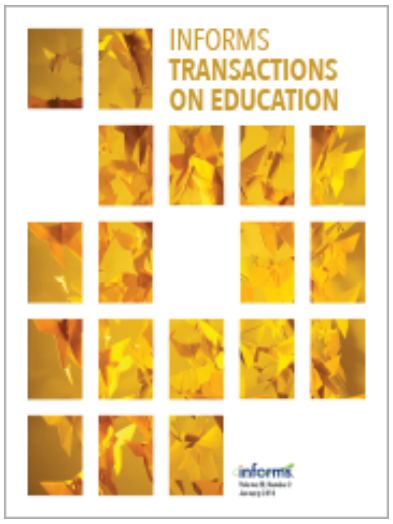
This article was downloaded by: [94.61.124.44] On: 22 September 2020, At: 06:37
Publisher: Institute for Operations Research and the Management Sciences (INFORMS)
INFORMS is located in Maryland, USA



INFORMS Transactions on Education

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Puzzle—More Logic Puzzle Apps Solved by Mathematical Programming

Sönke Hartmann

To cite this article:
Sönke Hartmann (2019) Puzzle—More Logic Puzzle Apps Solved by Mathematical Programming. INFORMS Transactions on Education 20(1):49-55. https://doi.org/10.1287/ited.2019.0212

Full terms and conditions of use: https://pubsonline.informs.org/Publications/Librarians-Portal/PubsOnLine-Terms-and-Conditions

This article may be used only for the purposes of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval, unless otherwise noted. For more information, contact permissions@informs.org.

The Publisher does not warrant or guarantee the article's accuracy, completeness, merchantability, fitness for a particular purpose, or non-infringement. Descriptions of, or references to, products or publications, or inclusion of an advertisement in this article, neither constitutes nor implies a guarantee, endorsement, or support of claims made of that product, publication, or service.

Copyright © 2019, The Author(s)

Please scroll down for article—it is on subsequent pages

With 12,500 members from nearly 90 countries, INFORMS is the largest international association of operations research (O.R.) and analytics professionals and students. INFORMS provides unique networking and learning opportunities for individual professionals, and organizations of all types and sizes, to better understand and use O.R. and analytics tools and methods to transform strategic visions and achieve better outcomes.
For more information on INFORMS, its publications, membership, or meetings visit http://www.informs.org

http://pubsonline.informs.org/journal/ited

INFORMS TRANSACTIONS ON EDUCATION
Vol. 20, No. 1, September 2019, pp. 49–55
ISSN 1532-0545 (online)

Puzzle

More Logic Puzzle Apps Solved by Mathematical Programming

Sönke Hartmann[a]

[a] HSBA Hamburg School of Business Administration, D-20457 Hamburg, Germany
Contact: soenke.hartmann@hsba.de, https://orcid.org/0000-0001-6687-1480 (SH)

Received: February 5, 2019
Revised: March 17, 2019; April 1, 2019
Accepted: April 2, 2019
Published Online in Articles in Advance: August 30, 2019

https://doi.org/10.1287/ited.2019.0212

Copyright: © 2019 The Author(s)

Abstract. This paper considers six logic puzzles (i.e., single-player games) that are available as smartphone apps—namely Starbattle, V3ck, Monodot, Knight Moves, Circuit Scramble, and Binary Sudoku. For each puzzle, an integer linear programming formulation is presented (a MathProg implementation is available as well). The purpose is to provide interesting examples and exercises for teaching mathematical programming in operations research and management science lectures. The puzzles lead to a broad variety of exercises, ranging from graph models to specific constraint types such as logical and inequality constraints.

Open Access Statement: This work is licensed under a Creative Commons Attribution 4.0 International License. You are free to copy, distribute, transmit and adapt this work, but you must attribute this work as "*INFORMS Transactions on Education*. Copyright © 2019 The Author(s). https://doi.org/10.1287/ited.2019.0212, used under a Creative Commons Attribution License: https://creativecommons.org/licenses/by/4.0/."

Supplemental Material: The implementation of the models is available at https://www.informs.org/Publications/Subscribe/Access-Restricted-Materials.

Keywords: puzzle • teaching modeling

1. Introduction

Logic puzzles are very popular among computer scientists and operations researchers. Consider, for example, Sudoku, which is one of the most liked puzzles. Mathematical programming formulations for Sudoku and related implementations have been published in several scientific journals and books, such as Chlond (2005), Koch (2006), Weiss and Rasmussen (2007), Rasmussen and Weiss (2007), Bartlett et al. (2008), and Oki (2012, section 9.1); they also can be found in various blogs. Further puzzles for which mathematical models have been proposed include, among others, the n queens problem (Letavec and Ruggiero 2002), other chessboard placement puzzles (Chlond and Toase 2002, Chlond 2010), Einstein's riddle (Yeomans 2003), logic grid puzzles (Chlond 2014), and the popular smartphone puzzle apps Flow Free (also called Numberlink) and Infinity Loop (Hartmann 2018).

One particular motivation for studying logic puzzles in operations research (OR) and management science is that they can be used when teaching mathematical modeling. It might be interesting for students to develop integer programs to solve puzzles. In recent years, a large number of logic puzzles appeared as smartphone apps, and considering the numbers of downloads reported in the app stores, many of the logic puzzle apps are immensely popular (Hartmann 2018). It may be particularly motivating for students to develop mathematical models for logic puzzle apps they can easily download and play on their devices.

This paper presents six logic puzzles that are available as smartphone apps and develops mathematical programming formulations to solve them. As in Hartmann (2018), the following criteria are used for selecting the puzzles. The first goal is to obtain models with different levels of difficulty, with a particular focus on models that are not too difficult (to use them in introductory lectures). The second goal is to include models with a broad range of constraint types. The third goal is to motivate students to work with the apps. Thus puzzles with rules that are simple and easy to learn are selected, and apps that are available for free are chosen (all apps discussed here are available for Android for free, and some of them are available for iOS as well).

The remainder of this paper is organized as follows. In each of the next six sections, a puzzle is introduced, and a mathematical model is developed to solve it. Section 2 presents the app Starbattle, which leads to a rather easy exercise. Then Sections 3 and 4 discuss the apps V3ck and Monodot, respectively, which allow for exercises related to graphs. The next three sections discuss puzzles that are related to specific constraint types. Section 5 deals with puzzles based on the moves of knights on chessboards; this can be used when teaching models for the traveling salesman problem and the related subtour elimination

constraints. The app Circuit Scramble presented in Section 6 allows for the study of the formulation of logical constraints (AND, OR, XOR, and NOT) in a linear model. In Section 7, a puzzle often called Binary Sodoku is considered. One of its rules leads to inequality constraints ($\neq$), which will be covered by the mathematical model. Finally, Section 8 draws some conclusions. Implementations of the models in MathProg (Makhorin 2010) are available online as supplemental material.

2. Starbattle

The app Starbattle is based on a grid of $n \times n$ cells. The grid is partitioned into n sets of connected cells; these sets are called *islands* (see Figure 1). The n stars must be placed in the grid such that each row, each column, and each island contains exactly one star. Moreover, all eight cells surrounding a cell with a star must remain empty. In a variant of the puzzle, $2n$ stars are placed, and each row, column, and island must contain two stars instead of one.

Let S be the number of stars that must be placed in each row, column, and island (we have either $S = 1$ or $S = 2$). The set of all cells in the grid is $C = \{(i,j) \mid i,j = 1,\cdots,n\}$. Next, let $A_k \subset C$ be the kth island, $k = 1,\cdots,n$. We assume that $A_k \cap A_{k'} = \emptyset$ for all $k \neq k'$, and $A_1 \cup A_2 \cup \cdots \cup A_n = C$.

The binary decision variables x_{ij} reflect the placement of the stars:

$$x_{ij} = \begin{cases} 1, & \text{if a star is placed in cell } (i,j), \\ 0, & \text{otherwise.} \end{cases}$$

Now we can give the mathematical model for Starbattle:

$$\sum_{j=1}^{n} x_{ij} = S, \qquad i = 1,\ldots,n, \qquad (1)$$

$$\sum_{i=1}^{n} x_{ij} = S, \qquad j = 1,\ldots,n, \qquad (2)$$

$$\sum_{(i,j)\in A_k} x_{ij} = S, \qquad k = 1,\ldots,n, \qquad (3)$$

$$x_{ij} + x_{i+1,j} + x_{i,j+1} + x_{i+1,j+1} \leq 1, \qquad i,j = 1,\ldots,n-1, \qquad (4)$$

$$x_{ij} \in \{0,1\}, \qquad i,j = 1,\ldots,n. \qquad (5)$$

Constraints (1) and (2) take care of the required number of stars in the rows and columns, respectively. Constraints (3) do the same for the islands. Next, constraints (4) observe the rule that the cells surrounding a star must be empty. This is achieved by making sure that each 2×2 square of cells contains at most one star. The binary decision variables are declared by constraints (5). Of course, in the case of $S = 1$, constraints (1), (2), and (5) correspond to those of the assignment problem (Hillier and Lieberman 2001).

3. V3ck

In the app V3ck, a graph (i.e., nodes and edges) is displayed on the screen. A number of different colors are given, and the user has to color each of the edges with one of the colors. Each node is associated with one or more numbers. Each number is displayed in one of the colors and reflects how many of the edges linked to this node have that color. Figure 2 shows a screenshot of the puzzle as well as the related solution. The puzzle is somewhat related to the edge-coloring problem in graph theory (Holyer 1981). Nevertheless, the constraints of the edge-coloring problem are different (there, adjacent edges must be assigned to different colors, which is not the case for V3ck).

For developing a formal model, let n be the number of nodes, and let C be the number of colors. The graph is given by $G = (V,E)$, where $V = \{1,\ldots,n\}$ is the set of nodes, and E is the set of edges. Parameter a_{ik} determines how many of the edges associated with node i must receive color k.

As can be seen from Figure 2, the puzzle is based on an undirected graph. For modeling purposes, however, it is more convenient to have a directed

Figure 1. Smartphone App Starbattle

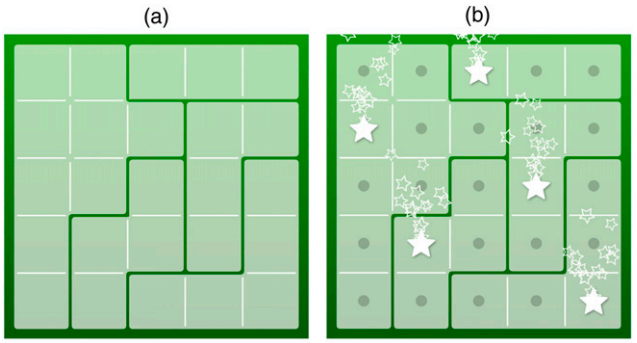
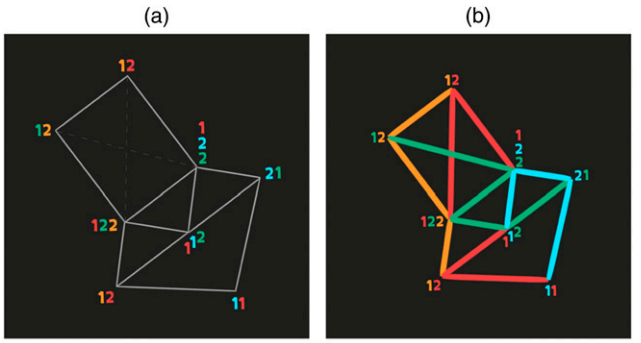


Figure 2. Smartphone App V3ck



graph; that is, each edge between nodes i and j is modeled by two directed edges or arcs (i,j) and (j,i). Now the binary decision variables that reflect the assignment of colors to the edges can be defined as follows:

$$x_{ijk} = \begin{cases} 1, & \text{if arc } (i,j) \in E \text{ obtains color } k, \\ 0, & \text{otherwise.} \end{cases}$$

As before, there is no objective, and we can give the constraints of the mathematical model:

$$\sum_{k=1}^{C} x_{ijk} = 1, \qquad (i,j) \in E, \tag{6}$$

$$x_{ijk} = x_{jik}, \qquad (i,j) \in E,\ k = 1,\dots,C, \tag{7}$$

$$\sum_{(i,j)\in E} x_{ijk} = a_{ik}, \qquad i = 1,\dots,n,\ k = 1,\dots,C, \tag{8}$$

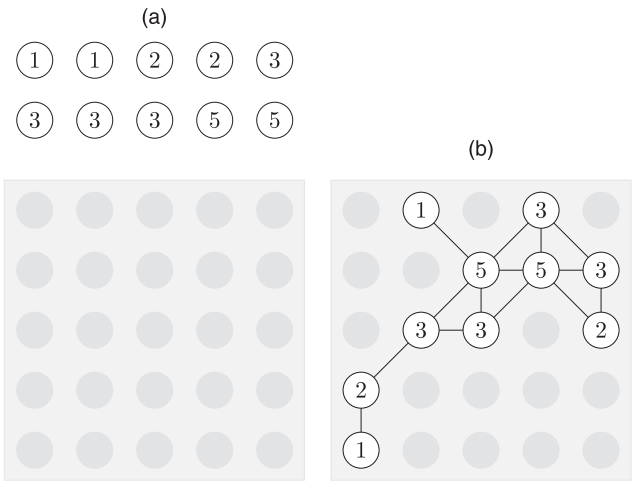
$$x_{ijk} \in \{0,1\}, \qquad (i,j) \in E,\ k = 1,\dots,C. \tag{9}$$

Constraints (6) make sure that every arc is assigned exactly one color. Next, constraints (7) observe that arcs (i,j) and (j,i) must receive the same color (this is only necessary because the model is based on a directed graph). For each node, constraints (8) check that the number of adjacent edges of each color is correct. Last, the binary decision variables are taken care of by constraints (9).

4. Monodot

In the app Monodot, the player's task is to construct a graph from a set of given nodes such that the specified number of neighbors of each node (i.e., the degree of each node) is met. In the beginning, a grid with $n \times n$ cells is displayed along with a list of nodes. The player has to drag each node to a cell of the grid. When a node is dropped, the app automatically adds an edge to each node already located in a horizontal, vertical, and diagonal neighbor cell. Each node contains a number, and when finished, this number must be equal to the degree of the node. Thus the player has to find a location for each node such that the required number of edges to neighbors is met (see Figure 3).

Figure 3. Puzzle Monodot



Puzzle Solution

In addition to the size of the grid n, we need some more notation. The number of nodes requiring t neighbors is denoted as a_t (because a node cannot have more than eight neighbors in the grid, we have $t = 1,\dots,8$). In the example of Figure 3, we have $a_3 = 4$ and $a_4 = 0$. Let $C = \{(i,j) \mid i,j = 1,\dots,n\}$ be the set of all cells of the grid. We can now define the neighborhood of cell (i,j) by

$$N_{ij} = \{(i-1,j-1),(i-1,j),(i-1,j+1),(i,j-1),(i,j+1),(i+1,j-1),(i+1,j),(i+1,j+1)\} \cap C.$$

In order to reflect the positioning of nodes in the grid, we introduce the following binary decision variables:

$$x_{ijt} = \begin{cases} 1, & \text{if a node which requires } t \text{ neighbors is} \\ & \quad \text{placed in cell } (i,j), \\ 0, & \text{otherwise.} \end{cases}$$

The constraints of the mathematical model are as follows:

$$\sum_{t=1}^{8} x_{ijt} \le 1, \qquad i,j = 1,\dots,n, \tag{10}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijt} = a_t, \qquad t = 1,\dots,8, \tag{11}$$

$$\sum_{(k,l)\in N_{ij}} \sum_{t=1}^{8} x_{klt} \ge \sum_{t=1}^{8} t x_{ijt}, \qquad i,j = 1,\dots,n, \tag{12}$$

$$\sum_{(k,l)\in N_{ij}} \sum_{t=1}^{8} x_{klt} \le \sum_{t=1}^{8} t x_{ijt} + M\Big(1 - \sum_{t=1}^{8} x_{ijt}\Big), \qquad i,j = 1,\dots,n, \tag{13}$$

$$x_{ijt} \in \{0,1\}, \qquad i,j = 1,\dots,n,\ t = 1,\dots,8. \tag{14}$$

Constraints (10) make sure that each cell of the grid is assigned at most one node. The required number of nodes with t neighbors is taken into account by constraints (11). Next, constraints (12) and (13) take care of the desired number of neighbor nodes in the resulting graph. According to (12), a node assigned to a cell has at least t neighbors, but no restriction is imposed if no node is assigned to the cell. Similarly, (13) implies that a node assigned to a cell has at most t neighbors, and again, no restriction is made if no node is assigned to the cell (M is a big number and can simply be set to 8). The final constraints (14) define the binary decision variables.

5. Knight Moves

A well-known puzzle is based on the moves of knights on a chessboard. A knight moves two squares

either vertically or horizontally and then one square to the left or to the right. The classical knight-based puzzle is to start in a given square and then successively visit all other squares of the chessboard without visiting any square more than once. There are several variants and extensions—for example, rectangular chessboards of different dimensions and irregular chessboards where certain squares may not be visited. Android apps providing knight-based puzzles include Knight's Alley, Knight Hopper, Knight's Move, and Knight Ride. Several algorithms for constructing a knight's tour (i.e., a knight's Hamiltonian path) on chessboards have been proposed in the literature; see, for example, Conrad et al. (1994) and Lin and Wei (2005).

In what follows, we consider a rectangular board with m rows and n columns (of course, the classical chessboard with $m = n = 8$ is included as a special case). The knight is placed on a square (i_s, j_s) from which it has to start. All squares must be visited exactly once in a continuous sequence of knight moves.

Let us define a graph $G = (V, E)$ where each node $(i, j) \in V$ corresponds to a square on the chessboard. For each node (i, j), the set of neighbor nodes is given by

$$N_{ij} = \{(i-2, j-1), (i-2, j+1), (i-1, j-2), (i-1, j+2), (i+1, j-2), (i+1, j+2), (i+2, j-1), (i+2, j+1)\} \cap V.$$

Of course, the neighbor nodes of node (i, j) represent the squares that can be reached from square (i, j) by a knight in a single move. The set E of edges is now defined by introducing an edge (i, j, k, l) between any two neighboring nodes $(i, j) \in V$ and $(k, l) \in N_{ij}$. In this way, the edges represent the possible knight moves.

We need two types of decision variables. Variables x_{ijkl} with $(i, j) \in V$ and $(k, l) \in N_{ij}$ are binary and reflect the edges corresponding to the selected moves of the knight:

$$x_{ijkl} = \begin{cases} 1, & \text{if the knight should move} \\ & \quad \text{from square } (i, j) \text{ to square } (k, l), \\ 0, & \text{otherwise.} \end{cases}$$

Moreover, we introduce variables $z_{ij} \geq 0$ for all nodes $(i, j) \in V$. They will be needed for a constraint that ensures that a single connected sequence of moves is generated. Now the constraints can be given as follows (again, an objective function is not needed):

$$\sum_{(k,l) \in N_{i_s, j_s}} x_{k,l,i_s,j_s} = 0, \tag{15}$$

$$\sum_{(k,l) \in N_{i_s, j_s}} x_{i_s,j_s,k,l} = 1, \tag{16}$$

$$\sum_{(k,l) \in N_{ij}} x_{klij} = 1, \qquad (i, j) \in V \setminus \{(i_s, j_s)\}, \tag{17}$$

$$\sum_{(k,l) \in N_{ij}} x_{ijkl} \leq 1, \qquad (i, j) \in V \setminus \{(i_s, j_s)\}, \tag{18}$$

$$z_{kl} - z_{ij} + M(1 - x_{ijkl}) \geq 1, \qquad (i, j) \in V,\ (k, l) \in N_{ij}, \tag{19}$$

$$x_{ijkl} \in \{0, 1\}, \qquad (i, j) \in V,\ (k, l) \in N_{ij}, \tag{20}$$

$$z_{ij} \geq 0, \qquad (i, j) \in V. \tag{21}$$

Constraints (15) and (16) make sure that there is no move to the start square and that there is one move from the start square, respectively. The next two types of constraints take care of the flow through the remaining squares. Constraints (17) observe that each such square will be visited. Moreover, constraints (18) control that a square can only be left at most once (note that the last square cannot be left). In addition, we need constraints (19), which mean that each node (i, j) is assigned a z value that is larger than that of its predecessor node (M is a large number that can be set to $m \cdot n$). In this way, cycles are made impossible, and a single connected sequence of moves is obtained. Note that these are the subtour elimination constraints suggested by Miller et al. (1960) for the traveling salesman problem. Finally, the decision variables are declared by constraints (20) and (21).

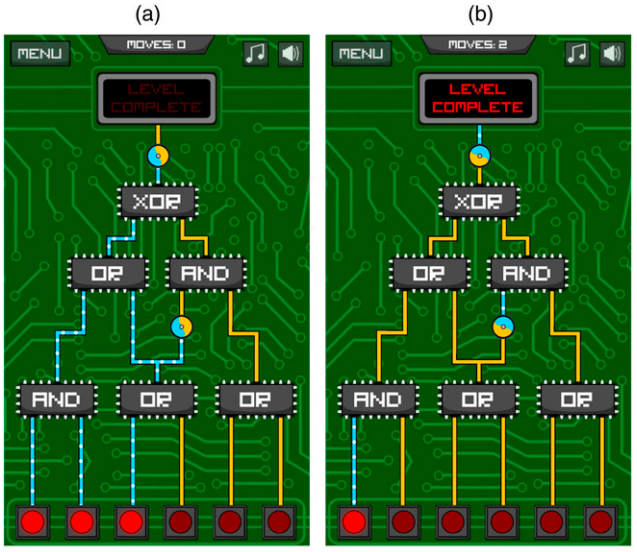
It may be mentioned that extensions of the knight move problem as given earlier can be incorporated into this model as well. In particular, irregular chessboards where certain squares may not be visited can be modeled by adapting graph G. Denoting the set of nodes reflecting the forbidden squares as F, we simply define V to contain all nodes except those included in F. The definition of the set of edges E given earlier need not be modified. The model (Equations (15)–(21)) automatically takes forbidden squares into account if V is adapted accordingly.

6. Circuit Scramble

The app Circuit Scramble displays a circuit that consists of logic gates; see the screenshots in Figure 4. There are AND, OR, and XOR gates, each with two inputs, and NOT gates with one input (the latter are called *inverters* in the app and are depicted as circles). Moreover, the circuit contains several inputs as well as one output node. The player has to decide which inputs have to be switched on (set to TRUE) or off (FALSE) such that the output node is TRUE and shows "level complete." The goal is to achieve this by changing the states of the smallest possible number of inputs.

We consider a graph with n nodes labeled $i = 1, \ldots, n$. Each input of the circuit as well as each output of a gate is reflected by a node. The set of nodes is now

Figure 4. Smartphone App Circuit Scramble



partitioned into subsets: I^1 is the set of the input nodes that are TRUE in the initial state, and I^0 is the set of the input nodes that are FALSE in the initial state. The remaining sets reflect the outputs of the different gate types: A contains the AND gates, O contains the OR gates, X contains the XOR gates, and N contains the NOT gates. Next, the circuit structure is considered. Let p_i and q_i be the two predecessor nodes (or inputs) of each AND, OR, and XOR gate, $i \in A \cup O \cup X$, and let p_i be the single predecessor (or input) of each NOT gate, $i \in N$. We assume that node n is the output node of the circuit that must be switched on.

Considering the screenshot of Figure 4, we assume that we label the nodes from bottom left to top right. Thus the six inputs would be indexed by $i = 1, \ldots, 6$, and the AND gate at the bottom left would be $i = 7$. Then we have $7 \in A$, and the predecessors of this gate are $p_7 = 1$ and $q_7 = 2$.

We introduce straightforward binary decision variables to reflect the logical state of each node $i = 1, \ldots, n$:

$$x_i = \begin{cases} 1, & \text{if node } i \text{ should be TRUE,} \\ 0, & \text{otherwise.} \end{cases}$$

Now we are ready to define the model that, unlike the previous models, contains an objective function:

$$\text{minimize} \quad \sum_{i \in I^1} (1 - x_i) + \sum_{i \in I^0} x_i, \tag{22}$$

subject to

$$x_i \le x_{p_i} \quad x_i \le x_{q_i} \quad x_i \ge x_{p_i} + x_{q_i} - 1, \qquad i \in A, \tag{23}$$

$$x_i \ge x_{p_i} \quad x_i \ge x_{q_i} \quad x_i \le x_{p_i} + x_{q_i}, \qquad i \in O, \tag{24}$$

$$x_i \le x_{p_i} + x_{q_i} \quad x_i \ge x_{p_i} - x_{q_i} \quad x_i \ge x_{q_i} - x_{p_i}$$
$$x_i \le 2 - x_{p_i} - x_{q_i}, \qquad i \in X, \tag{25}$$

$$x_i = 1 - x_{p_i}, \qquad i \in N, \tag{26}$$

$$x_n = 1, \tag{27}$$

$$x_i \in \{0, 1\}, \qquad i = 1, \ldots, n. \tag{28}$$

Objective (22) minimizes the deviation from the initial state of the inputs and thus the number of moves. Constraints (23)–(26) take care of the AND, OR, XOR, and NOR gates, respectively. Next, constraint (27) makes sure that the output of the circuit is TRUE—that is, the sign "level complete" is illuminated. The binary decision variables are defined by (28).

7. Binary Sudoku

A puzzle often referred to as Binary Sudoku includes a grid of $n \times n$ cells, some of which contain a 0 or a 1 (n is an even number). The task is to fill the grid by entering either a 0 or a 1 into each empty cell. The following rules have to be observed: First, each row and each column must contain exactly the same number of 0s and 1s. Second, no more than two consecutive cells (either horizontally or vertically) may contain the same number. Third, no two rows may be exactly the same, and no two columns may be identical. Figure 5 shows an example puzzle along with its solution.

There are many apps for this puzzle. Apps for Android include Mr. Binairo, Binaris 1001, LogiBrain Binary, Zuzu Binary Sudoku Puzzle, Binoxxo, and several others. Some apps use other symbols than 0 and 1 or two different colors, but the logic remains the same.

In addition to the size of the grid n, we need some notation to specify the initial state of the grid. Let A be the set of cells (i, j) that contain a 0 at the beginning, and let B be the set of cells that contain a 1.

We introduce binary decision variables x_{ij} that determine whether a 0 or a 1 should be entered into cell (i, j). Moreover, we need binary variables y_{ik} for constraints that ensure that rows $i = 1, \ldots, n$ and $k = i + 1, \ldots, n$ are different. Likewise, binary variables z_{ik}

Figure 5. Binary Sudoku

(a)

	0						1
0		0		1			
					1	1	
0	0			1			
0		1				1	0
			1			0	
				1		0	
1			1				0

Puzzle

(b)

1	0	1	0	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	0	1	1	0
0	0	1	0	1	1	0	1
0	1	1	0	1	0	1	0
1	0	0	1	0	1	0	1
0	1	1	0	1	0	0	1
1	1	0	1	0	0	1	0

Solution

are required for constraints that verify that columns $j = 1,\ldots,n$ and $k = j+1,\ldots,n$ are different. Now the constraints can be formulated as follows:

$$x_{ij} = 0, \qquad (i,j) \in A, \tag{29}$$

$$x_{ij} = 1, \qquad (i,j) \in B, \tag{30}$$

$$\sum_{j=1}^{n} x_{ij} = \frac{n}{2}, \qquad i = 1,\ldots,n, \tag{31}$$

$$\sum_{i=1}^{n} x_{ij} = \frac{n}{2}, \qquad j = 1,\ldots,n, \tag{32}$$

$$1 \le \sum_{k=j}^{j+2} x_{ik} \le 2, \qquad i = 1,\ldots,n,\ j = 1,\ldots,n-2, \tag{33}$$

$$1 \le \sum_{k=i}^{i+2} x_{kj} \le 2, \qquad i = 1,\ldots,n-2,\ j = 1,\ldots,n, \tag{34}$$

$$\sum_{j=1}^{n} 2^{j-1} x_{ij} - \sum_{j=1}^{n} 2^{j-1} x_{kj} \le M y_{ik} - 1, \qquad i = 1,\ldots,n,\ k = i+1,\ldots,n, \tag{35}$$

$$\sum_{j=1}^{n} 2^{j-1} x_{ij} - \sum_{j=1}^{n} 2^{j-1} x_{kj} \ge 1 - M(1 - y_{ik}), \qquad i = 1,\ldots,n,\ k = i+1,\ldots,n, \tag{36}$$

$$\sum_{i=1}^{n} 2^{i-1} x_{ij} - \sum_{i=1}^{n} 2^{i-1} x_{ik} \le M z_{jk} - 1, \qquad j = 1,\ldots,n, \quad k = j+1,\ldots,n, \tag{37}$$

$$\sum_{i=1}^{n} 2^{i-1} x_{ij} - \sum_{i=1}^{n} 2^{i-1} x_{ik} \ge 1 - M(1 - z_{jk}), \qquad j = 1,\ldots,n,\ k = j+1,\ldots,n. \tag{38}$$

Constraints (29) and (30) take the 0s and 1s of the initial state of the grid into account. Next, constraints (31) and (32) make sure that there are as many 0s as 1s in each row and each column. The rule that no more than two adjacent cells may contain the same number is taken care of by constraints (33) for rows and (34) for columns, respectively. They control that the sum of any three neighboring cells must be at least 1 (which avoids three 0s) and at most 2 (which avoids three 1s). The requirement that all rows must be different is considered by constraints (35) and (36), and constraints (37) and (38) do the same for columns. The binary vectors of the rows and columns are transformed into integers, and the constraints then make sure that these integers are not equal. With the help of additional variables y_{ik} and z_{ik},, we obtain constraints reflecting inequality ($\neq$). Note that M is a big number and can be set to $M = 2^{n+1}$. It should be noted that these constraints to achieve inequality work only for integer values.

8. Conclusions

This article has presented six logic puzzles that are available as smartphone apps, along with mathematical models to solve them. As in Hartmann (2018), the puzzle apps and models are used in the OR lecture in two bachelor of science programs—namely business administration and business informatics. The lecture is given in the fourth semester, after the students have completed basic lectures on mathematics.

The main purpose when using puzzles is to provide interesting examples for in-class discussions of modeling approaches as well as further exercises that students can do at home. In order to encourage students who are less familiar with mathematical formalisms, we often start with the development of models for given instances (rather than general models). The screenshots used in this paper are used as instances. Working with instances is perceived as being much easier than working with general models by most students.

Moreover, it should be mentioned that discussing logic puzzles can help to demonstrate the broad applicability of integer linear programming and the power of the algorithms included in solvers. It can be discussed in class that variables, linear constraints, and objective can, in fact, be seen as a specific "language" and that once a problem has been translated to that language, standard algorithms can be used to solve it. Whereas mathematical programming is fairly restricted because it allows only for linear expressions, it is also very general because it can solve many different types of problem settings. Logic puzzles can be used to emphasize that this is not limited to typical OR problems.

Of course, the main goal when teaching mathematical programming (and OR in general) is to enable students to identify problems in practice and to solve them. Beyond that, however, another goal is to improve the general analytical skills of the students. In this context, modeling logic puzzles can be a valuable addition to the modeling of the usual OR problems.

Acknowledgments

I am grateful to the developers of the apps for their kind permissions to reproduce the screenshots: Falk Kühnel, Alexander Kylburg, and Patrick Albertini (Starbattle; Paragraph Eins, Germany); Leia and Nicolas Jobard (V3ck; Dark Eye, France); and Rob van Staalduinen (Circuit Scramble; Suborbital Games, Inc., Canada). I also thank my daughter Solvei, who pointed me to the app Monodot.

References

Bartlett A, Chartier TP, Langville AN, Rankin, TD (2008) Integer programming model for the Sudoku problem. *J. Online Math. Appl.* 8(May):1–14.

Chlond MJ (2005) Classroom exercises in IP modeling: Su Doku and the log pile. *INFORMS Trans. Ed.* 5(2):77–79.

Chlond MJ (2010) Knight domination of 2-D surfaces: A spreadsheet approach. *INFORMS Trans. Ed*. 10(2):98–101.

Chlond MJ (2014) Logic grid puzzles. *INFORMS Trans. Ed*. 15(1): 166–168.

Chlond MJ, Toase CM (2002) IP modeling of chessboard placements and related puzzles. *INFORMS Trans. Ed*. 2(2):1–11.

Conrad A, Hindrichs T, Morsy H, Wegener I (1994) Solution of the knight's Hamiltonian path problem on chessboards. *Discrete Appl. Math.* 50(2):125–134.

Hartmann S (2018) Solving smartphone puzzle apps by mathematical programming. *INFORMS Trans. Ed*. 18(2):127–141.

Hillier FS, Lieberman GJ (2001) *Introduction to Operations Research*, 7th ed. (McGraw-Hill, New York).

Holyer I (1981) The NP-completeness of edge-colouring. *SIAM J. Comput.* 10(4):718–720.

Koch T (2006) Rapid mathematical programming or how to solve Sudoku puzzles in a few seconds. Haasis H-D, Kopfer H, Schönberger J, eds. *German Oper. Res. Proc. 2005* (Springer, Berlin), 21–26.

Letavec C, Ruggiero J (2002) The *n*-queens problem. *INFORMS Trans. Ed*. 2(3):101–103.

Lin S-S, Wei C-L (2005) Optimal algorithms for constructing knight's tours on arbitrary $n \times m$ chessboards. *Discrete Appl. Math.* 146(3): 219–232.

Makhorin A (2010) *Modeling Language GNU MathProg: Language Reference for GLPK Version 4.45* (Free Software Foundation, Boston).

Miller CE, Tucker AW, Zemlin RA (1960) Integer programming formulation of traveling salesman problems. *J. ACM* 7(4):326–329.

Oki E (2012) *Linear Programming and Algorithms for Communication Networks: A Practical Guide to Network Design, Control, and Management* (CRC Press, Boca Raton, FL).

Rasmussen RA, Weiss HJ (2007) Advanced lessons on the craft of optimization modeling based on modeling Sudoku in Excel. *INFORMS Trans. Ed*. 7(3):228–237.

Weiss HJ, Rasmussen RA (2007) Lessons from modeling Sudoku in Excel. *INFORMS Trans. Ed*. 7(2):178–184.

Yeomans JS (2003) Solving "Einstein's riddle" using spreadsheet optimization. *INFORMS Trans. Ed*. 3(2):55–63.