Programming . . . 2

2.1.1 Logical Constraints . . . 3

2.1.2 Linear Relaxations and Branch-and-Bound . . . 3

2.1.3 Better Formulations . . . 4

2.2 Graph Theory . . . 5

2.2.1 Connected Graphs . . . 5

2.2.2 Directed and Undirected Graphs . . . 6

2.2.3 Trees . . . 6

2.3 Minimum Cost Network Flow Problems . . . 7

2.4 Bilevel Programming . . . 7

3 Sudoku 8

3.1 Introduction . . . 8

3.2 The Model . . . 8

3.2.1 Set Partitioning Constraints . . . 9

3.3 Implementation . . . 10

3.4 Conclusion . . . 11

4 Battleship Solitaire Puzzle 12

4.1 Introduction . . . 12

4.2 The Models . . . 12

4.2.1 Grid Based Model . . . 13

4.2.2 Cell Based Model . . . 15

4.3 Implementation . . . 17

4.4 Comparison of the Models . . . 17

4.5 Conclusion . . . 18

5 Circuit Board Game **19**
5.1 Introduction 19
5.2 The Models 19
5.2.1 Spanning Tree Formulation 20
5.2.2 Minimum Cost Network Flow Formulation 21
5.2.3 Shortest Path Circuit Board 23
5.3 Implementation 25
5.4 Model Comparison 26
5.5 Conclusion 27

6 Hashi **28**
6.1 Introduction 28
6.2 The Hashi Model 29
6.2.1 Grid Formulation 29
6.2.2 Puzzle Initialization 29
6.2.3 The Model 30
6.2.4 Effect of Adding 'Human-Deduced' Constraints 34
6.3 Implementation 37
6.4 Model Comparison 37
6.5 Conclusion 38

7 Zebra Puzzle **39**
7.1 Introduction 39
7.2 The Models 40
7.2.1 IP Model 40
7.2.2 CP Model 42
7.3 Implementation 43
7.3.1 CP Heuristics 43
7.4 Model Comparison 44
7.4.1 Computational Efficiency Comparison 44
7.4.2 Strategy Comparison 45
7.5 Conclusion 45

8 Bilevel Programming for Sudoku **46**
8.1 Introduction 46
8.2 The BP Model 46
8.3 Implementation 48

9 Conclusion **49**

Bibliography **51**

A Appendix **54**
A.1 Repository and Implementation 54
A.2 Tables for Chapter 5 55
A.3 Tables for Chapter 6 57

Chapter 1

Introduction

It is a well known fact that what is challenging for humans is often not challenging for mathematicians. To prove this point, we have moved from solving daily newspaper puzzles by hand and have instead enlisted the aid of advanced mathematical theory.

Making better decisions is the motivation behind optimization, whether this relates to large scale supply chain management or logic based puzzles. In this report we will focus on the latter. This allows us to view optimization techniques in a different light and investigate unique instances. The puzzles featured in this report all rely heavily on deductive reasoning but have their own particular structure. The aim of this report is to develop models for deductive puzzles and investigate the characteristics that make one model superior to another. We will achieve this using Integer Programming, a famous optimization technique that is particularly suitable for this topic due to the compatibility between fixed rules and constraint design. In addition, we will reproduce a recent development in Bilevel Programming by Tjusila et al.[25] who created a model that given a filled Sudoku grid, finds the starting grid with the minimum number of clues which defines that solution uniquely.

1.1 Report Outline

Chapter 2 introduces the relevant concepts of Integer Programming as well as other mathematical topics that feature in this report. The subsequent chapters are organised such that each one is dedicated to discussing a different puzzle. We begin our exploration of using optimization for puzzles with the famous example of Sudoku in Chapter 3. This will enable us to explore the key concepts of Integer Programming through a widely known model. Following this we will adapt these concepts for more intricate puzzles. We will compare different models to show the wide range of methods that can be used. Chapter 8 then briefly outlines the new developments of Bilevel Programming applied to Sudoku.

To ensure consistent implementation of our models, we have used the software Gurobi Optimizer (v. 11.0.3) [10] unless stated otherwise. Additionally, the grid-based puzzles and their solutions will be visualised using the Python package NetworkX [11].

Chapter 2

Preliminaries

2.1 Integer Programming

Integer Programming (IP) is a form of constrained optimization in which at least one variable must be an integer and all objective functions and constraints are linear functions. To understand the details of IP we must first discuss Linear Programming (LP). A LP problem is also a form of constrained optimization. It finds a solution for continuous variables that maximises or minimises an objective function whilst also satisfying constraints. A LP problem with a combination of integer and continuous variables is often a Mixed Integer Programming problem. For ease of notation, we refer to LP problems where at least one of the variables must be integer as an IP problem [3, p. 3]. IP problems are often of the form

$$\begin{aligned} \text{minimise } & c^t x + h^t y \\ \text{subject to } & Ax + Gy \leq b, \\ & x_i \geq 0 \quad \forall i \in \{1, ..., n\}, \\ & y_i \in \mathbb{Z} \quad \forall i \in \{1, ..., p\}, \end{aligned} \tag{2.1}$$

where c, h and b are vectors of dimensions n, p and m respectively, A is an $m \times n$ matrix and G is an $m \times p$ matrix. The vectors x and y are decision variables; they contain the variables we want to optimize [4, p. 2]. A common type of decision variable is what is known as a binary variable. These are decision variables whose values are restricted to be either zero or one [3, p. 5].

An IP problem of the form shown by (2.1) can also be written as

$$\max\{c^t x + h^t y : (x, y) \in S\}, \tag{2.2}$$

where $S = \{(x, y) \in \mathbb{Z}^n_+ \times \mathbb{R}^p_+ \mid Ax + Gy \leq b\}$ [4, p. 6]. This set S also describes what we call the feasible region of the IP problem. This is the set of points that satisfy the constraints of the problem. As such, we define a formulation to be a polyhedron $P \subseteq \mathbb{R}^{n+p}$ such that $S \subseteq P$ [3, pp. 80-81].

As mentioned above, an objective function is the linear function in a problem that is maximised or minimised. In problem (2.1) the objective function is $c^t x + h^t y$. In many applications of IP the objective is to minimise costs or maximise revenue, but it is often the case for puzzles that a solution must only satisfy the rules of the game. Thus, in this report we will cover many examples of problems in which one solution is not preferable to another. These are called feasibility problems and their objective function is

$$\text{minimise } \mathbf{0}^t x,$$

where maximise and minimise can be used interchangeably [4, p. 164].

2.1.1 Logical Constraints

Binary variables are instrumental for modelling logical expressions. For example, if we are are modelling two related statements A and B we can use binary decision variables x_A and x_B. Let $x_A = 1$ when A is true and $x_A = 0$ when it is false, similarly let $x_B = 1$ when B is true and $x_B = 0$ when it is false. Using these binary variables allows us to model logical expressions as linear constraints. Table 2.1 displays common logical relationships transformed into linear constraints [3, pp. 56-58].

$\mathbf{A \cap B}$	$\mathbf{A \cup B}$	$\mathbf{A \implies B}$	$\mathbf{A \iff B}$	$\mathbf{\sim A}$
$x_A + x_B = 2$	$x_A + x_B \geq 1$	$x_A \leq x_B$	$x_A - x_B = 0$	$x_A = 0$

Table 2.1: Logical expressions represented as linear constraints using binary variables.

2.1.2 Linear Relaxations and Branch-and-Bound

Unlike LP problems, IP problems are more complex to solve. As a result, a crucial concept for solving IP problems is that of linear relaxations (LR). They are models in which the integrality constraints from an IP problem have been removed. The LR of an IP problem, Equation (2.2), is

$$\max\{c^t x + h^t y : (x, y) \in P\}$$

where $P = \{(x, y) \in \mathbb{R}^n_+ \times \mathbb{R}^p_+ \mid Ax + Gy \leq b\}$ [4, p. 3].

A typical method to solve IP problems is Branch-and-Bound but this is performed by the solver. Regardless, it is still important to understand how the solutions are found. This method involves solving a series of LP problems using a strategy called divide and conquer. The given IP problem will be split into easier to solve subproblems. From the solutions to the subproblems we can determine the solution to the original IP problem. The Branch-and-Bound solving process can be organised into a tree to display how we have reached the solution. The subproblems in this method are the LR of relevant IP problems. Below we detail the method for a maximisation problem. Note that the minimisation process follows the same method but interchanges upper and lower bounds [3, pp. 272-273].

1. We begin at the root node, this is the LR of the original IP problem. If the optimal solution of the LR is integer then the process is complete.

Otherwise, this objective value is an upper bound for the IP problem's optimal objective value. The root node is then split into two branches.

2. The branches following the root node are created by applying valid cuts that remove the non-integer solution for the LR. This is done so that the nodes of these branches are mutually exclusive and their union contains the previous LR solution. We can now solve the LR of the IP problems at each node.

3. The solutions at each node inform us whether this branch can be pruned and a better lower bound for the objective value of the original IP problem is found. A node can be pruned if the LR is infeasible, has an integer optimal solution or if the upper bound for this subproblem is less than or equal to the lower bound of a problem above it.

4. Whenever an integer optimal solution is found to a LR it becomes a candidate solution to the original IP problem. The best candidate solution is called the incumbent solution.

5. This process continues until all branches have been explored. The solution to the IP problem, if one exists, is the final incumbent solution.

Optimality Gap

Since the Branch-and-Bound method finds upper and lower bounds for the IP problem, an important notion is that of an optimality gap. This is the gap between the upper bound $\bar{z}$ and lower bound $\underline{z}$:

$$\frac{|\bar{z} - \underline{z}|}{|\underline{z}|}.$$

As the Branch-and-Bound method explores the branches, $\bar{z}$ and $\underline{z}$ should become closer together, therefore closing the gap until they are equal at which point an optimal solution has been found [28, p. 25]. We will use this concept to evaluate how close a model has gotten to solving an IP problem when an optimal solution is not found in the allotted time frame.

2.1.3 Better Formulations

Throughout this report we will be examining and comparing different IP models. In order to understand which models are superior we must define what it means for a formulation to be better than another. Although a model with less constraints and fewer integer variables can be faster to solve, we will be focusing on the feasible region of the LR for the IP models. In general, we want to design a formulation such that the difference between the solution to the IP problem and the solution to its LR is small. Therefore, given two formulations P_1 and P_2 for X, P_1 is a better formulation than P_2 if $P_1 \subset P_2$. Unfortunately, for large IP problems checking if one formulation is a subset of another is computationally expensive. Thus, in practice we will use the fact that if P_1 is a better formulation than P_2, for a minimisation problem, the optimal value of P_1 will be greater than the optimal value of P_2 [3, pp. 80-82]. Figure 2.1 displays two formulations for an IP problem whose feasible points are represented by the black dots, where P_1 is better than P_2. It is clear that the minimum value of P_1 must be greater than that of P_2.

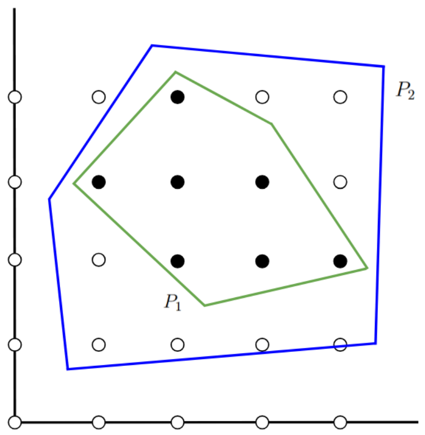


Figure 2.1: Formulations P_1 in green and P_2 in blue where P_1 is a better formulation than P_2 since $P_1 \subset P_2$. The black dots represent the feasible points in the IP problem.

2.2 Graph Theory

Many optimization problems in the real world have graphical representations. For example, optimizing the route snowploughs take in the winter and optimizing the placement of pipelines in power plants. Several of the puzzles we examine in this report utilise concepts from Graph Theory so we will be establishing some key terms here.

A set of points connected by lines is called a graph, where the points are referred to as vertices and the lines are called edges [27, p. 2].

The term adjacent is extremely important and has different meanings depending on the context. Two vertices are adjacent if they are connected by an edge, then those vertices are incident to that edge. Whereas, two distinct edges are adjacent if they are both incident to the same vertex [27, p. 12]. If all pairs of vertices in a graph are adjacent then this graph is called complete [27, p. 18]. We note that a loop is an edge which connects a vertex to itself, but the puzzles in this report do not utilise them [27, p. 2]. We also define the degree of a vertex as the number of edges it is incident to [27, p. 1]. Figure 2.2 depicts a complete graph with five vertices and ten edges, denoted K_5. Every vertex of K_5 has degree four.

2.2.1 Connected Graphs

Another important concept is that of a connected graph, in order to understand them fully we first establish some basic terms. We refer to a walk as a means of reaching one vertex from another using the edges adjacent to them. Then, a cycle is a walk in which no vertices appear more than once with the exception of the start/end vertex, as highlighted in red in Figure 2.2. Furthermore, a path is a walk where a vertex never arises more than once. Then we can define a connected graph as one where any two vertices are connected by a path [27, pp. 3-4]. The complete graph K_5 displayed in Figure 2.2 is an example of a connected graph.

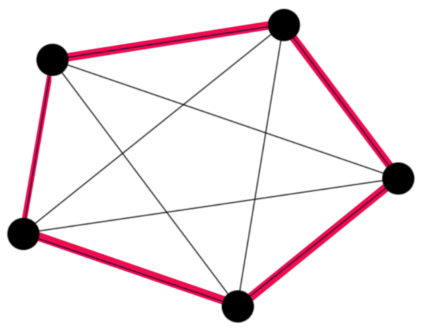


Figure 2.2: The complete graph K_5. A cycle of length five is highlighted in red.

2.2.2 Directed and Undirected Graphs

A directed graph consists of a finite number of vertices which are joined by arcs. Arcs are edges with direction [27, p. 22]. Directed graphs are frequently used to represent networks like pipes or streets and are also helpful for illustrating circuits. We will refer to graphs without direction as undirected graphs.

2.2.3 Trees

We define a tree as a connected graph that does not contain any cycles [27, p. 61]. An important notion that will be useful in this report is that of a spanning tree. For a given connected graph G, its spanning tree is a tree that connects all of its vertices. Its construction follows from choosing a cycle from G, if any, arbitrarily removing one of its edges and repeating this until there are no more cycles left [27, p. 62]. The following theorem justifies constraints used to model the puzzle in Chapter 5.

Theorem 1. *[27, p. 62] If graph G, with n vertices, contains no cycles and has $n-1$ edges, then G is a tree.*

Proof: We will begin by proving that G is connected by contradiction. Assume G is a disconnected graph with $n-1$ edges and no cycles. Then, each component of G is a connected graph. The components do not contain any cycles since G as a whole does not. We wish to show that the number of vertices in each component is one more than the number of edges. Note that the components themselves are trees; they are connected graphs with no cycles. A tree with k vertices contains $k-1$ edges by induction:

Base Case $k = 1$: A tree with one vertex has no edges so the statement holds.

Induction Hypothesis: Assume a tree with k vertices contains $k-1$ edges.

Induction Step: We know that a tree with k vertices contains $k-1$ edges so we add one vertex to this tree, now the graph has $k+1$ vertices. We can attach the new vertex to the graph using one edge without creating any cycles. This is a tree since it is a connected graph without cycles. This tree has $k+1-1=k$ edges.

Conclusion: From the combination of the base case and the induction step we can conclude that the induction hypothesis holds.

Since the number of vertices in each component of G is one more than the number of edges, this means that the total number of vertices in G must be more than the number of edges by at least two. This contradicts the assumption that G contains $n-1$ edges. Therefore, the graph G must be connected.

We have established that when G contains no cycles and has $n-1$ edges it must be connected. We have defined a tree as a connected graph with no cycles, so we can conclude that graph G is indeed a tree.

2.3 Minimum Cost Network Flow Problems

A notable group of IP problems are what are classed as Minimum Cost Network Flow (MCNF) problems. This type of problem includes transportation problems, maximum flow and assignment problems. In general, these networks contains three types of vertices, the source vertex, the demand vertex and intermediate vertices. Each vertex in a network has a supply, if the vertex is a source the supply will be positive, a demand vertex has negative supply and an intermediate vertex has no supply. There is a cost c_{ij} associated with transporting supply between every vertex i and j. The problem is to determine how much flow can be transported between each vertex while minimising costs. An important rule that must be upheld at all times is that flow may not be created or destroyed across the network [3, p. 249].

2.4 Bilevel Programming

Bilevel Programming (BP) extends the principles of IP. It refers to an optimization problem that contains two levels of decision making, each with its own decision variables, constraints and objective functions. These two layers of optimization form a hierarchical structure, involving the two decision-makers commonly referred to as the 'leader', the upper-level problem (ULP), and the 'follower', the lower-level problem (LLP). The core principle here is that the follower and leader problems interact in terms of information such as the decision variables and optimum solutions being passed between levels [6, pp. 1-2]. By the optimum solutions of the LLP, we refer to the set of all possible solutions that optimize the LLP [7, pp. 4-5].

Chapter 3

Sudoku

3.1 Introduction

Sudoku, originally named ‘Number Place’, was invented by Howard Garns in 1979 who adapted the game created by Leonhard Euler in 1783 called ‘Latin square’. It was not until 1984, when the game was published in Japan under the name *Suji wa dokushin ni kagiru* which means ‘the numbers must be single’, that the name was changed to Su-doku [2, pp. 6-7]. Sudoku consists of a 9×9 grid where each column, row and inner 3×3 box must contain one occurrence of the numbers 1 to 9. Figure 3.1 depicts an example of Sudoku which begins with certain numbers that must be included in the solution.

3			4			9		
				7			6	5
	9	7	6					
	8		2	3		5		
	7						9	
		2		9	7		8	
					5	7	3	
7	4			8			5	
		5			1			8

Figure 3.1: Sudoku puzzle to be solved.

In this chapter we will examine an IP model for the game to begin our understanding of how optimization methods can be applied to puzzles. In particular, this applicability lies in the specific constraints we use and the theory behind modelling games on grids. Furthermore, a crucial takeaway is how we are able to transform logical rules into linear equations using binary variables.

3.2 The Model

The following IP formulation is based on the common IP model created to solve Sudoku puzzles [1, p. 3]. We have adapted the notation and some indices to improve the clarity of the model. Note that this a general model so we do not use explicit grid dimensions thus allowing for scalability.

For an $n \times n$ grid let the individual cells be denoted (i, j) where i is the row and j is the column where this cell appears. Let $m = \sqrt{n}$ represent the dimension of the inner boxes of the puzzle. Let $D = \{1, 2, ..., n\}$ and $E = \{1, 2, ..., m\}$. Let C denote the $n \times n$ matrix of clues for the puzzle. Its ij-th entry, c_{ij}, contains either a number that must be in that cell in the solution or a 0 representing a cell that must be filled. The decision variables are

$$x_{ijk} = \begin{cases} 1, & \text{if cell } (i,j) \text{ contains the integer } k, \text{ for } i,j,k \in D, \\ 0, & \text{otherwise,} \end{cases}$$

and the formulation is

$$\begin{aligned}
&\text{minimise } \mathbf{0}^t x && && (3.1\text{a})\\
&\text{subject to } \sum_{i=1}^{n} x_{ijk} = 1, && \forall j, k \in D, && (3.1\text{b})\\
&\sum_{j=1}^{n} x_{ijk} = 1, && \forall i, k \in D, && (3.1\text{c})\\
&\sum_{j=mq-m+1}^{mq} \; \sum_{i=mp-m+1}^{mp} x_{ijk} = 1, && \forall k \in D, \ \forall p, q \in E, && (3.1\text{d})\\
&\sum_{k=1}^{n} x_{ijk} = 1, && \forall i, j \in D, && (3.1\text{e})\\
&x_{ijc_{ij}} = 1, && \{i, j \in D \mid c_{ij} \neq 0\}, && (3.1\text{f})\\
&x_{ijk} \in \{0, 1\} \ \forall i, j, k \in D. && && (3.1\text{g})
\end{aligned}$$

Since one solution to Sudoku is not preferable to another as long as it satisfies the rules, this is a feasibility problem. Hence the objective function Equation (3.1a) minimises $\mathbf{0}$. Next, we use Equation (3.1b) to add the constraint that each number appears exactly once in each column. Similarly, Equation (3.1c) ensures that each number appears once in each row. Equation (3.1d) creates the constraint that the $m \times m$ inner boxes all contain one occurrence of each number. Then, Equation (3.1e) adds the requirement that every square in the grid contains precisely one number. Finally, Equation (3.1f) is the constraint that requires all feasible solutions to include the numbers initially provided in the puzzle in the correct places. Equations (3.1b)-(3.1e) are all examples of set partitioning constraints.

3.2.1 Set Partitioning Constraints

Set partitioning constraints are used to ensure that given a certain number of options exactly one is chosen [3, p. 111]. In this case the options are numbers from 1 to 9, but in other applications these sets may be: possible routes for delivery trucks to take, which lecture should occur at a given time, or even who should receive a kidney transplant from whom.

3.3 Implementation

Figure 3.2 displays the solution to the puzzle depicted in Figure 3.1 computed using the model.

3	6	8	4	5	2	9	1	7
2	1	4	8	7	9	3	6	5
5	9	7	6	1	3	8	2	4
1	8	9	2	3	4	5	7	6
4	7	3	5	6	8	1	9	2
6	5	2	1	9	7	4	8	3
8	2	6	9	4	5	7	3	1
7	4	1	3	8	6	2	5	9
9	3	5	7	2	1	6	4	8

Figure 3.2: The solution to the Sudoku puzzle depicted in Figure 3.1 computed using IP.

To examine the speed of the model for varying levels of difficultly, we tested it on ten standard 9×9 puzzles all found in the Telegraph's book of Sudoku puzzles [14]. As such our model was easily verified by comparison to the provided solutions. Here we measure difficulty by the number of clues that are given. This process also allowed us to confirm the validity of our model by inspection. Table 3.1 shows how long it took to solve each puzzle using the model stated above. Based on this table, the level of difficulty, green for 'Easy' puzzles, yellow for 'Medium' and red for 'Hard', does not affect the computation time. Every puzzle takes less than 0.05 seconds to be solved and displays no consistency across levels of difficulty. This is expected since these are very small IP problems that do not take long to solve. A difference in the number of clues provided on this scale is thus insignificant. Therefore we can conclude that what is considered a difficult problem for people in terms of deductive reasoning does not translate to a computational challenge for IP.

Number of Clues	Time (seconds)
36	0.02916
32	0.03869
32	0.04757
31	0.04716
29	0.01603
26	0.03181
23	0.01719
23	0.03190
22	0.03150
22	0.03841

Table 3.1: Number of Clues given in the Sudoku puzzle compared to the computation time. Green entries correspond to "Easy" puzzles, yellow to "Medium" and red to "Hard".

3.4 Conclusion

In this chapter we set out to create an IP formulation that can solve Sudoku puzzles. This revealed characteristics that certain puzzles feature which are compatible with famous IP modelling techniques. They are particularly suitable due to Sudoku puzzles being number based and the grid's similarity to a matrix. This allowed us to easily index our decision variables in an intuitive manner, using grid coordinates. Specifically, we were able to apply the well established set partitioning constraints to the requirement that each number can be used once in the designated sections.

Notably, we found that puzzles which are labelled as difficult to solve do not necessarily have longer computation times. This can be attributed to the small size of the problems. Many of the puzzles explored in this report are typically played on a small scale in order for humans to be able to complete them in a reasonable amount of time. Thus, many of the IP models are solved promptly as confirmed in the next chapter.

Chapter 4

Battleship Solitaire Puzzle

4.1 Introduction

The Battleship Solitaire Puzzle (BSP) featured in this chapter is a logic puzzle based on the Battleship board game, first appearing in Spanish magazine 'Humor & Juegos' in 1982 [19, p. 156].

The puzzle begins with a mostly empty grid and the aim is to locate all the ship positions from the given clues. Initially we are given a 10×10 grid with the number of ship parts contained in each row and column, how many ships of each length are on the grid, and some squares already filled in. The aim is to then place the ships according to these parameters such that ships are not overlapping or touching. Figure 4.1 depicts a completed puzzle. Overall, the puzzle is quite similar to Sudoku. Both puzzles start off with a partially filled grid and the player is required to determine what is contained in the remaining squares based off various clues and the rules of the game.

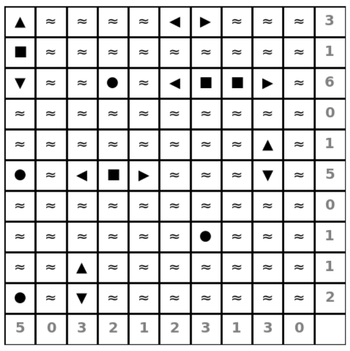


Figure 4.1: Completed BSP on a 10×10 grid.

4.2 The Models

In this chapter, we construct an IP model for the BSP and then compare it to an existing model for various instances. Since this puzzle is similar to Sudoku, we use this model as a base and then account for more complex rules. For example, it is easy to see how the constraints which ensure that the correct number of ship pieces are in each row or column will resemble the set partitioning constraints in the Sudoku model. One key difference with the BSP is that we must also consider the additional rule regarding the placement of the ships; there is a set number of ships of each length and conflicting ships must not be chosen.

Both models will represent a celltype using the variable $k \in \{0, ..., 6\}$. They are defined in Table 4.1.

k	Celltype	Description
0	$\approx$	Water
1	◀	Left end
2	▶	Right end
3	▲	Top end
4	▼	Bottom end
5	■	Hull
6	●	Buoy

Table 4.1: Variable $k \in \{0, ..., 6\}$ representing each celltype, and their description.

4.2.1 Grid Based Model

To mitigate the complexity of this problem we form our model using sets containing all the possible ship positions and find our solution by choosing a valid combination of ships. Thus, many of the requirements are inherently met. This method was chosen over a cell based approach to reduce the number of constraints and variables. Meuffels et al. also formulated a model that follows similar logic [19, p. 159].

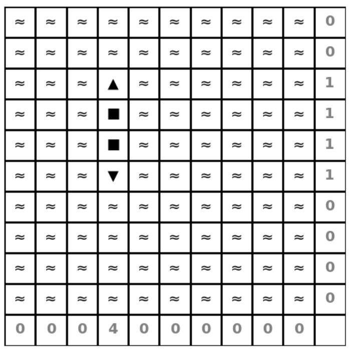


Figure 4.2: A 10×10 grid with a vertically placed ship of length 4 contained in A.

For the Grid Based Model (GBM) let L be the set of all ship lengths. Then, we utilise grids to describe the possible positions of the ships. As such, each grid contains one ship in a potential position. Note that this means there can be multiple grids containing the same ship in order to represent all of its possible locations. Therefore we have a set A containing these grids. Figure 4.2 displays a grid in A with a potential location of a ship with length 4 on a 10×10 grid. We will use A_{ija} to refer to a cell in row i, column j, for grid $a \in A$ containing variable k as in the table. A subset of A is P_l, the grids specifying the positions of ships with length $l \in L$.

For an $m \times n$ grid let $M = \{1, ..., m\}$ and $N = \{1, ..., n\}$. Let R_i be the number of cells containing a ship in row i, for $i \in M$. Let C_j denote the number of cells containing a ship in column j, for $j \in N$. Let S_l represent the number of ships of length l that must be in the solution. Let T be the initial $m \times n$ grid containing the clues, such that

$$T_{ij} = \begin{cases} k, & \text{if cell } (i,j) \text{ contains celltype } k, \\ 100, & \text{otherwise.} \end{cases}$$

We then define the following parameters to describe the grids of A,

$$X_{ija} = \begin{cases} 1, & \text{if cell } (i,j) \text{ contains a ship in grid } a \in A, \\ 0, & \text{otherwise,} \end{cases}$$

$$E_{ija} = \begin{cases} 1, & \text{if } X_{ija} = 1,\ X_{i\ j+1\ a} = 1,\ X_{i+1\ j\ a} = 1 \text{ or } X_{i+1\ j+1\ a} = 1, \\ 0, & \text{otherwise,} \end{cases}$$

for $i \in M$ and $j \in N$. Our decision variables are

$$y_a = \begin{cases} 1, & \text{if grid } a \in A \text{ is included in solution,} \\ 0, & \text{otherwise.} \end{cases}$$

The GBM is then as follows,

$$\begin{aligned} \text{minimise } & \mathbf{0}^t x && && \text{(4.1a)} \\ \text{subject to } & \sum_{a \in A} y_a A_{ija} = T_{ij}, && \{i \in M, j \in N | T_{ij} \neq 100\}, && \text{(4.1b)} \\ & \sum_{a \in A} \sum_{j \in N} y_a X_{ija} = R_i, && \forall i \in M, && \text{(4.1c)} \\ & \sum_{a \in A} \sum_{i \in M} y_a X_{ija} = C_i, && \forall j \in N, && \text{(4.1d)} \\ & \sum_{a \in A} E_{ija} y_a \leq 1, && \forall i \in M,\ j \in N, && \text{(4.1e)} \\ & \sum_{a \in P_l} y_a = S_l, && \forall l, && \text{(4.1f)} \\ & y_a \in \{0, 1\}\ \forall a \in A. && && \text{(4.1g)} \end{aligned}$$

The BSP is a feasibility problem, as such, a solution must only satisfy all the constraints. Thus, the objective function Equation (4.1a) minimises $\mathbf{0}$. Equation

(4.1b) ensures that all the clues are included. Equation (4.1c) ensures that our solution has the correct number of ship pieces per row. Equation (4.1d) does the same for the columns. Equation (4.1e) makes sure that there are no ships which are adjacent or overlapping. It does so through the use of the parameter E_{ija}. This parameter checks surrounding cells and is equal to 1 if there is a ship piece in any of them. If two different grids a both have $E_{ija} = 1$ at the same (i, j) then these two ships are incompatible. Therefore, for each (i, j) the sum of E_{ija} corresponding to our chosen grids must always be less than or equal to 1. Equation (4.1f) ensures that we have the correct number of ships for each ship length. For this we simply make sure the sum of the y_a for each P_l aligns with S_l.

4.2.2 Cell Based Model

We will be comparing the GBM to a Cell Based Model (CBM). This model was formulated by Meuffels et al. [19, pp. 157-158]. The CBM requires more constraints and variables since we have to additionally constrain that the ships are formed properly and fit within the grid outline. The CBM inspects each cell to decide which celltype it should be.

Let R_i and C_j denote the number of cells containing a ship in row i and column j respectively where $i \in M$ and $j \in N$. Let T_{ijk} be the initial starting grid where $k \in \{0, ..., 6\}$ represents the celltypes. Let S_l be the number of ships needed of each length l. The decision variables for the CBM are defined as:

$$x_{ijk} = \begin{cases} 1, & \text{if cell } (i,j) \text{ contains symbol k,} \\ 0, & \text{otherwise,} \end{cases}$$

$$s_{ijl} = \begin{cases} 1, & \text{if a ship of length l starts in cell } (i,j), \\ 0, & \text{otherwise.} \end{cases}$$

The formulation of the CBM then follows

$$\text{minimise } \mathbf{0}^t x \tag{4.2a}$$

$$\text{subject to } \sum_k x_{i,j,k} = 1, \qquad \forall i \in M,\ \forall j \in N, \tag{4.2b}$$

$$x_{i,j,k} = T_{i,j,k}, \qquad \{i \in M, j \in N, k | T_{i,j,k} = 1\}, \tag{4.2c}$$

$$\sum_j \sum_{\{k|k\neq 0\}} x_{i,j,k} = R_i, \qquad \forall i \in M, \tag{4.2d}$$

$$\sum_i \sum_{\{k|k\neq 0\}} x_{i,j,k} = C_j, \qquad \forall j \in N \tag{4.2e}$$

$$s_{i,j,1} = x_{i,j,6}, \qquad \forall i \in M,\ \forall j \in N, \tag{4.2f}$$

$$
\begin{aligned}
& ls_{i,j,l} \leq x_{i,j,1} + x_{i,j+l-1,2} \\
& \quad + \sum_{c=j+1}^{j+l-2} x_{i,c,5} + x_{i,j,3} \\
& \quad + x_{i+l-1,j,4} + \sum_{r=j+1}^{i+l-2} x_{r,j,5}, && \{i \in M, j \in N, l \in L | l \neq 1\}, && \text{(4.3a)} \\
& \sum_i \sum_j s_{i,j,l} = S_l, && \forall l \in L, && \text{(4.3b)} \\
& \sum_{k|k\neq 0} x_{i,j,k} + \sum_{k|k\neq 0} x_{i+1,j+1,k} \leq 1, && \forall i \in M,\ \forall j \in N, && \text{(4.3c)} \\
& \sum_{k|k\neq 0} x_{i,j,k} + \sum_{k|k\neq 0} x_{i-1,j+1,k} \leq 1, && \forall i \in M,\ \forall j \in N, && \text{(4.3d)} \\
& x_{i,j,1} \leq x_{i,j+1,2} + x_{i,j+1,5}, && \forall i \in M,\ \forall j \in N, && \text{(4.3e)} \\
& x_{i,j,1} \leq x_{i,j-1,0}, && \{i \in M, j \in N | j \neq 1\}, && \text{(4.3f)} \\
& x_{i,j,2} \leq 1 - x_{i,j+1,k}, && \{i \in M, j \in N | j \neq J\}, && \text{(4.3g)} \\
& x_{i,j,2} \leq x_{i,j-1,1} - x_{i,j-1,5}, && \forall i \in M,\ \forall j \in M, && \text{(4.3h)} \\
& x_{i,j,3} \leq x_{i+1,j,4} + x_{i+1j,5}, && \forall i \in M,\ \forall j \in N, && \text{(4.3i)} \\
& x_{i,j,3} \leq x_{i-1,j,0}, && \{i \in M, j \in N | i \neq 1\}, && \text{(4.3j)} \\
& x_{i,j,4} \leq x_{i+1,j,0}, && \{i \in M, j \in N | i = I\}, && \text{(4.3k)} \\
& x_{i,j,4} \leq x_{i-1,j,3} + x_{i-1,j,5}, && \forall i \in M,\ \forall j \in N, && \text{(4.3l)} \\
& x_{i,j,5} \leq x_{i-1,j,3} + x_{i,j+1,2} + x_{i,j+1,5}, && \forall i \in M,\ \forall j \in N, && \text{(4.3m)} \\
& x_{i,j,5} \leq x_{i-1,j,3} \\
& \quad + x_{i-1,j,5} + x_{i,j-1,1} \\
& \quad + x_{i,j-1,5}, && \forall i \in M,\ \forall j \in N, && \text{(4.3n)} \\
& x_{i,j,5} \leq x_{i+1,j,4} \\
& \quad + x_{i+1,j,5} + x_{i,j-1,1} \\
& \quad + x_{i,j-1,5}, && \forall i \in M,\ \forall j \in N, && \text{(4.3o)} \\
& x_{i,j,5} \leq x_{i+1,j,4} \\
& \quad + x_{i+1,j,5} + x_{i,j+1,2} \\
& \quad + x_{i,j+1,5}, && \forall i \in M,\ \forall j \in N, && \text{(4.3p)} \\
& x_{i,j,6} = 1 - x_{i,j-1,6}, && \{i \in M, j \in N | j \neq 1\}, && \text{(4.3q)} \\
& x_{i,j,6} = 1 - x_{i-1,j,6}, && \{i \in M, j \in N | j \neq 1\}, && \text{(4.3r)} \\
& x_{i,j,k} \in \{0,1\}\ \forall i \in M,\ \forall j \in N,\ \forall k, && && \text{(4.3s)} \\
& s_{i,j,l} \in \{0,1\}\ \forall i \in M,\ \forall j \in N,\ \forall l \in L. && && \text{(4.3t)}
\end{aligned}
$$

Equation (4.2b) ensures that each cell only contains a single symbol. Equation (4.2c) makes sure the cells known from the initial grid are used. Equations (4.2d) and (4.2e) require that each row and column contains the correct number of ship parts. Equation (4.2f) ensures that if the symbol for a single ship is used in cell (i, j) then $s_{i,j,1}$ is equal to one and Equation (4.3a) does the same where if the beginning of a ship is contained in cell (i,j) of length $l > 1$ then $s_{i,j,l} = 1$. Equation (4.3b) requires that the correct number of ships for each ship length are contained within the grid. Equations (4.3c) and (4.3d) establish that ships are not diagonally adjacent. Equations (4.3e - 4.3p) make sure that only the correct symbols are attached to each other. Equations (4.3q) and (4.3r) ensure that ships of length one do not touch.

4.3 Implementation

For the GBM the sets of all the ship positions had to be generated. Unlike the model, however, only the set A was used as it was easier to implement constraint four by indexing using a set containing the start point and end point of each P set and using A in conjunction with this. A separate function to compute $E_{i,j,a}$ was also made for convenience.

The CBM model by Meuffels et al. had some initial incompatibilities with the implementation using Gurobi that had to be adjusted. Notably, the model had specified that if any (i, j) were out of bounds, that the value of $x_{i,j}$ would be 0. To avoid errors in the implementation, indexing was handled with care to avoid out of bounds $x_{i,j}$ in the constraints. These changes did not affect the outcome of the model.

In order to check validity of the models, puzzles used in the solver and for testing were taken from the Battleship puzzle website where puzzles and their solutions are available [23].

4.4 Comparison of the Models

To compare the models multiple puzzles with different difficulties and grid sizes were tested. These grid sizes, along with the number of ships of each length that they must contain are described in Table 4.2.

	Ship Length				
Grid Size	**1**	**2**	**3**	**4**	**5**
6×6	3	2	1	0	0
8×8	3	3	2	1	0
10×10	4	3	2	1	0
15×15	5	4	3	2	1

Table 4.2: Number of ships of each length for each grid size.

Table 4.3 displays the computation times for these differing puzzle instances for the GBM and CBM.

We observe that for both models the computation times increase slightly as the size of the puzzles grow. In saying this the GBM consistently outperforms

		Time (s)	
Size	**Level**	**GBM**	**CBM**
6x6	Easy	0.01	0.01
6x6	Hard	0.01	0.12
8x8	Easy	0.02	0.08
8x8	Hard	0.02	0.22
10x10	Easy	0.01	0.02
10x10	Hard	0.01	0.05
15x15	Easy	0.03	0.21
15x15	Hard	0.03	0.15

Table 4.3: The computation times of the GBM and CBM on varying puzzle instances.

the CBM. It is interesting to note that the CBM takes longer for harder puzzles of the same size, while this is not the case with the GBM.

In [19], it was found that for very large grids, 30×20, the CBM took multiple hours to solve. This is due to the large number of constraints necessary to construct the model using the cell-based approach. Specifically, the need to ensure the ships fit on the defined grid.

A noteable exception, was the reduction in computation time for the CBM model between the 8×8 and 10×10 grids. This may be attributed to the fact that a similar number of ships were being placed on a larger grid.

During this testing, it was observed that if we had (4.1b) as our first constraint in the GBM the computation time drastically decreased compared to its original position as the final constraint. Since all constraints in IP are satisfied by a feasible solution simultaneously, this difference in computation time is likely due to the methods applied by the Gurobi solver.

4.5 Conclusion

From our investigation, it can be seen that the model based on possible ship positions was more efficient. Although the BSP at a glance seemed similar to Sudoku, deciding the value of each cell required a huge amount of constraints. As such, while this approach worked well for Sudoku, it was not the best method for a BSP model. Not only did the GBM circumvent the need for any constraints regarding the structure of ships, it was also more efficient. From this it can be seen how important the choices made when constructing a model are for its efficiency as well as the value of finding strategies to reduce the required number of constraints as this simplifies the model.

Chapter 5

Circuit Board Game

5.1 Introduction

The Circuit Board Game was created by Kousha Nejad and was featured as a New York Times puzzle in 2021 [15]. The puzzle consists of a square grid where each cell contains a point or is blacked out. The goal of the game is to connect all of the points without creating any enclosed areas. A point can connect to either one or three others that are vertically or horizontally adjacent to it. The initial grid contains lines that must be in the solution. Figure 5.1 depicts a possible starting grid of the puzzle.

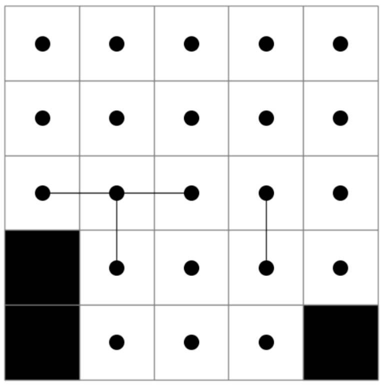


Figure 5.1: Circuit Board Game on a 5×5 grid to be solved.

A notable similarity to the Sudoku puzzle is that this is a grid based game. However, as we explore how this puzzle can be formulated as an IP model we will observe that the game can be modelled independently from the grid. Thus, beyond this surface level similarity, the commonalities between how the Circuit Board Game and Sudoku are formulated are surprisingly limited.

5.2 The Models

In this chapter we will be examining two different IP models we created as well as an extension to the puzzle. The first model stems from the observation that solutions to this puzzle are spanning trees. The second model formulates the puzzle as a Minimum Cost Network Flow problem. Creating two different models allows us to compare them and find which one is a better formulation. Finally, we will discuss the model we created to solve the puzzle if it also required finding the shortest path between two points.

5.2.1 Spanning Tree Formulation

We will now consider this puzzle as a graph. To solve it we must construct a connected graph with no cycles where the degrees of each vertex are either one or three. In Chapter 2 we established that a tree is a connected graph with no cycles and a spanning tree is a tree that connects all vertices of a graph. In light of this, we formulated a model which incorporates creating a spanning tree while adhering to the degree constraints and ensuring that the given edges are included in the solution. Denote the following model as Circuit Model 1 (CM1).

We will refer to the given puzzle as graph $G(V, E)$, where $v \in V$ are vertices and $e \in E$ are edges in the graph. Note that for this formulation we are only considering choices that are 'possible'. This means that the set of vertices V does not include any blacked out cells and the set of edges E are the edges that satisfy the required positioning of two adjacent vertices. Let $e \in E$ be such that $e = (i, j)$ when edge e is connecting vertex i to vertex j, where $i < j$ and $i, j \in V$. Let H be the set of edges given in the puzzle that must be in the solution.

The decision variables for CM1 are

$$x_e = \begin{cases} 1, & \text{if edge } e \in E \text{ is included in the tree,} \\ 0, & \text{otherwise,} \end{cases}$$

and the auxiliary binary variable

$$y_i = \begin{cases} 1, & \text{if the degree of vertex } i \in V \text{ is 3,} \\ 0, & \text{if the degree of vertex } i \in V \text{ is 1.} \end{cases}$$

Thus, CM1 is

$$\begin{aligned}
\text{minimise} \quad & \sum_{e \in E} x_e && && \text{(5.1a)} \\
\text{subject to} \quad & x_e = 1, && \forall e \in H, && \text{(5.1b)} \\
& \sum_{\{e \in E \ \mid \ i \in e\}} x_e = 1 + 2y_i, && \forall i \in V, && \text{(5.1c)} \\
& \sum_{e \in E} x_e = |V| - 1, && && \text{(5.1d)} \\
& \sum_{\{e=(i,j) \in E \ \mid i \in N\}} x_e \leq |N| - 1, && \forall N \subset V, \ \emptyset \neq N, && \text{(5.1e)} \\
& x_e \in \{0, 1\} \ \forall e \in E, && && \text{(5.1f)} \\
& y_i \in \{0, 1\} \ \forall i \in V. && && \text{(5.1g)}
\end{aligned}$$

In CM1 the objective function, Equation (5.1a), minimises the number of edges included in the solution. To include the edges given in the initial puzzle we use Equation (5.1b) to set the corresponding decision variables to one. Next, we use Equation (5.1c) to establish a constraint on every vertex so that all degrees are either one or three. This is accomplished by adding all possible edges that include a certain vertex, denoted i, and using the associated auxiliary binary variable y_i to ensure that there are either one or three of these edges in the solution.

The following well-established constraints ensure that the feasible region only consists of spanning trees [4, p. 154]. Theorem 1 states that a graph is a tree if it contains no cycles and has $n - 1$ edges, where n is the number of vertices in the graph. This concept provides the basis for these constraints. Equation (5.1d) imposes the requirement that the sum of all edges included in the solution must be one less than the number of vertices in the graph. Then Equation (5.1e) enforces that the graph does not include any cycles. These are referred to as Subtour Elimination Inequalities (SEI) [28, p. 8] and come from the standard formulation of the Travelling-Salesman Problem [5, p. 398]. The SEI ensure that, given a subset of $|N|$ vertices, there is not a cycle connecting those vertices since a cycle would require at least $|N|$ edges to be adjacent to these vertices and this constraint only allows up to $|N| - 1$.

Design Decisions

We have made certain modelling choices to simplify the formulation. As such, the initial sets of vertices and edges exclude the blacked out cells so that there does not have to be constraints that consider them.

5.2.2 Minimum Cost Network Flow Formulation

Our next model, denoted Circuit Model 2 (CM2), solves the Circuit Board Game by formulating the model as a MCNF problem. CM2 is based on a model by Cheng-Wei Lu [15]. This model lacked clarity and included incorrect constraints, however it provided a framework to understand how this puzzle can be viewed as a MCNF problem, as such their structure was used to outline our formulation.

Similar to CM1 we will refer to the given puzzle as graph $G(V, E)$, where $v \in V$ are the vertices (skipping over blank ones) and E is the set of possible edges. Let $e \in E$ be such that $e = (i, j)$ when edge e is connecting vertex i to vertex j, where $i < j$. Let $a \in A$ be the possible arcs. Let $a \in A$ be such that $a = (i, j)$ when arc a is connecting vertex i to vertex j, $\forall i, j \in V$. Thus, for an arc to be possible, either it, or the arc in the other direction must be an edge in the edge set E. Again, let H be the set of edges that are given in the puzzle.

Additionally, the MCNF formulation requires the parameter S_i to represent the supply of vertex i. This is defined such that one vertex will have $S_i = |V| - 1$, and all other vertices will have $S_i = -1$. Therefore, this model has one source vertex and the rest are demand vertices. Let the Big-M constant be $M = 99$.

The decision variables are $x_{ij} \in \mathbb{R}^+$, the flow over each arc $\forall (i,j) \in A$,

$$z_{ij} = \begin{cases} 1, & \text{if either } x_{ij} \text{ or } x_{ji} \text{ is greater than 0,} \\ 0, & \text{otherwise,} \end{cases}$$

and

$$y_i = \begin{cases} 1, & \text{if the degree of vertex } i \in V \text{ is 3,} \\ 0, & \text{if the degree of vertex } i \in V \text{ is 1.} \end{cases}$$

Hence, CM2 is

$$\begin{aligned}
\text{minimise} \quad & \sum_{(i,j)\in E} z_{ij} && && \text{(5.2a)} \\
\text{subject to} \quad & \sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = S_i, \forall i \in V, && && \text{(5.2b)} \\
& x_{ij} + x_{ji} \le M z_{ij}, && \forall (i,j) \in E, && \text{(5.2c)} \\
& M(x_{ij} + x_{ji}) \ge z_{ij}, && \forall (i,j) \in E, && \text{(5.2d)} \\
& \sum_{\{e\in E \ \mid \ i\in e\}} z_e = 1 + 2y_i, && \forall i \in V, && \text{(5.2e)} \\
& z_{ij} = 1, && \forall (i,j) \in H, && \text{(5.2f)} \\
& x_{ij} \in \mathbb{R}^+ \ \forall i,j \in V, && && \text{(5.2g)} \\
& z_{ij} \in \{0,1\} \ \forall i,j \in V, && && \text{(5.2h)} \\
& y_i \in \{0,1\} \ \forall i \in V. && && \text{(5.2i)}
\end{aligned}$$

Once again the objective function, Equation (5.2a), minimises the number of edges included in the solution. In a typical MCNF formulation the objective function would include the arcs and their associated costs, but in this case each arc is worth the same so the costs are all set to one and the decision variable must only consider edges. Minimising the number of edges will prevent any cycles from being included in the solution. The first constraints, Equation (5.2b) ensure no flow is lost or created across the network. They are classified as flow conservation equations [3, p. 249]. They ensure that the net flow into and out of all vertices matches their allotted supply. Since there is only one source vertex and the rest are demand vertices, this constraint also ensures that the solution is a connected graph. The flow conservation equations imply that a path must exist between every demand vertex and the source, which can only occur in a connected graph.

Equations (5.2c) and (5.2d) are either-or constraints that relate the decision variables z_{ij}, which are for all edges, to the decision variables x_{ij} which are for all arcs. We require that $z_{ij} = 1$ if either x_{ij} or x_{ji} is greater than 0 and $z_{ij} = 0$ otherwise. Using $M = 99$, Equation (5.2c) enforces that $z_{ij} = 1$ if either x_{ij} or x_{ji} is greater than 0, while Equation (5.2d) ensures that if $x_{ij} = x_{ji} = 0$ then $z_{ij} = 0$. In order to ensure that all vertices have degree of either one or three, Equation (5.2e) uses the auxiliary decision variable y_i to guarantee that the sum of edges that are adjacent to vertex i is equal to one or three, for every $i \in V$. Finally, Equation (5.2f) includes the edges given by the puzzle in every feasible solution by setting the corresponding decision variables to one.

Design Decisions

Similar to CM1, this model does not need to contain constraints for the blacked out cells since the given information does not include them as viable choices.

Furthermore, the choice of $M = 99$ does not restrict any feasible solutions whilst also not introducing unnecessarily large numbers into the formulation. A large number is not an issue for IP models but can cause computational problems for the solver. We know that $M = 99$ is sufficiently large in practice because the flow decision variables x_{ij} are linked to the supply parameters, defined as either $|V| - 1$ or -1. Since flow cannot be lost or gained it is bounded by

$$0 \leq x_{ij} + x_{ji} \leq |V| - 1.$$

We will be implementing this model on grids with sizes up to 10×10. Since the puzzle is typically on a 5×5 grid, this allows us to investigate what occurs in larger instances. A puzzle on a 10×10 grid has 100 vertices or less so the supply will be at most 99. Therefore, the choice of $M = 99$ is appropriate for the puzzles we will test but will need to be increased analogously for larger puzzles.

5.2.3 Shortest Path Circuit Board

To consider an additional layer of complexity to the puzzle we will add a supplementary rule. Now, in order to solve the puzzle we must also ensure that the solution includes the shortest path connecting vertex s to vertex t. This is an example of a Shortest Path Problem (SPP). A SPP involves finding the path of minimum length connecting two vertices and is often applied to directed graphs [4, p. 137]. Figure 5.2 depicts an example of this augmented puzzle. The player must connect the blue vertices using the smallest number of edges possible whilst also satisfying the other rules of the game.

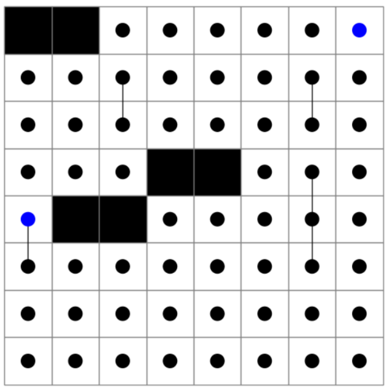


Figure 5.2: A Shortest Path Circuit Board Game on an 8×8 grid.

The model created to solve this extended puzzle builds off of CM1. We denote this model as Circuit Model 3 (CM3). We will use the notation from CM1 with the addition of the endpoints s and t. The decision variables are then as follows:

For the puzzle as a whole we have

$$x_e = \begin{cases} 1, & \text{if edge } e \in E \text{ is included in the tree,} \\ 0, & \text{otherwise,} \end{cases}$$

and the auxiliary binary variable

$$y_i = \begin{cases} 1, & \text{if the degree of vertex } i \in V \text{ is 3,} \\ 0, & \text{if the degree of vertex } i \in V \text{ is 1,} \end{cases}$$

and for the SPP we have

$$p_e = \begin{cases} 1, & \text{if edge } e \in E \text{ is included in the path connecting } s \text{ and } t, \\ 0, & \text{otherwise,} \end{cases}$$

and the auxiliary binary variable

$$u_i = \begin{cases} 1, & \text{if vertex } i \in V \text{ is included in the path connecting } s \text{ and } t, \\ 0, & \text{otherwise.} \end{cases}$$

The formulation is then

$$\begin{aligned}
\text{minimise} \quad & \sum_{e \in E} x_e + \sum_{e \in E} p_e && && (5.3a) \\
\text{subject to} \quad & x_e = 1, && \forall e \in H, && (5.3b) \\
& \sum_{\{e \in E \ \mid \ i \in e\}} x_e = 1 + 2y_i, && \forall i \in V, && (5.3c) \\
& \sum_{e \in E} x_e = |V| - 1, && && (5.3d) \\
& \sum_{\{e=(i,j) \in E \ \mid i \in N\}} x_e \leq |N| - 1, && \forall N \subset V, \ \emptyset \neq N, && (5.3e) \\
& p_e \leq x_e, && \forall e \in E, && (5.3f) \\
& \sum_{\{e \in E \ \mid \ u \in e\}} p_e = 1, && \forall u \in \{s, t\}, && (5.3g) \\
& \sum_{\{e \in E \ \mid \ i \in e\}} p_e = 2u_i, && \forall i \in V \setminus \{s, t\}, && (5.3h) \\
& p_e, x_e \in \{0, 1\} \ \forall e \in E, && && (5.3i) \\
& u_i, y_i \in \{0, 1\} \ \forall i \in V. && && (5.3j)
\end{aligned}$$

This model is inspired by a single-flow formulation [21, p. 6] of a SPP but adapted for an undirected graph. The objective function, Equation (5.3a), now additionally minimizes the number of edges included in the path between vertex s and t as well as the total number of edges in the solution. Equation (5.3f) requires that the chosen path is represented in the solution by ensuring that if an edge is included in the path that edge must also be in the spanning tree.

Equations (5.3g) and (5.3h) ensure that the path connects s and t by establishing specialised requirements for the decision variables corresponding to which edges are in the path, p_e. First, Equation (5.3g) specifies that the vertices s and t must be adjacent to exactly one edge in the path by summing over their adjacent edges and ensuring that only one is part of the path. These are examples of the set partitioning constraints utilised in the IP model for Sudoku. They also imply that edges on the path must exist between s, t and other vertices in the graph establishing s and t as the endpoints. Then we leverage the fact that on a path all vertices except the endpoints must be adjacent to at least two edges. Since these decision variables consider only the path and not the graph as a whole, Equation (5.3h) implies that all vertices in the graph except for s and t, must be adjacent to either zero or two edges on the path. This is accomplished by summing over the edges adjacent to all vertices except for s and t, and creating the requirement that the sum is equal to either zero or two using the auxiliary binary variable u_i.

5.3 Implementation

The models described above have been implemented to reflect the formulations previously laid out, but steps have been taken to reduce their computation time. All three models include constraints which fix a decision variable in order to satisfy the given puzzle. These constraints are Equations (5.1b), (5.2f) and (5.3b) which fix the binary decision variables to one. To reduce the size of the model, these equations were not implemented as constraints but as lower bounds.

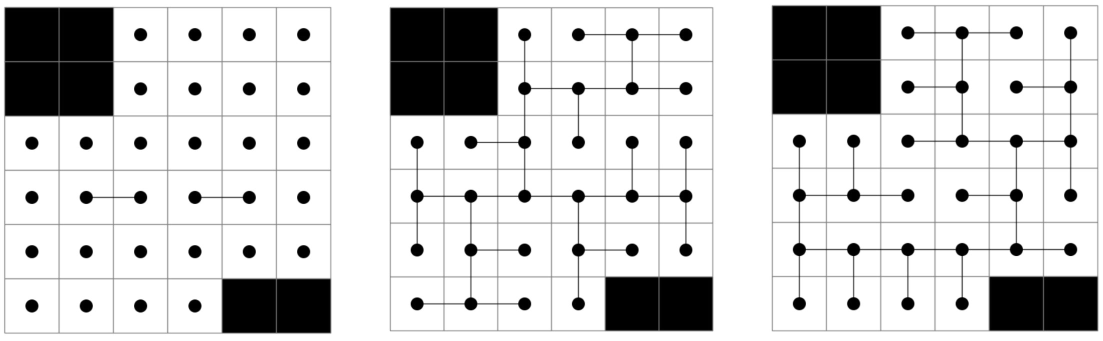

(a) 6×6 unsolved puzzle. (b) Solution found using CM1. (c) Solution found using CM2.

Figure 5.3: The given puzzle as well as the optimal solutions found using CM1 and CM2.

A consideration in terms of computing time were the SEI provided by Equation (5.1e) in CM1. The number of these constraints is exponential in terms of the number of vertices in the graph; if a graph has n vertices this would imply that there are $2^n - 2$ SEI. To avoid having to introduce such a large number of constraints we apply the constraints as lazy constraints. Lazy constraints are used to remove integer solutions that otherwise would have been feasible [17]. This is achieved by solving the model without the SEI and then checking if this solution contains a cycle. If one exists, then a lazy constraint is added to remove it. This constraint is of the form of Equation (5.1e) with the set of vertices N

being those in the cycle that was detected. The process requires the use of callbacks. A callback is a function that allows us to communicate with the solver and check if the current solution satisfies constraints not included in the model. If indeed these constraints are not satisfied they are added to the model [20].

When implementing our models we have used puzzles we designed ourselves. Figure 5.3 depicts a puzzle on a 6×6 grid as well as the solutions found using CM1 and 2 respectively. Notably, Figure 8.1b and 5.3c show that for this puzzle, our models find different optimal solutions. Although one correct solution is not actually preferable to another, this does raise the question of which model is better. The performance of the models will be hence be examined in Section 5.4.

Finally, we implemented CM3. The solution to the puzzle given by Figure 5.2 is displayed in Figure 5.4. Notably, the shortest path connecting the blue vertices, coloured red, is longer than other paths that connect these vertices. The actual shortest path would contain eleven edges while this path contains fifteen. This is caused by the edges given in the initial grid that must be included in the solution. Simply, a path with less than fifteen edges is not feasible for this puzzle.

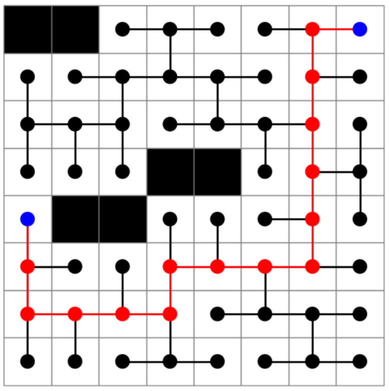


Figure 5.4: The solution to the Shortest Path Circuit Board shown in Figure 5.2. The shortest path connecting the blue vertices is coloured red.

For all three models it was easy to confirm visually that the solution found did in fact satisfy the rules of the Circuit Board Game. For CM3 we also had to check that the shortest path found by the model could not be reduced further without making the puzzle infeasible. This was accomplished by giving the model all possible shorter paths as well as the edges given in the original puzzle to confirm that the problem was indeed infeasible. This was often easy to deduce by hand.

5.4 Model Comparison

CM1 and 2 both solve the same problem but using vastly different approaches. CM1 exploits the graphical properties of the puzzle while CM2 implements a common optimization technique. At first glance, CM2 appears superior to CM1 since its constraints do not increase exponentially with the number of vertices. However, the SEI in CM1 can be implemented using callbacks to avoid this exponential behaviour while all constraints in CM2 are required to establish the network. To assess which formulation is better we considered two different aspects; which model is faster and which model's LR has a greater objective value.

The computation times for the models have been measured for five different puzzles for each grid size ranging from 3×3 to 10×10. This is displayed in Table A.1 and A.2 for CM1 and 2 respectively. They show that CM1 and 2 perform similarly until the number of vertices reaches thirty. After this point CM2 slows down, and for puzzles with 62 vertices or more, consistently hits the time limit of thirty minutes. Once the time limit is reached, the optimality gap increases as the number of vertices increase. While CM1 does slow down as the grids increase in size, the time it takes to solve the puzzle remains under eleven seconds. Therefore, we can conclude that both models have reasonable computation times for puzzles with less than 45 vertices, but unlike CM2, CM1 performs well for larger grids.

We evaluated the LR of CM1 and CM2 for one puzzle on grid sizes ranging from 3×3 to 10×10, displayed in Table A.3. It is important to note that Equation (5.1d) fixes the objective value of CM1 to be $|V| - 1$. So, the LR of CM1 will also have this objective value. This provides a baseline for how close the solution to the LR of CM2 is to a feasible integer solution. We note that the objective value of CM1, denoted z^*_{CM1}, is consistently greater than that of CM2, denoted z^*_{CM2}. A key observation is that z^*_{CM2} decreases from over 60% of the corresponding z^*_{CM1} to almost 50% of z^*_{CM1} as the number of vertices increases. This means that as the puzzles are increasing in size, the LR of CM2 is getting further from a feasible integer solution. The objective value of the LR of CM2 being smaller than that of CM1 indicates that CM1 is a better formulation than CM2.

5.5 Conclusion

Our investigation into different formulations for the Circuit Board Game has revealed that, when implemented efficiently, a model that exploits the graphical features of the game is superior to one that requires a simpler implementation. CM1 was faster than CM2 and also a better formulation. Additionally, we explored the incorporation of a SPP into the game. This was successful and revealed how different methods including the creation of a spanning tree, and finding a shortest path can work simultaneously.

Although this game may seem to have similarities with Sudoku, in terms of the grid and initial clues, the formulations were vastly different. The fact that the game is played on a grid had no impact on how the variables were defined. Instead, the models use the given sets of vertices and edges which makes them independent of the grid. This careful setup led to the models requiring less constraints and being simpler overall. These characteristics make models easy to adapt to consider different rules.

Chapter 6

Hashi

6.1 Introduction

Hashiwokakero, more commonly known as 'Hashi' or 'Bridges', is a Japanese Pencil Puzzle (JPP) created in 1990 by the Japanese gaming company Nikoli [16, p. 61]. JPPs are one player logic puzzles whose popularity lies in their cultural independence, since most of the puzzles are entirely language neutral [13, p. 391]. This neutrality allows them to be solved entirely using deduction, and it is indeed this deductive element that makes them exceptional candidates for optimization.

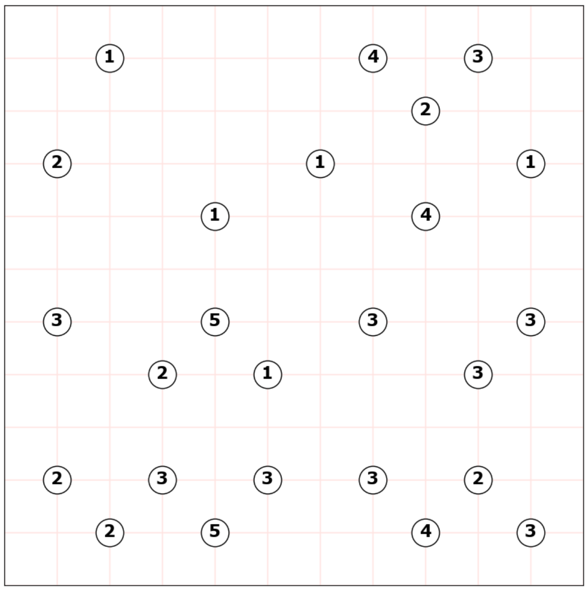


Figure 6.1: Unsolved twenty-five island Hashi puzzle.

Having previously examined the game of Sudoku, arguably the most notable example of a JPP, we will adapt upon the set partitioning constraints used, whilst introducing additional constraints to address the further combinatorial intricacies of Hashi.

The goal of Hashi is to create a connected system of 'islands' by drawing 'bridges' between them. The puzzle commences with islands placed in their fixed positions on the intersections of the gridlines. Each island has a number inside them from the set $\{1, ..., 8\}$, representing the number of bridges the islands possess in the solution. Figure 6.1 depicts an example of a possible Hashi starting grid, with twenty-five islands.

The puzzle is solved when all the islands have been connected in such a way that the labels on them are exactly adhered to and the whole system is connected. Additionally, all bridges must be either horizontal or vertical, with no bridge intersecting another, and any pair of islands having a maximum of two bridges connecting them.

Although the Hashi puzzle takes place on a grid, we investigate how shifting the focus of our formulation to consider the islands themselves offers the potential for simplification of our constraints. In this way we highlight the importance of alternative definitions of decision variables, to deal with redundancies, which is unlike the challenges faced in the Sudoku puzzle.

6.2 The Hashi Model

In this chapter we will formulate an IP model used to solve the game of Hashi given a feasible starting grid. From here we will further build upon our model, investigating whether the inclusion of additional logical constraints reduces the complexity of the optimization.

6.2.1 Grid Formulation

The Hashi grid is not constrained to a specific shape, its dimensions ultimately depend on the placement of the islands. To this extent, we will work with right quadrilateral grids, whose size is just great enough to contain the islands. Specifically, an edge row or column only exists in the puzzle grid if it contains an island, however inner rows or columns can be empty without consequence.

We note that, dependent on the puzzle instance, certain solutions have largely untouched rows and columns. Thus, to ensure that we are not creating obsolete decision variables and running wasted computation we never formulate the grid itself. Instead, we think of our puzzle in terms of our island positions, so that we do not consider individual squares, only whether two islands have the potential to be bridged, based on the rules previously outlined.

6.2.2 Puzzle Initialization

We will define a Hashi puzzle uniquely by the positions and labels of its islands, namely the puzzle $H(P, L, B)$, where P are the positions of the islands, L are their labels and B are the 'possible' bridges that can be placed on the grid.

Let k be the number of islands on the starting grid. Then we refer to islands by their unique index, an integer in the set $\{0, ..., k-1\}$, which for clarity throughout our model formulation will be referred to using a, b, c or d, unless a specific index is required.

The assignment of these island indices is sequential. It follows that an island precedes (has a lower index than) all islands that it occurs to the left of and/or above. This choice of indexing ensures that there are no unnecessary clauses in our constraints, as we are able to more easily compare the relative positions of two islands without needing to refer to their position tuple.

In saying this, our island positions are represented by the list of tuples P, such that $P_a = (i_a, j_a)$ gives the position of island a in terms of the ith row and jth column of the grid. The labels of the islands are a list, L, with each element in the range $\{1, ..., 8\}$. The indexing of our islands is aligned within P and L. Specifically, we have that $|P| = |L| = k$, with the position tuple P_a and the label L_a referring to the same island.

In this formulation we only consider the bridges, B, that are 'possible' under our constraints. The term 'possible' refers to the bridges that adhere to the restrictions of adjacency and orientation. In graphical terms, this means possible bridges do not 'jump' over islands, hence they can only connect to the first island met when traversing the horizontal and vertical axes. In this regard, any island can be part of no more than four of the possible bridges in B. We also note that, at this point, we do not consider the fact that many of these possible bridges would intersect, as only a subset of them will appear in the solution.

A specific bridge, $q \in B$ is hence referred to by its two endpoint islands, specifically, the tuple $q = (a, b)$ is the bridge q that connects the islands a and b. As the bridges are undirected, B does not contain duplicates, and by convention $a < b$.

6.2.3 The Model

The decision variables for the model are x^n for $n \in \{0, 1, 2\}$ and y. These x^n are binary decision variables of length $|B|$ which are used to formulate the occurrences of the different bridges, namely 'no bridge' ($n = 0$), 'single bridge' ($n = 1$) and 'double bridge' ($n = 2$). For a given bridge $q = (a, b) \in B$, x_q^n defines whether there exists n bridge(s) between the islands a and b. Formally,

$$x_q^n = \begin{cases} 1, & \text{if } \exists n \text{ bridge(s) between } q = (a, b), \\ 0, & \text{otherwise.} \end{cases}$$

Then the following components are defined to ensure connectivity in the solution. We use a MCNF model to ensure that the solution is a connected graph so we define A to be the set of all possible arcs. In this way, A is formed of all possible bridges $q = (a, b) \in B$ and their reversals $q' = (b, a)$. Then we define the parameter S_a, $\forall a \in \{0, ..., k-1\}$ as the supply of each of the islands. This is initialised such that there is one source island with $S_0 = k - 1$, and all other islands are demand islands with $S_a = -1$. As a result, we define the final decision variable $y_q \in \mathbb{R}^+$ to be the flow over each arc $\forall q \in A$.

In our formulation we also refer to the property o in relation to a bridge. This defines the orientation of a bridge q, such that

$$o_q = \begin{cases} 1, & q \text{ is vertical,} \\ 0, & q \text{ is horizontal.} \end{cases}$$

The Hashi Model is then defined as follows, with $q, r \in B$ such that $q = (a, b)$, $r = (c, d)$, and $q' \in A$ such that $q' = (b, a)$. Note the Big-M constant is $M = 126$.

$$
\begin{aligned}
&\text{minimise } \mathbf{0}^T\mathbf{x} && && \text{(6.1a)}\\
&\text{subject to } \sum_{\substack{a\in\lambda\\ \lambda\in B}} x_a^1 + 2x_a^2 = L_a, && \forall a \in \{0, ..., k-1\} && \text{(6.1b)}\\
&x_q^0 + x_q^1 + x_q^2 = 1, && \forall q \in B && \text{(6.1c)}\\
&x_q^0 + x_r^0 \geq 1, && \forall q, r \in B && \text{(6.1d)}\\
& && \text{s.t. } o_q = 0,\ o_r = 1, && \\
& && a \neq b \neq c \neq d, && \\
& && a > c, && \\
& && b < d, && \\
& && j_a < j_c, && \\
& && j_b > j_c. && \\
&\sum_{q\in A} y_q - \sum_{q'\in A} y_{q'} = S_a, && \forall a \in \{0, ..., k-1\} && \text{(6.1e)}\\
&y_q + y_{q'} \leq M(1 - x_q^0), && \forall q \in B && \text{(6.1f)}\\
&y_q + y_{q'} \geq 1 - x_q^0, && \forall q \in B && \text{(6.1g)}\\
&x_q^n \in \{0, 1\}, && \forall q \in B, && \text{(6.1h)}\\
&y_q \in \mathbb{R}^+, && \forall q \in A. && \text{(6.1i)}
\end{aligned}
$$

We will now reason why every constraint was used to formulate the puzzle. The first of these, Equation (6.1b), ensures that each island has the exact number of bridges connected to it as given by its label L_a. In this constraint, λ is a subset of the set of possible bridges B, such that every bridge in λ has island a as one of its endpoints. The summation over this subset is thus taking into account all bridges that could have been connected to a and determining if they exist as single bridges (x_a^1), double bridges ($2x_a^2$, to take into account the weight of two bridges), or not at all (given by the absence of x_0) in the solution.

Equation (6.1c) requires that any pair of adjacent islands have at most two bridges connecting them. For each of the bridges $q \in B$, the summation ensures that only one of the x^n decision variables has a value of 1, so that only one type of bridge occurs. This is a set partitioning constraint similar to that employed in the Sudoku model.

Equation (6.1d) ensures that no bridges intersect. Let $q, r \in B$ be two bridges that intersect, which is determined by the sub-conditions that follow, then we ensure that this intersection never occurs by constraining that at least one of x_q^0 or x_r^0 be 1, which means that at least one of bridges q and r does not occur in the solution. The sub-conditions that determine if a pair of possible bridges intersect have been refined to cover all possible cases of how two bridges are placed in relation to each other. Let $q = (a, b), r = (c, d) \in B$ be such that all islands a, b, c, d are distinct irrespective of order.

Case I: *Bridges q and r have the same orientation.*

In this case there is no way that the two bridges can intersect.
By contradiction, assume that q and r are of the same orientation and intersect. This means that q and r are in the same row/column and overlap. As a result one of the bridges must 'jump' over one island of the other bridge. This contradicts the structure of B, which says for a bridge to be possible, the end islands must be adjacent. Hence two possible bridges of the same orientation cannot intersect.

Case II *Bridges q and r have different orientations.*

Without loss of generality, we assume that q defines the horizontal bridge ($o_q = 0$) and r the vertical ($o_r = 1$). Following the outlined definition of a bridge, this means that island a occurs to the left of b ($a < b$), and island c occurs above d ($c < d$). Comparing the island indexes then gives us the ability to cover every possible position of the two bridges. The corresponding visualisations of these subcases are provided in Figure 6.2.

(a) $a < c$:

i $a < b < c < d$, implies q exists entirely to the right of and below p, hence an intersection is impossible.

ii $a < c < b < d$, thus c is 'on' bridge q. By construction, this would make q an impossible bridge as it would 'jump' over c, and no longer adhere to the adjacency constraint of possible bridges. As such an intersection of this form cannot exist.

iii $a < c < d < b$, contradicts our assumption that r is vertical, hence can be ruled out following the reasoning of Case I.

(b) $a > c$:

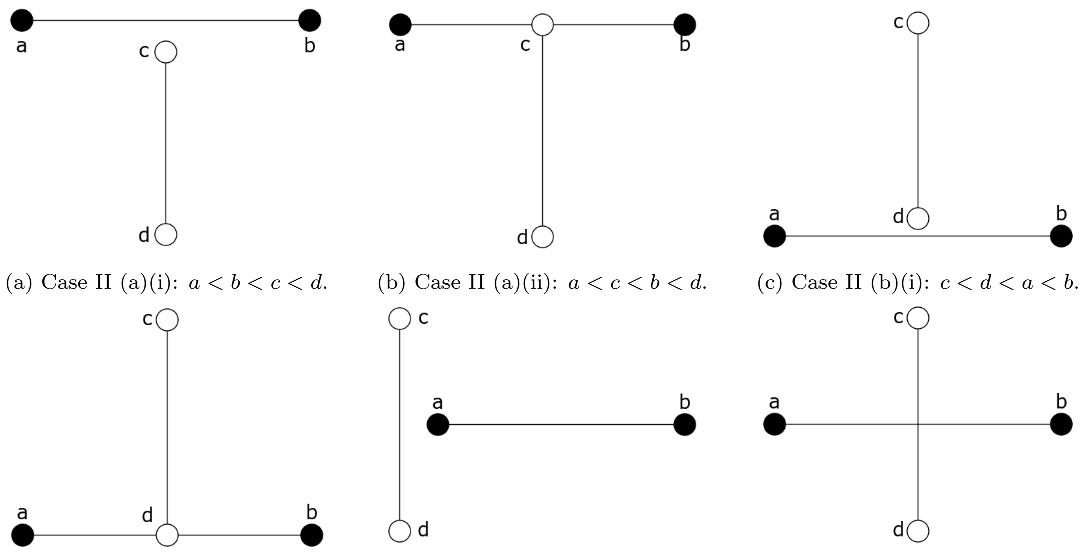
i $c < d < a < b$, implies r exists wholly to the left of and above q, hence intersection is impossible.

ii $c < a < d < b$, thus bridge q jumps r. Following analogously from subcase II(a)(ii), this would make q an impossible bridge, hence an intersection of this form never occurs.

iii $c < a < b < d$, describes two possibilities: either a and b lie on the same side of r ($b < c$), hence no intersection occurs, or they lie on different sides ($c < b$) and as a result an intersection of q and r occurs. Using the exact grid positions of the islands given in P, we isolate the specific conditions for this intersection. We have that $P_a = (i_a, j_a)$ describes the position of island a, with the other islands defined analogously. Then when bridges q and r occur in the form $a > c$, $a < b < d$, they will intersect if

$$j_a < j_c < j_b.$$

This covers the case entirely as neither $j_a \neq j_c$ nor $j_c \neq j_b$ (if these conditions were true it would imply that a or b, respectively, lie on r, making it an impossible bridge by construction rules).



(a) Case II (a)(i): $a < b < c < d$. (b) Case II (a)(ii): $a < c < b < d$. (c) Case II (b)(i): $c < d < a < b$.

(d) Case II (b)(ii): $c < d < a < b$. (e) Case II (b)(iii): $c < a < b < d$, no intersection. (f) Case II (b)(iii): $c < a < b < d$, intersection.

Figure 6.2: The graphical representation of the subcases described in Case II

Equations (6.1e - 6.1g) work simultaneously to ensure that the whole system is connected. These equations are a variation of a typical MCNF formulation. In this model we treat one island as the source while the others act as demand islands, as initialised by their respective supplies in S_a. Thus, Equation (6.1e) is the network flow constraint that ensure the preservation of flow across the solution. This is constrained by guaranteeing that the net flow through every island is equal to its supply. By allocating one island as the source and the rest as demand islands we ensure that the solution is a connected system. In particular, there is a flow between all islands and the source, meaning that paths exist between all island pairs, and as a result, the system is connected. Then Equations (6.1f and 6.1g) are necessary to align the decision variables for the bridges x_q^n to those for the related arcs, y_q and $y_{q'}$. Equation 6.1f is a Big-M constraint, utilising the parameter M, and together they create either-or logic between our decisions. In this way they dictate that when either y_q or $y_{q'}$ are greater than 0, that $x_q^0 = 0$, otherwise $x_q^0 = 1$. This relates all decision variables such that when there is flow between two islands, a bridge of some form must occur.

Note that, like Sudoku, the objective function of our Hashi model, Equation (6.1a), is set to minimise **0**. This is the case as our model describes a feasibility problem, the aim is to determine if there exists a solution that satisfies our constraints, instead of maximising any value. Graphically, this means we are looking for any point in our feasibile region.

Design Decision

In our model we use a Big-M constraint to model the flow across the network, represented by Equation (6.1f). To show that the choice of $M = 126$ does not restrict any of our feasible solutions we will first examine the left side of this inequality. This is the sum of the incoming and outgoing flow at any island. Since the flow can not be created or destroyed it is bounded by the supply. Thus,

$$y_q + y_{q'} \leq k - 1$$

since the supply in this network is either $k - 1$ or -1. The right side of the inequality is $M(1 - x_q^0)$ which is at most M since x_q^0 is binary. Therefore the inequality holds for all $M \geq k - 1$. When we test our model the largest Hashi board contains 127 islands. Therefore $M = 126$ is sufficiently large for our applications.

6.2.4 Effect of Adding 'Human-Deduced' Constraints

The game of Hashi is ultimately one reliant on human deduction, this means that through familiarity with the game we are able to spot patterns and, to this effect, combinations of island labels and placements that limit the possible bridges that can be placed, which we call the 'Human-Deduced' Constraints (HDC). For example, take an island with a label of '4' that has only two possible bridges (it could be a corner island or just be placed in such a way that it only has two adjacent islands), then this island must have 'double' bridges to both of these adjacent islands as this is the only way the label constraint can be satisfied.

This is an example of a deduction that can be made in the first layer of the puzzle. By this we refer to inferences regarding bridge placements made from solely looking at the labels and placements of islands on the starting grid. Thus, no deduction will be made using bridges that have been placed. What follows is an outline of these HDC, and the ramifications they have on the model's execution time when implemented.

We will be very specific about which constraints are added. These are only the deductions that can be made on 'first glance', as described above. If we were to allow for more layers of deduction this would sequentially end in the puzzle being solved entirely by deduction and not optimization. We have also limited the number of constraints so that each potential label in the set {1, ..., 8} has at most one constraint related to it, with the exception of '1' and '2', as these are trivial. These chosen constraints are deemed to be the simplest, but from them enough can be inferred to provide a starting point to the solution, without, again, defeating the need for optimization. The deductions then follow, along with their corresponding implications on the variables. Assume the bridge in question has index a, and let $S_a \subset B$ denote the subset of possible bridges that contain bridge a, such that $q \in S_a$ if $a \in q$ for the possible bridge tuple q.

- Any island of label '1' cannot be connected to another island of label '1'.

$$x_q^0 = 1, \quad \text{if } L_a = L_b = 1 \text{ and } q = (a, b) \in S_a. \tag{6.2}$$

- Any island of label '1' with only one adjacent island must be connected to it via a 'single' bridge.

$$x_q^1 = 1, \quad \text{for } q \in S_a, \text{ if } |S_a| = 1 \text{ and } L_a = 1. \tag{6.3}$$

- Any island of label '2' with only one adjacent island must be connected to it via a 'double' bridge.

$$x_q^2 = 1, \quad \text{for } q \in S_a, \text{ if } |S_a| = 1 \text{ and } L_a = 2. \tag{6.4}$$

- Any island of label '2' with two adjacent islands, one of which has label '1', must have at least a 'single' bridge to the other island.

$$x_q^0 = 0, \quad \text{s.t. } S_a = \{q, r\}, \ L_r = 1, \ L_q \neq 1. \tag{6.5}$$

- Any island of label '3' with only two adjacent islands must have at least a 'single' bridge to all of them.

$$x_q^0 = 0, \quad \forall q \in S_a, \text{ if } |S_a| = 2 \text{ and } L_a = 3. \tag{6.6}$$

- Any island of label '4' with only two adjacent islands must have 'double' bridges to both of them.

$$x_q^2 = 1, \quad \forall q \in S_a, \text{ if } |S_a| = 2 \text{ and } L_a = 4. \tag{6.7}$$

- Any island of label '5' with only three adjacent islands must have at least a 'single' bridge to all of them.

$$x_q^0 = 0 \quad \forall q \in S_a, \text{ if } |S_a| = 3 \text{ and } L_a = 5. \tag{6.8}$$

- Any island of label '6' with only three adjacent islands must have 'double' bridges to all of them.

$$x_q^2 = 1, \quad \forall q \in S_a, \text{ if } |S_a| = 3 \text{ and } L_a = 6. \tag{6.9}$$

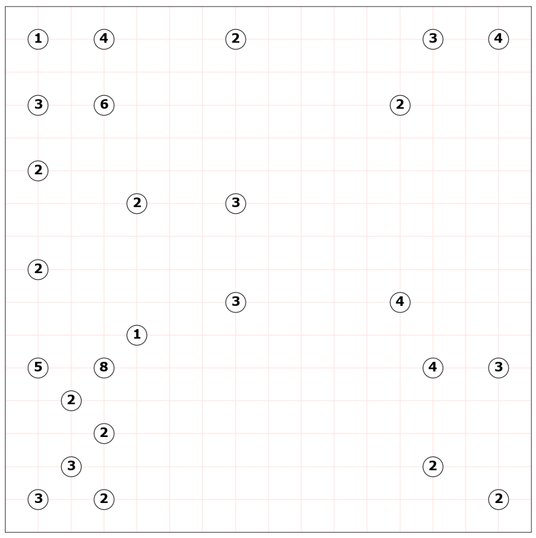
- Any island of label '7' must have at least a single bridge to all of its four adjacent bridges.

$$x_q^0 = 0 \quad \forall q \in S_a, \text{ if } L_a = 7. \tag{6.10}$$

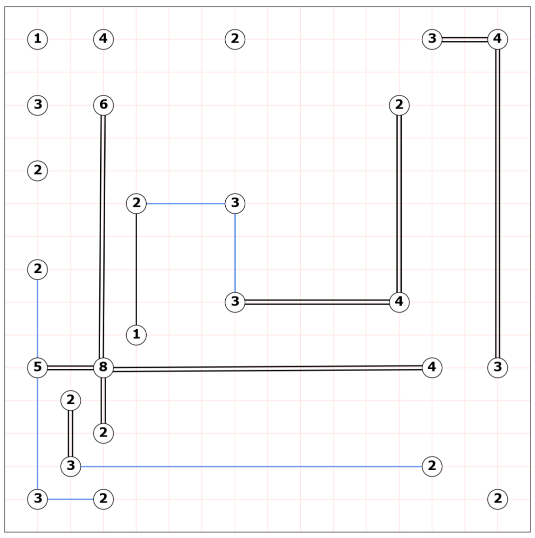
- Any island of label '8' must have 'double' bridges to all of its four adjacent bridges.

$$x_q^2 = 1, \quad \forall q \in S_a, \text{ if } L_a = 8. \tag{6.11}$$

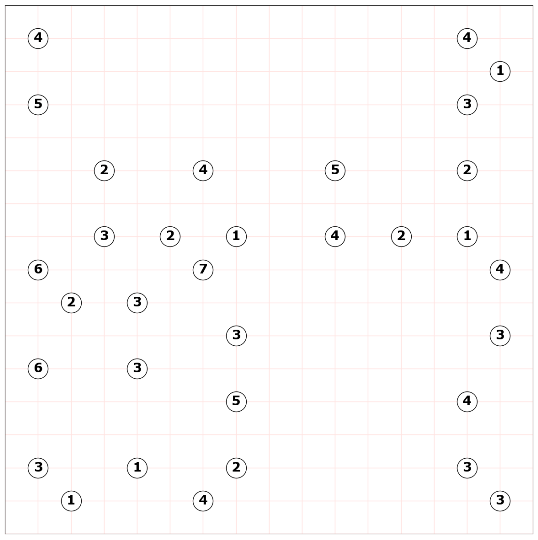
For the constraints which set $x_q^n = 1$ where $n \in \{0, 1, 2\}$, it follows that the other two related decision variables $x_q^m = 0, \ \forall m \in \{0, 1, 2\}, \ m \neq n$. However, we are not required to explicitly constrain these as Equation (6.1c) in our original model implies this, by ensuring only one of $x_q^n = 1$.



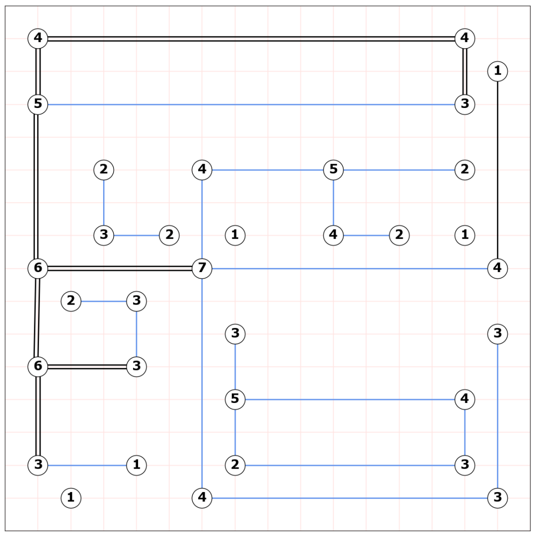
(a) Unsolved twenty-six island Hashi puzzle.



(b) Puzzle after initial deductions.



(c) Unsolved thirty-three island Hashi puzzle.



(d) Puzzle after initial deductions.

Figure 6.3: Top row shows a twenty-six island Hashi puzzle, and bottom row shows a thirty-three island Hashi puzzle. Each visualising the outcome of applying our initial HDC.

Figure 6.3 illustrates the application of the relevant constraints on two 15×15 Hashi boards, the first with twenty-six islands and the second with thirty-three. Here, bridges in black are included in the solution in this exact form, and those in blue are bridges that we know appear in some capacity, whether this is as a 'single' or 'double'. We note that here there are bridges in blue that we can verify must be singles by inspection, but this would be included in the next layer of deduction, and so we leave it for the optimization of the model. From these two boards we can see that the number of deduction constraints we make in this initial layer varies greatly, even when the size of the board is fixed.

6.3 Implementation

The model has been constructed in accordance with the constraints detailed above, but additional steps have been taken to reduce the complexity of the puzzle and thus reduce the computation time. These steps relate to both the specific coding choices made, and certain conventions that have been applied to eliminate redundant computations.

In our original model, Equation (6.1e) fixes a decision variable as part of ensuring the connectivity of our solution. This fixing of decision variables is also the main premise behind how we implement our HDC, Equations (6.2 - 6.11). For Equation (6.1e), and the HDC that fix a x_q^n decision variable to 1, we set the lower bound of the respective decision variable to 1, which due to their binary nature, fixes them to 1. Analogously, for the HDC that fix a x_q^n decision variable to 0 (those that have deduced at-least single bridges) we set the upper bound to 0. Fixing these as their bounds effectively removes them as decision variables, as they are constant throughout the optimization. This reduces the size of the problem, making it potentially faster to solve.

Figure 6.4 depicts the solution to the Hashi puzzle given in Figure 6.1.

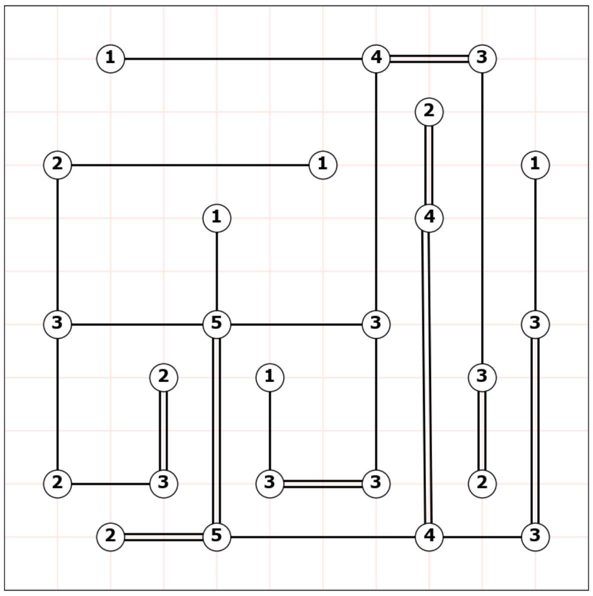


Figure 6.4: Solved twenty-five island Hashi puzzle, using the Hashi Model.

6.4 Model Comparison

To explore the efficiency and scalability of the model, as well as testing its validity, we performed the optimization on a range of Hashi boards. These boards were taken the Hashi website [24], and varied in size, from 5×5 to 30×30, as well as the number of islands they contained.

As outlined in Figure 6.3, the number of HDC varies depending on the specific puzzle instance. The number of these deductions that were made determined how many decision variables were fixed before optimization. Thus, in order to quantify the effect of these fixes, we compare the computations before and after their addition. Note that in order to accurately compare the variations of our model, we removed the 'Presolve' of the Gurobi optimizer, allowing us to observe more precise differences.

Table A.4 summarises the computation times for different puzzle instances, each defined uniquely by its grid size, number of islands and set of possible bridges. This allows us to assess which factors of the initial puzzle grid have the greatest effect on the computation time of our model. As shown in Table A.4 the computation time of both model variants were consistently quick, irrespective of the features of the initial board. It is evident that the size of the board is directly proportional to the number of islands and in turn the possible bridges. Thus, we can generalize from the observed differences between grid sizes that increasing the puzzle's scale results in higher computation time, even if these differences were minute. It is important to note that some instances deviated from the general trend. In these cases, we can attribute this variation to the randomness inherent in the Gurobi solver.

Furthermore, in the majority of the tested instances, the computation time of the model was reduced with the addition of the HDC. The cases in which this difference was most substantial occured in puzzles with a high number of deductions relative to the number of possible bridges, and thus our x_q^n decision variables. This confirms our intuition that removing infeasible solutions shrinks the problem. Overall this indicates that although the HDC are additional constraints their implementation as bounds does not increase the size of the problem, rather it reduces it. Thus our HDC model variation offers a more efficient formulation.

6.5 Conclusion

In this chapter we have successfully implemented an IP model for the game of Hashi. By adopting a thoughtful and deliberate approach to the initial modelling decisions we were able to model intricate graphical restrictions using minimal constraints. This allowed for the creation of an efficient model that can be easily applied to larger puzzle instances without an exponential increase in computation time.

Notably, by providing the model with the islands and possible bridges we were able to establish independence from a grid-based approach. This reduced the need for unnecessary decision variables, and highlights the fundamental difference between the formulation of Hashi and the game of Sudoku. Although we employed set partitioning constraints in both, the variables defined by predetermined sets and the incorporation of network flow constraints were required to tackle the added complexity of Hashi.

The exploration of potential improvements to our base model revealed a variation that utilised the human element of deduction required to solve the puzzle by hand. By shrinking the problem, the added constraints enhanced the model. This ultimately illustrated that more constraints do not necessarily result in slower execution.

Chapter 7

Zebra Puzzle

7.1 Introduction

The Zebra Puzzle, also known as Einstein's Puzzle, is a famous logic puzzle that provides players with a set of clues from which they must assign properties to houses. Unlike the other puzzles that have been modelled, the Zebra Puzzle relies purely on logical deduction without a predefined grid structure. Here is the first Zebra Puzzle that appeared in Life International in 1962 [26, p. 11]:

> Five men live in consecutive houses on a street. The houses are painted different colours. The men are of different nationalities, own different pets, have different favourite drinks and smoke different cigarette brands. Determine who owns a zebra given these clues:
>
> 1. The Englishman lives in the red house.
> 2. The Spaniard owns the dog.
> 3. Coffee is drunk in the green house.
> 4. The Ukrainian drinks tea.
> 5. The green house is immediately to the right of the ivory house.
> 6. The Old Gold smoker owns snails.
> 7. Kools are smoked in the yellow house.
> 8. Milk is drunk in the middle house.
> 9. The Norwegian lives in the first house.
> 10. The Chesterfield smoker lives next to the man with the fox.
> 11. Kools are smoked in the house next to the house with the horse.
> 12. The Lucky Strike smoker drinks orange juice.
> 13. The Japanese man smokes Parliaments.
> 14. The Norwegian lives next to the blue house.

Having created an IP formulation for Sudoku we will build upon some similarities to create an IP model for the Zebra Puzzle. When the Zebra Puzzle is viewed as a 5×5 table of clues, it is clear that both puzzles aim to complete vacant entries based on inferences gathered from these clues. The distinction is that the Zebra Puzzle presents the clues in the form of logical statements. In the event that these logical statements are sufficient and non-redundant, the unique solution to the puzzle still cannot be confirmed until we have considered all statements together. The Zebra Puzzle has a constraint not represented in the logical statements: for each individual house, there can only be one property of each attribute assigned to it and at the same time, each property can only be assigned to one house. The properties and attributes are shown in Table 7.1.

A similar characteristic was identified in the rules of Sudoku, wherein each position is restricted to a single number, and each number is limited to a single appearance in each row, column, and inner 3×3 box. Utilising the set partitioning constraints implemented in the Sudoku model we have established the fundamental constraints that dictate the Zebra Puzzle's implicit restriction.

In addition to considering the problem as an IP model, we also formulate it as a Constraint Programming (CP) model to investigate which strategy enhances computational efficiency. These insights will further our understanding of how to address the challenges arising in non-numeric logic puzzles.

7.2 The Models

In this section we will explain how to develop an IP and a CP model for the Zebra Puzzle. It is significant to note that the specific wording of the puzzle influences both models. Thus, we have divided the clues into distinct categories and outlined how to formulate their constraints using specific examples.

7.2.1 IP Model

For our IP model of the Zebra Puzzle, we do not have an explicit objective function. The goal is simply to find a feasible solution that satisfies all the constraints. Thus, our objective function is

$$\text{minimise } \mathbf{0}^t x.$$

Then, we define binary decision variables of the form $x_{ij}^{\text{attribute}}$, where $i, j \in \{1, ..., 5\}$ denote the house index and the properties respectively. The house indices i directly imply the location of the houses, which is another important consideration of the puzzle. Thus, when we refer to 'neighbours' this means that the indices of the houses are adjacent. For indexing purposes and clarity throughout our model the properties of each attribute are ordered alphabetically. These are listed in Table 7.1.

Attribute	**Properties**
Colour	Blue, Green, Ivory, Red, Yellow
Nationality	Englishman, Japanese, Norwegian, Spaniard, Ukranian
Drink	Coffee, Milk, Orange Juice, Tea, Water
Cigarette	Chesterfields, Kools, Lucky Strike, Parliaments, Old Gold
Pet	Dog, Fox, Horse, Snails, Zebra

Table 7.1: House attributes and their corresponding properties.

Hence, we formally define our decision variables as

$$x_{ij}^{\text{attribute}} = \begin{cases} 1, & \text{if house } i \text{ has property } j, \\ 0, & \text{otherwise,} \end{cases}$$

for attribute $\in$ {colour, nationality, drink, cigarette, pet}.

The implicit conditions outlined above motivate our first constraints

$$\sum_{j=1}^{5} x_{ij}^{\text{attribute}} = 1, \quad \forall i \in \{1,2,3,4,5\}, \tag{7.1a}$$

$$\sum_{i=1}^{5} x_{ij}^{\text{attribute}} = 1, \quad \forall j \in \{1,2,3,4,5\}, \tag{7.1b}$$

for all attribute $\in$ {colour, nationality, drink, cigarette, pet}. Equations (7.1a) ensure that every house i has exactly one property from each attribute. Equations (7.1b) require that for each attribute, every property is assigned to one house.

The clues given in the puzzle are logical statements that we can represent as linear constraints. Based on the categories described by Gregor et al. [9, p. 161], we have divided the constraints into three categories: Attribute Connection Constraints (ACC), Direct Location Constraints (DLC), and Indirect Location Constraints (ILC). To illustrate these we will review specific examples, from which all other statements can be derived analogously.

ACC

This type of constraint corresponds to the clues that provide a confirmed relationship between two properties. We note that this is not sufficient to deduce where these properties are placed as these clues do not mention location. From our example puzzle, Clue 1 states that "The Englishman lives in the red house", this is an example of an ACC. According to this clue, one of the houses must have the Englishman and red property simultaneously. This implies that the corresponding decision variables are 1, for one house, and 0 for the rest. So, for an unknown $i^* \in \{1, ..., 5\}$ and $\forall i \neq i^*$ we have

$$x_{i^* j=\text{red}}^{\text{colour}} = x_{i^* j=\text{englishman}}^{\text{nationality}} = 1 \quad \text{and} \quad x_{ij=\text{red}}^{\text{colour}} = x_{ij=\text{englishman}}^{\text{nationality}} = 0.$$

Thus we encompass these cases in the following constraint

$$x_{i\ j=\text{red}}^{\text{colour}} - x_{i\ j=\text{englishman}}^{\text{nationality}} = 0, \quad \forall i \in \{1, ..., 5\}.$$

DLC

This category represents the clues that place a given property in an exact house index i. Clue 8 states "Milk is drunk in the middle house". Here "middle house" refers to the house index $i = 3$, as such the decision variable $x_{i=3\ j=\text{milk}}^{\text{drink}} = 1$.

ILC

Keywords such as “next to”, “immediately right of”, and “immediately left of” often appear in the Zebra Puzzle’s logical statements. As such, this category of constraint describes clues that provide location information, without being able to specify specific indices. One example is Clue 14, stating “The Norweigian lives next to the blue house”. This constraint has another level of complexity as it must consider the concept of how house locations relate to each other. We formulate this logical statement as follows,

$$\sum_{i=2}^{5} x_{ij=\text{blue}}^{\text{colour}} \cdot x_{i-1\ j=\text{norwegian}}^{\text{nationality}} + \sum_{i=2}^{5} x_{ij=\text{norwegian}}^{\text{nationality}} \cdot x_{i-1\ j=\text{blue}}^{\text{colour}} = 1.$$

This formula takes into account the possibility of two situations. The first sum accounts for the Norwegian living immediately to the left of the blue house, whilst the second considers them living immediately to the right. Making our summation equal to one ensures that only one of these cases occurs. Note that the summations are indexed from two as we consider a house being immediately to the left of another as having an index one less.

7.2.2 CP Model

CP is a technique widely used in logic puzzles, as constraints are declared allowing for easier interpretation of complex logical statements. CP models the problem as a set of categories, variables, domains, and constraints, from which the solver systematically eliminates infeasible options until a solution is found [12, p. 1]. In this section we will demonstrate how to model the Zebra Puzzle in this way based on the CP model created by Gregor et al. [9, pp. 161-162].

The domains of the variables in our CP model are pre-defined sets of properties. Each house in the Zebra Puzzle is a predicate represented by its set of variables, the attributes. The variables and their domains are given in Table 7.2.

Variable	**Domain**
House[i].v_{Color}	{Blue, Green, Ivory, Red, Yellow}
House[i].$v_{\text{Nationality}}$	{Englishman, Japanese, Norwegian, Spaniard, Ukrainian}
House[i].v_{Drink}	{Coffee, Milk, Orange Juice, Tea, Water}
House[i].$v_{\text{Cigarette}}$	{Chesterfields, Kools, Lucky Strike, Parliaments, Old Gold}
House[i].v_{Pet}	{Dog, Fox, Horse, Snails, Zebra}

Table 7.2: Variable categories and their corresponding domains.

To begin with, we have a unique constraint that each property appears exactly once across the five houses,

$$\sum_{i=1}^{5} x[i,v] = 1, \quad \forall v \in V.$$

Here, V is the union of all variable domains. Then, $x[i,v]$ is a binary variable, where $x[i,v] = 1$ if house i is assigned the value v, and $x[i,v] = 0$ otherwise.

Similar to our IP model, we can classify clues into three types: ACC, DLC, and ILC. Based on the definitions introduced previously, we can formulate CP examples for each type of constraint. Table 7.3 presents how the three types are formulated as CP constraints for modeling and execution:

Type	**Clue**	**CP Constraint**
ACC	"The Englishman lives in the red house"	House$[i].v_{\text{nationality}}$ = Englishman $\Rightarrow$ House$[i].v_{\text{color}}$ = Red
DLC	"The Norwegian lives in the first house"	House$[1].v_{\text{nationality}}$ = Norwegian
ILC	"The Norwegian lives next to the blue house"	|position(Norwegian) − position(Blue House)| = 1

Table 7.3: The CP formulation of clues of different types.

7.3 Implementation

In addition to using Gurobi for our IP model, we have used the Prolog solver for our CP model. Both approaches yield the same result as displayed in Table 7.4 and match the verified solution given in Vassberg et al. [26, p. 22]. This confirms the accuracy of our models, when applied to this specific instance of the Zebra Puzzle. We have presented the logic behind the construction of the clues as constraints, however a limitation inherent to the puzzle itself is the need for a new model to handle the specific word choices. In saying this, the puzzles follows a standard format for which our methods can be applied analogously.

House	**Color**	**Nationality**	**Drink**	**Cigarette**	**Pet**
1	Yellow	Norwegian	Water	Kools	Fox
2	Blue	Ukrainian	Tea	Chesterfields	Horse
3	Red	Englishman	Milk	Old Gold	Snails
4	Ivory	Spaniard	Orange Juice	Lucky Strike	Dog
5	Green	Japanese	Coffee	Parliaments	Zebra

Table 7.4: Solution for the given Zebra Puzzle, found by both the CP and IP models.

7.3.1 CP Heuristics

Tracing the execution of the computational process of the CP model we can see the execution details of the trace of problem solving. Figure 7.1 shows the search path extends from a series of potential matchings, gradually exploring possible house assignments. The path in red is the pruned branch due to a contradiction of a specific clue. From the trace, we find that if the order of initial constraints input into the solver is not chosen well, it can lead to the investigation of unnecessary branches. Many of these branches must extend to deeper levels before being ruled out, resulting in additional computations. Thus, Gregor et al. [9, pp. 163-164] propose three heuristic strategies to improve computational efficiency.

Heuristic 1: Prioritising the DLC

A more effective approach involves prioritizing the DLC, as it explicitly defines the exact position of a house within the list. These are the most explicit constraints

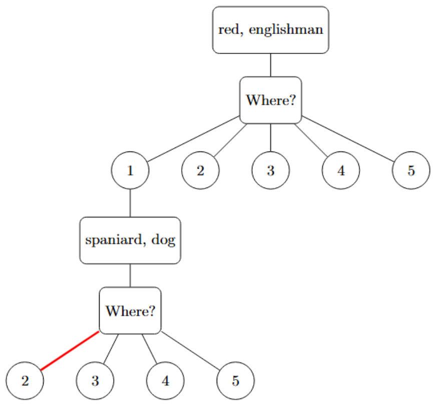


Figure 7.1: Top segment of the search tree.

within the puzzle as they fix specific properties to a given house index. This allows us to eliminate the greatest number of branches relating to this property, as they will all be trivial contradictions.

Heuristic 2: Deferring the ILC

As defined previously, the ILC use terms such as "next to", "immediately right of" and "immediately left of". Unlike the other constraints, the ILC allow for more potential solutions that meet the requirements. This means we are not able to cut off as many branches, therefore deferring such constraints can be beneficial.

Heuristic 3: Prioritising Mutually Exclusive Elements

Mutually exclusive elements (MEE) are properties belonging to the same attribute. For example, in the puzzle mentioned in this section, the MEE of 'zebra' are 'horse', 'snail', 'dog', and 'fox'. Mutually exclusive constraints (MEC) for a certain property include their MEE. For the zebra, a MEC is "The Spaniard owns the dog". In the puzzle, we do not specify what house the zebra lives in. Thus, if we do not prioritise MEC, we will consider the possibility that the zebra is in every house. Based on the clue listed above we know that the zebra is not in the Spaniard's house, so it is unnecessary to explore these branches. Therefore, if we prioritize all MEC we will improve our overall computational efficiency.

7.4 Model Comparison

We will compare the types of optimization discussed above for two different criteria. The first is the time it takes to solve the models and the second is the compatibility of the optimization methods for this puzzle.

7.4.1 Computational Efficiency Comparison

Comparing the results of Gurobi (IP) and Prolog (CP) for the puzzle given above, we find that both solvers exhibit excellent computational times of less than 0.1

seconds, as displayed in Table 7.5. However, Prolog still slightly outperforms Gurobi's computational time regardless of heuristics. It is reasonable to speculate that Gurobi takes relatively longer because it cannot leverage the logical properties of the puzzle in the way that Prolog can using CP.

If we look at the different implementations of our CP model, we find that both the number of inferences and the computation time decrease significantly with the change in the ordering of computations due to the addition of the heuristics. In particular, the number of inferences decreases drastically from 21,612 to 663. Overall, our CP model consistently outperforms the IP model solved using Gurobi, with this advantage only furthered by the addition of applicable heuristics.

Method	Number of Inferences	Computation Time (s)
Gurobi (IP)	-	0.050
Prolog (CP) - Original	21,612	0.005
Prolog (CP) - Heuristics 1 & 2	5,174	0.001
Prolog (CP) - Heuristics 1, 2 & 3	663	<0.001

Table 7.5: Comparison of Gurobi (IP) and Prolog (CP) Performance

7.4.2 Strategy Comparison

The IP model determines the solution of the Zebra Puzzle through linear constraints made entirely of binary variables, which is then solved using standard optimization techniques. These techniques, implemented by Gurobi, include Branch-and-Bound. The Zebra Puzzle is an objectively small problem although the existence of constraints like the ILI make the model more complex. They require advanced constraint formulations to be transformed from logical statements into linear constraints. In saying this, IP is still an efficient solution method.

In contrast, CP by design allows constraints to be expressed directly in terms of logical rules. Due to the logical statements of the Zebra Puzzle, CP is easier to understand and implement. In addition, three relevant heuristics were successfully implemented to improve the performance of the model in the Prolog solver. Thus, CP is more efficient than IP when applied to the Zebra Puzzle as a result of its suitability for problems involving complex logical constraints.

7.5 Conclusion

We compared our IP and CP models for the Zebra Puzzle to understand the respective optimization method's compatibility with language based logic problems. Inspired by the use of set partitioning constraints for Sudoku we were able to experiment with the typically unexplored method of solving language puzzles with IP. Having accomplished this we compared our approach to the standard method of modelling logical statements, CP. As expected, CP was more efficient.

Another key finding is that different modelling approaches in CP affect the computational performance. We can confirm that including heuristics that can logically eliminate unnecessary examinations of dead end branches is crucial to lowering computational time.

Chapter 8

Bilevel Programming for Sudoku

8.1 Introduction

Throughout this report the focal point has been *solving* deductive puzzles. In this chapter we will shift our focus onto the the creation of the puzzles themselves. It has been proven that in order for a Sudoku to be valid, to have a unique solution, it must contain at least seventeen clues [18, p. 190]. However, not every solution has this property. This has inspired the Minimum Sudoku Clue Problem; what is the minimum number of clues for a puzzle with a specified unique solution? Recently, Tjusila et al. developed a model to solve the Minimum Sudoku Clue Problem using BP [25, pp. 2-4]. We will be reproducing their BP model in hopes of understanding how their formulation can be extended to other puzzles.

8.2 The BP Model

For this BP model the ULP is trying to find which cells of the given Sudoku solution need to be passed to the LLP in order for the solution to be unique. The LLP then checks if there are any other possible solutions given the target grid with entries missing. The ULP and LLP essentially work against eachother. The ULP wants the Sudoku to be valid, but the LLP's objective is to determine that the solution is not unique.

We begin by reviewing the notation used to create an IP model for Sudoku. Let (i, j) denote the cells of an $n \times n$ grid. Let $m = \sqrt{n}$ represent the dimension of the inner boxes of the puzzle. Let $D = \{1, 2, ..., n\}$ and $E = \{1, 2, ..., m\}$. Thus, the decision variables for both problems are

$$x_{ijk} = \begin{cases} 1, & \text{if cell } (i, j) \text{ contains the integer } k, \text{ for } i, j, k \in D, \\ 0, & \text{otherwise,} \end{cases}$$

and the ULP also has the decision variables

$$y_{ij} = \begin{cases} 1, & \text{if entry in cell } (i, j) \text{ is passed to the LLP,} \\ 0, & \text{otherwise.} \end{cases}$$

We also use $S(y)$ to represent the set of optimum solutions of the LLP. Let G be the complete Sudoku grid that we are trying to solve the Minimum Sudoku Clue Problem for. Note that G_{ij} is the entry in cell (i, j). We use z to denote the penalty accumulated by the LLP if the Sudoku is valid.

The ULP model is

$$\text{minimise} \sum_{i=1}^{n}\sum_{j=1}^{n} y_{ij} \tag{8.1a}$$

$$\text{subject to } z = 1, \tag{8.1b}$$

$$y_{ij} \in \{0,1\}\ \forall i, j \in D, \tag{8.1c}$$

$$(x, z) \in S(y). \tag{8.1d}$$

The objective function of the ULP, Equation (8.1a), minimises the number of starting clues required for the Sudoku to be valid. Equation (8.1b) requires that the Sudoku has a unique solution. This is determined by the LLP.

The LLP model is

$$\text{minimise } z \tag{8.2a}$$

$$\text{subject to } \sum_{i=1}^{n} x_{ijk} = 1, \qquad \forall j, k \in D \tag{8.2b}$$

$$\sum_{j=1}^{n} x_{ijk} = 1, \qquad \forall i, k \in D \tag{8.2c}$$

$$\sum_{k=1}^{n} x_{ijk} = 1, \qquad \forall i, j \in D \tag{8.2d}$$

$$\sum_{j=mq-m+1}^{mq} \sum_{i=mp-m+1}^{mp} x_{ijk} = 1, \quad \forall k \in D,\ \forall p, q \in E \tag{8.2e}$$

$$x_{ijG_{ij}} \geq y_{ij}, \qquad \forall i, j \in D, \tag{8.2f}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijG_{ij}} - z \leq n^2 - 1, \tag{8.2g}$$

$$z \in \{0,1\}, \tag{8.2h}$$

$$x_{ijk}\ \forall i, j, k \in D. \tag{8.2i}$$

In the LLP the objective function, Equation (8.2a), minimises the penalty that occurs if the Sudoku is valid. Equations (8.2b)-(8.2e) are the constraints that enforce the rules of Sudoku. These constraints were also used in the IP Sudoku model in Chapter 3. Equation (8.2f) requires that any solution found by the LLP must include the clues passed from the ULP. Then, Equation (8.2g) is the constraint that makes the solution found by the LLP different from the given grid G. This follows from the fact that the LLP is minimising the binary variable z, thus when z is zero the solution found by the LLP must have at least one different cell from the target solution G. If z is one, they must be equal.

8.3 Implementation

We successfully implemented the BP model using MibS [22] in conjunction with BilevelJuMP [8] on Julia. These solvers had to be used because Gurobi does not support solving BP problems. An important consideration that must be made when implementing all BP models is that the solver must know which level each variable and constraint is a part of. Thus the model was implemented using the constraints and variables described above while additionally specifying whether each component is part of the ULP or LLP. Any equality constraints also had to be adjusted so that they were equivalent inequalities to work in MibS. For example, $z = 1$ would be the same as $z \geq 1$ as it is a binary variable. Figure 8.1 displays an example of a Minimum Sudoku Clue Problem solved using this BP model.

8	7	6	9	1	2	5	3	4
9	5	4	3	8	7	6	1	2
2	1	3	6	4	5	7	8	9
7	3	8	4	5	6	2	9	1
5	6	1	8	2	9	3	4	7
4	2	9	7	3	1	8	6	5
6	4	2	1	7	8	9	5	3
3	9	5	2	6	4	1	7	8
1	8	7	5	9	3	4	2	6

(a) Target unique solution passed to the ULP.

							1	2
		3		4	5			
	6					3		7
4	2				1			
			1		8			
			2	6				
		7				4		

(b) Minimum number of clues required to achieve this unique solution.

Figure 8.1: The solution grid given to the BP model and its solution for the Minimum Sudoku Clue Problem.

Chapter 9

Conclusion

In this report we have analysed a wide range of optimization techniques which can be utilised to solve logic puzzles. We began our exploration using the famous puzzle Sudoku to demonstrate the simpler concepts of Integer Programming from which we established a foundation for more complex examples. While these puzzles are all considered deductive, they demonstrated distinct challenges. As such we constrained spatial, logic and numeric restrictions. From employing a variety of techniques to deal with these constraints, we highlighted the extensive set of possibilities for modelling the same concepts. To this end, we have identified some considerations that should be at the forefront when aiming to create a superior model.

A notable finding was that the strategic nature of solving puzzles could be exploited to reveal characteristics that simplified our models. Due to a large number of constraints and variables often causing longer computation times we found that taking a thoughtful and simplistic approach to our models led to increased efficiency. This was a recurring theme in the puzzle formulations we designed in Chapters 4-7. In the Battleship Solitaire Puzzle, we were able to significantly reduce the amount of constraints required when we leveraged the possible placements of whole ships the way a player would. Then, the observation that the solution to the Circuit Board Game was a spanning tree inspired a model that was more efficient for large instances compared to the more prominent Minimum Cost Network Flow formulation. We devised additional constraints for the game of Hashi that included bridges in the solution based on a thorough understanding of the game. Here, the ability to fix some variables reduced the number of possibilities that needed to be explored. Finally, for the Zebra Puzzle, the compatibility of word based logic with Constraint Programming meant that this more intuitive method was faster than Integer Programming. Overall, these considerations emphasised the importance of the initial decisions made to set up a model and the subsequent influence this has on its complexity.

The replication of the Bilevel formulation of Sudoku demonstrated the theory behind performing BP to create models that can reverse engineer a unique starting puzzle. This opens up the potential of nested optimization to initialise other games. This motivates a framework for the further work of this project. For example, the game of Hashi could be made more difficult by considering the density of the grid. In particular, creating an instance with a specific amount of islands for a fixed grid size enables us to select higher density cases. As a result, there are more initial decisions which makes reaching the unique solution more challenging. Finding higher density puzzles whilst also ensuring the solution is unique is compatible with the follower-leader format of Bilevel Programming. Similar lines of research could be followed for the other puzzles, prompting us to explore the potential application of these methods for more complex logical scenarios.

Not only did we establish strategies for formulating these models, validation was accomplished by implementing them using optimization software. It now stands that what is challenging for even mathematicians is clearly not challenging for computers.

Bibliography

[1] Andrew Bartlett, Timothy P Chartier, Amy N Langville, and Timothy D Rankin. An integer programming model for the Sudoku problem. *Journal of Online Mathematics and its Applications*, 8(1):1798, 2008.

[2] Seymour S Block and Santiago A Tavares. History of Sudoku. In *Before Sudoku: The World of Magic Squares*. Oxford University Press, 2009.

[3] Der-San Chen, Robert G Batson, and Yu Dang. *Applied Integer Programming: Modeling and Solution*. John Wiley Sons, 2010.

[4] Michele Conforti, Gerard Cornuejols, and Giacomo. Zambelli. *Integer Programming*. Graduate Texts in Mathematics, 271. Springer International Publishing, 2014.

[5] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a Large-Scale Traveling-Salesman Problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

[6] Stephan Dempe. *Foundations of Bilevel Programming*. Nonconvex Optimization and its Applications. Kluwer Academic Publishers, 2002.

[7] Stephan Dempe and Alain Zemkoho. *Bilevel Optimization : Advances and Next Challenges*. Optimization and its Applications. Springer, first edition, 2020.

[8] Joaquim Dias Garcia, Guilherme Bodin, and Alexandre Street. BilevelJuMP.jl: Modeling and solving bilevel optimization in Julia. *arXiv preprint arXiv:2205.02307*, 2022.

[9] Michal Gregor, Katarína Zábovská, and Vladimír Smataník. The Zebra Puzzle and Getting to Know your Tools. In *19th International Conference on Intelligent Engineering Systems*, pages 159–164, 2015.

[10] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024.

[11] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring Network Structure, Dynamics, and Function using NetworkX. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, 2008.

[12] William Leler. *Constraint Programming Languages: Their Specification and Generation*. Addison-Wesley Series in Computer Science. Addison-Wesley Publishing Company, 1988.

[13] Huw Lloyd, Matthew Crossley, Mark Sinclair, and Martyn Amos. J-pop: Japanese puzzles as optimization problems. *IEEE transactions on games*, 14(3):391–402, 2022.

[14] Telegraph Media Group Ltd. *The Telegraph Big Book of Suduko 1*. Octopus Publishing Group, 2018.

[15] Cheng-Wei Lu. Circuitboard scenario. https://miro.neos-server.org/app/circuitboard2. Accessed 17-10-2024.

[16] Reza Firsandaya Malik, Rusdi Efendi, and Eriska Amrina Pratiwi. Solving Hashiwokakero Puzzle Game with Hashi Solving Techniques and Depth First Search. *Bulletin of Electrical Engineering and Informatics*, 1(1):61–68, 2012.

[17] Sonja Mars. What is the difference between user cuts and lazy constraints? https://support.gurobi.com/hc/en-us/articles/360025804471-What-is-the-difference-between-user-cuts-and-lazy-constraints. Accessed 28-02-2025.

[18] Gary McGuire, Bastian Tugemann, and Gilles Civario. There Is No 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem via Hitting Set Enumeration. *Experimental Mathematics*, 23(2):190–217, 2014.

[19] W.J.M. Meuffels and Dick den Hertog. Solving the Battleship puzzle as an integer programming problem. *INFORMS Transactions on Education*, 10(3):156–162, 2010.

[20] Gurobi Optimization. Python Callbacks Reference - Gurobi Optimizer Reference Manual. https://docs.gurobi.com/projects/optimizer/en/current/reference/python/callback.html. Accessed 28-02-2025.

[21] Leonardo Taccari. Integer Programming Formulations for the Elementary Shortest Path Problem. *Eur. J. Oper. Res.*, 252(1):122–130, 2016.

[22] Sahar Tahernejad, Ted K. Ralphs, and Scott T. DeNegre. A branch-and-cut algorithm for mixed integer bilevel linear optimization problems and its implementation. *Mathematical programming computation*, 12(4):529–568, 2020.

[23] Puzzle Team. Battleships. https://www.puzzle-battleships.com/. Accessed 13-03-2025.

[24] Puzzle Team. Hashi. https://www.puzzle-bridges.com/. Accessed 9-02-2025.

[25] Gennesaret Tjusila, Mathieu Besançon, Mark Turner, and Thorsten Koch. How many clues to give? A bilevel formulation for the minimum Sudoku clue problem. *Oper. Res. Lett.*, 54:107105, 2024.

[26] Dylan Vassberg and J. Vassberg. Is Einstein's Puzzle Over-Specified? *75th Symposium: 21st Century Challenges in Computational Engineering Science*, 2009.

[27] Robin Wilson. *Introduction to Graph Theory.* Pearson Education Limited, 2010.

[28] Laurence A. Wolsey. *Integer Programming.* John Wiley Sons, Inc., second edition, 2020.

Appendix A

Appendix

A.1 Repository and Implementation

The files containing the code for implementing our models are available on our GitHub page.

A.2 Tables for Chapter 5

CM1					
$\lvert V\rvert$	$\lvert E\rvert$	Best Upper Bound	Best Lower Bound	Optimality Gap (%)	Time (s)
4	3	3	3	0	0.000
4	3	3	3	0	0.000
6	7	5	5	0	0.016
6	6	5	5	0	0.000
8	9	7	7	0	0.000
10	12	9	9	0	0.006
12	16	11	11	0	0.000
12	15	11	11	0	0.016
14	20	13	13	0	0.000
14	20	13	13	0	0.016
20	30	19	19	0	0.000
20	30	19	19	0	0.000
22	34	21	21	0	0.000
22	34	21	21	0	0.000
22	34	21	21	0	0.006
30	49	29	29	0	0.032
30	48	29	29	0	0.031
32	51	31	31	0	0.041
34	56	33	33	0	0.047
34	53	33	33	0	0.031
42	71	41	41	0	0.071
42	67	41	41	0	0.095
44	70	43	43	0	0.104
46	78	45	45	0	0.098
48	81	47	47	0	0.158
58	94	57	57	0	0.379
58	96	57	57	0	0.251
62	108	61	61	0	0.419
62	108	61	61	0	0.648
62	105	61	61	0	0.472
76	127	75	75	0	1.631
76	134	75	75	0	1.489
78	138	77	77	0	1.040
78	134	77	77	0	3.733
78	135	77	77	0	0.917
94	168	93	93	0	2.199
94	164	93	93	0	4.164
96	171	95	95	0	4.342
98	176	97	97	0	10.957
98	173	97	97	0	4.750

Table A.1: Computation time for CM1 on select puzzles from 3×3 grids to 10×10 grids.

CM2					
$\lvert V\rvert$	$\lvert E\rvert$	Best Upper Bound	Best Lower Bound	Optimality Gap (%)	Time (s)
4	3	3	3	0	0.000
4	3	3	3	0	0.000
6	7	5	5	0	0.000
6	6	5	5	0	0.000
8	9	7	7	0	0.000
10	12	9	9	0	0.016
12	16	11	11	0	0.016
12	15	11	11	0	0.000
14	20	13	13	0	0.016
14	20	13	13	0	0.000
20	30	19	19	0	0.047
20	30	19	19	0	0.016
22	34	21	21	0	0.080
22	34	21	21	0	0.126
22	34	21	21	0	0.132
30	49	29	29	0	0.235
30	48	29	29	0	0.189
32	51	31	31	0	0.251
34	56	33	33	0	0.691
34	53	33	33	0	0.142
42	71	41	41	0	14.106
42	67	41	41	0	1.058
44	70	43	43	0	0.646
46	78	45	45	0	54.991
48	81	47	47	0	61.646
58	94	57	57	0	9.072
58	96	57	57	0	359.293
62	108	61	56	8.20	1800
62	108	61	56	6.56	1800
62	105	61	58	4.92	1800
76	127	75	69	6.67	1800
76	134	75	66	12.00	1800
78	138	77	67	12.99	1800
78	134	77	69	10.39	1800
78	135	77	79	11.69	1800
94	168	93	77	16.13	1800
94	164	93	78	16.13	1800
96	171	95	80	15.79	1800
98	176	97	80	16.49	1800
98	173	97	81	16.49	1800

Table A.2: Computation time for CM2 on select puzzles from 3×3 grids to 10×10 grids.

$\lvert V\rvert$	$\lvert E\rvert$	z^*_{CM1}	z^*_{CM2}	z^*_{CM2}/z^*_{CM1}
6	7	5.000	3.020	0.604
14	20	13.000	8.000	0.615
22	34	21.000	12.000	0.571
34	56	33.000	17.020	0.516
42	71	41.000	21.010	0.512
62	108	61.000	31.010	0.508
76	134	75.000	39.000	0.520
98	176	97.000	49.480	0.510

Table A.3: Objective values for the LR of CM1 and CM2 for puzzles of increasing vertex quantities and the ratio between them. The objective values are denoted z^*_{CM1} and z^*_{CM2} for the LR of CM1 and CM2 respectively.

A.3 Tables for Chapter 6

Hashi Model					
n	k	$\|B\|$	Number of HDC	Computation Time (s)	Computation Time with HDC (s)
5	8	9	7	0.00244	0.00064
5	7	7	9	0.00261	0.00206
5	11	14	20	0.01229	0.00203
7	10	11	9	0.00259	0.00432
7	12	14	21	0.01125	0.00237
7	13	15	15	0.00842	0.00237
7	14	17	11	0.00913	0.00736
7	14	17	24	0.00388	0.00337
10	20	24	21	0.01082	0.00995
10	22	27	25	0.01315	0.01003
10	23	30	21	0.01466	0.03081
10	25	34	26	0.01138	0.00949
10	25	32	15	0.01221	0.00930
15	26	32	36	0.00648	0.00358
15	29	35	41	0.00748	0.00959
15	33	43	47	0.05117	0.00592
15	33	42	54	0.01518	0.00554
25	63	84	62	0.02206	0.00852
25	83	119	70	0.06532	0.01806
30	127	197	93	0.20876	0.08316

Table A.4: Computation times for the Hashi Model before and after the addition of HDC, from puzzle instances ranging from 5×5 to 30×30.