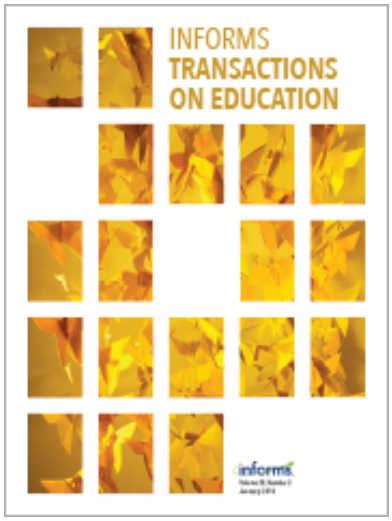


INFORMS Transactions on Education

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

The Snake Eggs Puzzle: Preparing Students for Benders Decomposition

Mitchell Harris, Michael Forbes

To cite this article:
Mitchell Harris, Michael Forbes (2023) The Snake Eggs Puzzle: Preparing Students for Benders Decomposition. INFORMS Transactions on Education 23(3):210-217. https://doi.org/10.1287/ited.2023.0281

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

INFORMS TRANSACTIONS ON EDUCATION
Vol. 23, No. 3, May 2023, pp. 210–217
ISSN 1532-0545 (online)

Puzzle

The Snake Eggs Puzzle: Preparing Students for Benders Decomposition

Mitchell Harris,[a,*] Michael Forbes[a]

[a] School of Mathematics and Physics, The University of Queensland, Brisbane, St Lucia, QLD 4072, Australia
*Corresponding author
Contact: m.g.harris@uq.edu.au, https://orcid.org/0000-0002-0939-8516 (MH); m.forbes@uq.edu.au (MF)

Received: August 29, 2022
Revised: October 25, 2022
Accepted: October 26, 2022
Published Online in Articles in Advance: February 8, 2023

https://doi.org/10.1287/ited.2023.0281

Abstract. Logic puzzles are an effective way to introduce students to advanced solution techniques in operations research, such as Lagrangian relaxation, Dantzig-Wolfe decomposition, and Benders decomposition. The Snake Egg puzzle asks the player to draw a one-cell wide path, or "snake," in a grid. The remaining cells should form a fixed number of separate, connected, discontiguous regions called "eggs." We propose two solution approaches: a flow-based model and lazy constraints. Instead of providing the complete model at the outset, we will step through the puzzle in a manner suitable to the classroom, emphasizing the skills that are crucial to successfully implementing advanced techniques. The puzzle functions in particular as a prelude to Benders decomposition.

Funding: M. Harris is supported by an Australian Government RTP (research training program) scholarship.

Keywords: integer programming • puzzles • lazy constraints • Benders decomposition

1. Introduction

Logic puzzles are an effective way to teach modeling techniques in operations research, because many puzzles can be formulated as integer programs (IPs). Sudoku, for instance, is now a classical teaching example; see Conforti et al. (2014). Chlond (2015a) formulated an IP model of a chess avoidance puzzle. Chlond (2015b) formulated IP models of the Wijuko and ABC logic puzzles. Hartmann (2018) formulated IP models of several popular mobile puzzles. Many other examples can be found in this journal. But besides providing many IP modeling examples, logic puzzles are also an opportunity to demonstrate the kind of speedups that can be achieved by using advanced operations research techniques, such as Lagrangian relaxation, Dantzig-Wolfe decomposition, and, of most interest to us, Benders decomposition. For example, Pearce and Forbes (2017) proposed both row and column generation approaches to the so-called fillomino puzzle, which is difficult to solve by a direct approach.

Benders decomposition is a method of solving certain difficult optimization problems, where new constraints are added to a master problem in each iteration; see Benders (1962). Row generation algorithms like this have historically been implemented either in a loop, where the master problem is solved from scratch in each iteration, or by a specialized branch-and-cut procedure. Since 2012, however, modern solvers such as Gurobi (Gurobi Optimization LLC 2022) have included lazy constraints, allowing us to easily add user cuts at nodes of a branch-and-bound tree.

Lazy constraints, unlike valid inequalities, are allowed to cut off feasible solutions to the base model. In that way, they can be used to enforce properties that have been left out of the full model. A classic example is the Dantzig-Fulkerson-Johnson formulation of the traveling salesman problem. Although the exact formulation has exponentially many subtour elimination constraints, by adding only the ones that are violated by incumbent solutions to the relaxed problem, we can solve even relatively large instances in a reasonable amount of time; see Dantzig et al. (1954).

Teaching experience indicates that Benders decomposition is a nontrivial conceptual step above the standard linear and integer programming techniques covered during a first course in operations research. Thus, it is beneficial to introduce some of the basic notions with a logic puzzle before covering the correctness proof or any industrial applications. In this paper we will study the Snake Egg puzzle. The puzzle is credited to Serkan Yürekli and (at this time) can be found at https://www.gmpuzzles.com/blog/2016/01/snake-egg-by-serkan-yurekli/.

After stating the puzzle, we will try to formulate it as an integer program, visualizing the new solution each time new constraints are formulated. In doing so, we stress the value of incremental modelling and the types of early end-to-end testing techniques that are crucial for successfully implementing an advanced technique.

One of the puzzle rules—the connectivity of the snake eggs—turns out to be the most challenging part. We consider two approaches to enforcing this rule: a multicommodity flow approach and lazy constraints. Although enforcing connectivity is an interesting use of flow-based constraints, we find the lazy approach to be superior with respect to runtime. We also demonstrate some useful techniques and principles for benchmarking various approaches to a problem.

After solving the snake egg puzzle, students can meet Benders decomposition familiar with several important concepts, such as omitting the complicating aspects of a problem, adding new constraints at nodes of the branch-and-bound tree, the importance of strong cutting planes, and disaggregated cuts.

1.1. The Snake Egg Puzzle

The Snake Egg puzzle is played on a grid of cells. We want to draw a *snake*—that is, a one-cell wide path moving orthogonally—in the grid. The head and tail of the snake are fixed and labeled with an "O" in our figures. The snake may not touch itself orthogonally. The remaining cells should form a fixed number of separate, connected, discontiguous regions—or *eggs*—of sizes $1, 2, 3, \ldots$, etc., up to the specified number of eggs. The value of some cells may be fixed in advance. We consider two instances: a 6×6 instance with five eggs and the 10×10 instance with nine eggs from the aforementioned blog post (see Figure 1).

Figure 1. Two Instances of the Snake Egg Puzzle with Solutions

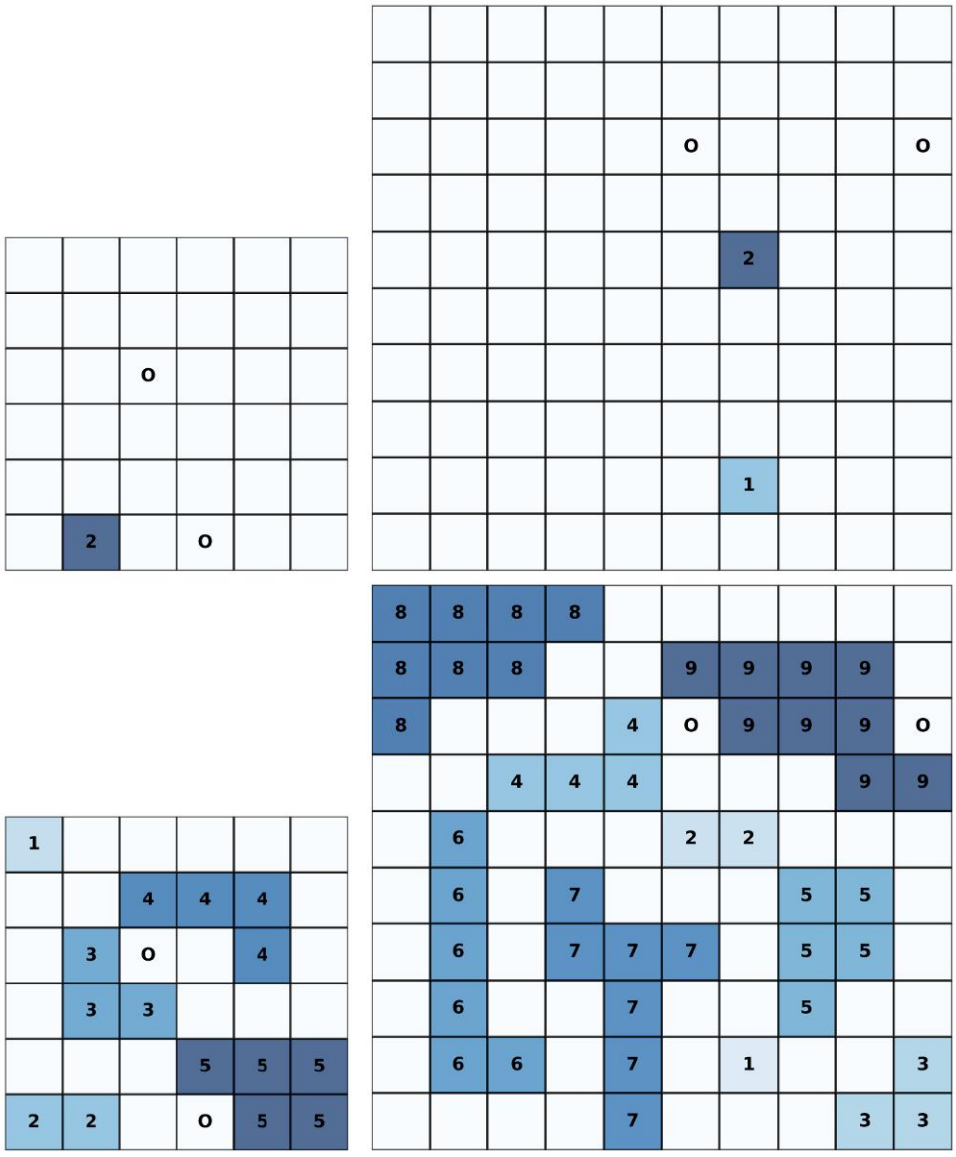


Table 1. Notation

Sets	Descriptions
S	The set of cells
$N(s) \subset S$	The set of cells adjacent to cell $s \in S$
$K \subset \mathbb{N}_1$	The set of egg sizes
$K_0 = K \cup \{0\}$	The set of cell types
$P \subset S$	The set of cells with predetermined values
$\mathsf{pre}(s) \in K_0$	The predetermined value of cell $s \in P$

2. Formulation

The puzzle is characterized by the following sets and data (Table 1).

To formulate the puzzle as an integer program, we need to determine a suitable set of variables. Because we want to associate each cell with an integer k, it is tempting to try and use general integer variables. But this will make it difficult to write down the constraints. Instead, we define binary variables, $x_{sk} \in \{0,1\}$ for $s \in S$ and $k \in K_0$, with the following interpretation:

$$x_{sk} = 1 \text{ if and only if cell } s \text{ is part of egg } k.$$

Here, $x_{s0} = 1$ means that cell s is part of the snake. In our visualizations, we leave snake tiles blank, except for the head and tail.

Using binary variables—which are more numerous—may contradict the instinct to minimize the number of integer variables when possible. But modern IP solvers are very good at exploiting the geometry of the unit hypercube for the generation of cutting planes, when making branching decisions, in pre-solve, and in other contexts.

Several constraints are now obvious:

- Each cell belongs to exactly one egg or the snake:

$$\sum_{k \in K_0} x_{sk} = 1 \quad \text{for all } s \in S. \tag{1}$$

- There are k cells in egg k for $k \geqslant 1$:

$$\sum_{s \in S} x_{sk} = k \quad \text{for all } k \in K. \tag{2}$$

Figure 2. A Solution to (1–3)

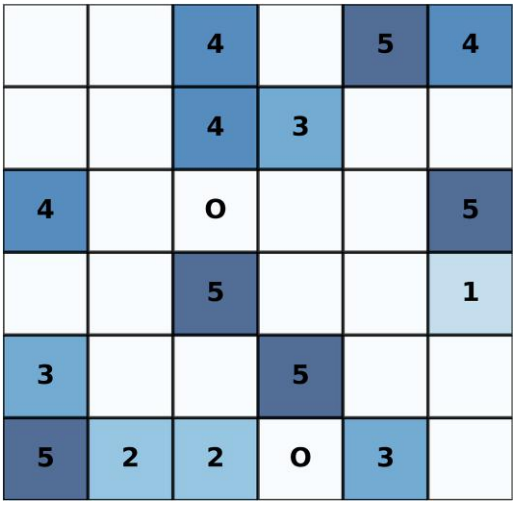
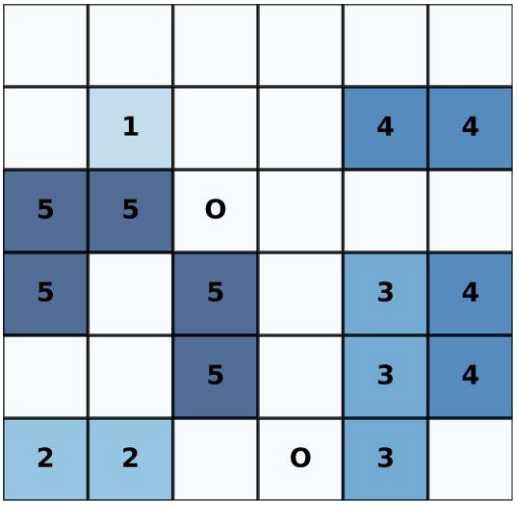


Figure 3. A Solution to (1–4)



- The values of preset cells are fixed:

$$x_{\mathsf{spre}(s)} = 1 \quad \text{for all } s \in P. \tag{3}$$

We can solve (1–3) to get a starting visualization that helps to prompt other constraints; see Figure 2.

Although the eggs are the right size, they have been scrambled. We want to pull together the cells of similar type. A start is to observe that each cell of type $k \in K$, $k \geqslant 2$ should have at least one neighbor with the same egg type. We can impose this with

$$x_{sk} \leqslant \sum_{s' \in N(s)} x_{s'k} \quad \text{for all } s \in S \text{ and all } k \in K,\ k \geqslant 2. \tag{4}$$

After re-solving the model, we get the solution in Figure 3.

This is closer to a valid solution, but different eggs can still touch. To push the eggs apart we impose

Figure 4. A Solution to (1–5)

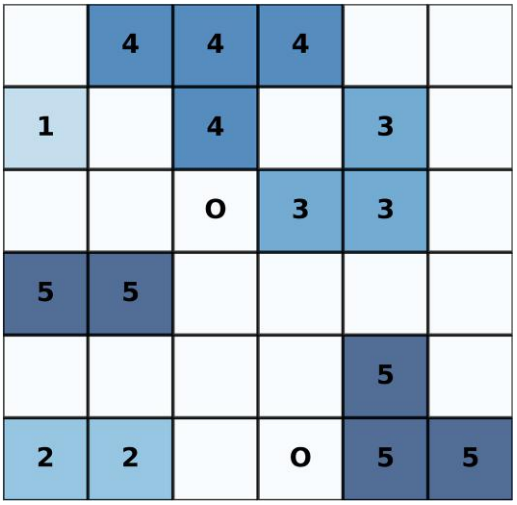
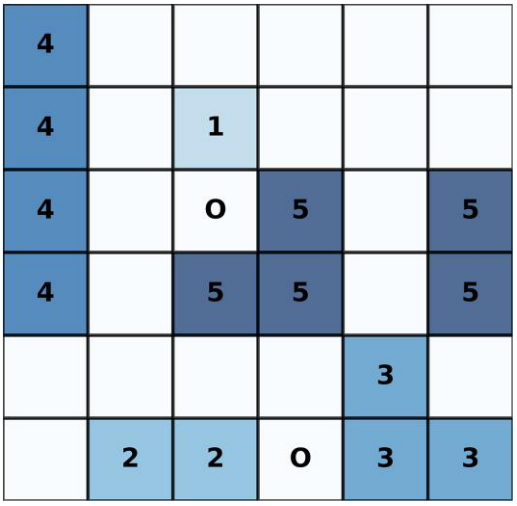


Figure 5. A Solution to (1–6)



$$x_{sk} + \sum_{k' \in K \setminus \{k\}} x_{s'k'} \leqslant 1 \quad \text{for all } s \in S \text{ and all } k \in K \text{ and all } s' \in N(s). \tag{5}$$

Now if cell s is a part of egg k, then each cell $s' \in N(s)$ is part of k as well, or it is part of the snake. Solving the model again gives the solution in Figure 4.

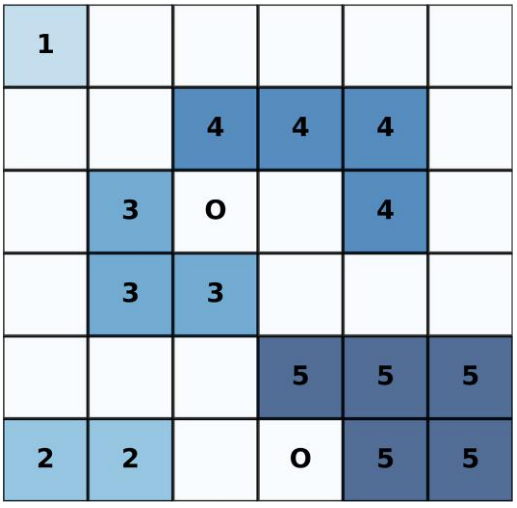
There are two more obvious problems; the 5th egg is disconnected, and the snake is not actually a snake. We know that the head and tail of the snake need to be adjacent to exactly one other snake cell each:

$$\sum_{s' \in N(s)} x_{s'0} = 1 \quad \text{for all } s \in P \text{ such that } \mathsf{pre}(s) = 0. \tag{6}$$

This gives the solution in Figure 5. Next, each interior cell of the snake should be adjacent to exactly two other snake cells. A common suggestion from students is

$$\sum_{s' \in N(s)} x_{s'0} = 2x_{s0} \quad \text{for all } s \in S \setminus P.$$

Figure 6. A Solution to (1–8)



This works if $x_{s0} = 1$. But if $x_{s0} = 0$, then it prohibits any neighbor of s from being a snake cell, leading to an infeasible model. Instead, we add two sets of constraints:

$$\sum_{s' \in N(s)} x_{s'0} \geqslant 2x_{s0} \quad \text{for all } s \in S \setminus P. \tag{7}$$

$$\sum_{s' \in N(s)} x_{s'0} \leqslant 4 - 2x_{s0} \quad \text{for all } s \in S \setminus P. \tag{8}$$

If $x_{s0} = 1$, then together these constraints imply $\sum_{s' \in N(s)} x_{s'0} = 2$, as required. But when $x_{s0} = 0$, we get $0 \leqslant \sum_{s' \in N(s)} x_{s'0} \leqslant 4$, which is vacuous. Re-solving the model now gives the solution in Figure 6.

This is a valid solution to the 5×5 instance. But an astute student will notice that the 4th and 5th eggs are connected only by coincidence. In general, (4) is sufficient only for the 2nd and 3rd eggs. This becomes clear when we try to solve the 10×10 instance. See Figure 7, where the 8th and 9th eggs are disconnected and there is a separate loop in the snake. We will consider two approaches to enforcing connectedness and eliminating loops.

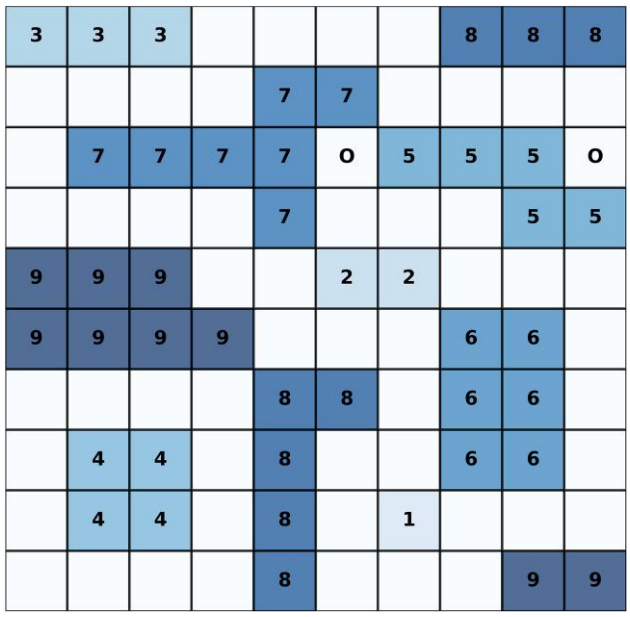
2.1. Flow Formulation

We can enforce the continuity of the snake eggs and the snake by imposing a multicommodity flow superstructure. The idea is to choose an arbitrary source for each egg and the snake. Each other cell in each egg receives one unit of flow from its source. Flow conservation then guarantees that the cells are connected.

For each $s \in S$ and $k \in K_0$, $k \neq 1$, let y_{sk} be a binary variable with the following interpretation:

$y_{sk} = 1$ if and only if cell s is the source node for egg k.

Figure 7. A Solution to (1–8) on the 10×10 Puzzle



For $s \in S$, $s' \in N(s)$, and $k \in K_0$, $k \neq 1$, let $f_{ss'k}$ be a continuous variable with the following interpretation:

$f_{ss'k}$ is the flow of commodity k from cell s to cell s'.

We make sure there is only one source per egg, including the snake, with

$$\sum_{s \in S} y_{sk} = 1 \quad \text{for all } k \in K_0,\ k \neq 1. \tag{9}$$

We guarantee that only a type k cell can be the source for egg k with

$$y_{sk} \leqslant x_{sk} \quad \text{for all } s \in S \text{ and all } k \in K_0,\ k \neq 1. \tag{10}$$

For $k \geqslant 2$, we limit the outflow of each cell to $k-1$ if that cell is the source of commodity k, to $k-2$ if it is a type k cell, but not a source, and to zero otherwise with

$$\sum_{s' \in N(s)} f_{ss'k} \leqslant (k-2)x_{sk} + y_{sk} \quad \text{for all } s \in S \text{ and all } k \in K,\ k \geqslant 2. \tag{11}$$

We enforce flow conservation for the eggs with

$$\sum_{s' \in N(s)} f_{s'sk} - \sum_{s' \in N(s)} f_{ss'k} = x_{sk} - k y_{sk} \quad \text{for all } s \in S \text{ and all } k \in K,\ k \geqslant 2. \tag{12}$$

In other words, if cell s is the source for commodity k, then the net outflow is $k-1$. Otherwise, the net inflow is 1. Finally, we enforce flow conservation for the snake with

$$\sum_{s' \in N(s)} f_{s's0} - \sum_{s' \in N(s)} f_{ss'0} = x_{s0} - M y_{s0} \quad \forall s \in S, \tag{13}$$

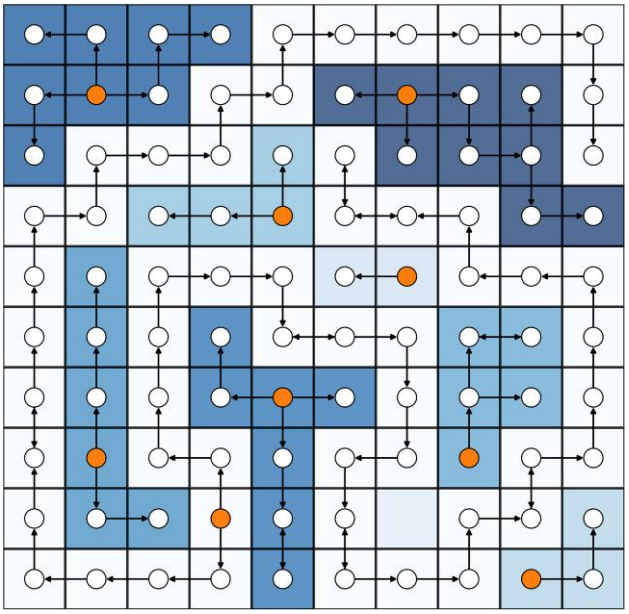
where $M = |S| - \sum_{k \in K} k$ is the length of the snake. Adding (9), (10), (11), (12), and (13) gives us an exact formulation of the snake egg puzzle, and—after solving the new model—the solution in Figure 8. Source nodes are marked in orange, and the flows are indicated with arrows.

Note that retaining (4) means no flows of commodities 2 and 3 are needed. We could also fix the source of the snake flow to one of the ends. Constraints (1–13) constitute a compact formulation of the snake egg puzzle, because the number of variables and constraints is polynomial in the size of the puzzle.

3. Lazy Constraints

Although the flow formulation of the 10×10 puzzle is not intractable, we can speed up the solution time by using lazy constraints. In this section, we drop the flow variables and constraints (9–13). We will call (1–8) the *master problem*. One way to solve the Snake Egg puzzle is to solve the master problem over and over again in a loop, where in each iteration we check whether the eggs are connected. If they are not, we add a new constraint that bans that solution and continue. Say that x' is the incumbent solution. One such constraint is

Figure 8. A Solution to (1–13) and a Valid Solution to the 10×10 Puzzle



$$\sum_{s \in S} \sum_{\substack{k \in K_0 : \\ x'_{sk} = 0}} x_{sk} \geqslant 1,$$

which says that the value of at least one cell needs to change. A cut like this is called a "no-good cut" because it eliminates the current "no-good" solution. Because there are only finitely many solutions to the master problem, eventually we are guaranteed to find a "good" solution. But this is essentially a brute-force approach and is certainly worse than the flow approach.

The question to ask ourselves is as follows: Given the current master solution, can we come up with a stronger constraint that not only cuts off the current solution but also a large number of other bad solutions at the same time, a good cut, so to speak? Say that C is a disconnected subset of the k^{th} egg in the current solution. The no-good cut can be satisfied by exchanging the value of any two cells in $S \setminus C$ while preserving the disconnected egg. But because $|C| < k$ and the eggs must be connected, we know that region C can only be part of egg k if at least one cell directly adjacent to C is part of the k^{th} egg as well. Let's make this formal.

For $k \in K$, define $\mathcal{C}_k$ to be the set of all subsets $C \subset S$ with $|C| < k$ such that C is connected. For $C \in \mathcal{C}_k$, let

$$N(C) = \bigcup_{s \in S} N(s) \setminus C$$

denote the set of strict neighbors of C. Consider the following family of constraints:

$$\sum_{s \in C} x_{sk} \leqslant |C| \sum_{s' \in N(C)} x_{s'k} \quad \forall k \in K,\ k \geqslant 2,\ \forall C \in \mathcal{C}_k. \tag{14}$$

In other words, for $k \in K$, $k \geqslant 2$ and $C \in \mathcal{C}_k$, the set C may not be a subset of egg k unless at least one neighbor of C is, too. Now (1–8, 14) is an exact formulation of the Snake Egg puzzle. A formal proof of this fact is in the Appendix.

Unlike the flow model, this is a noncompact formulation, because (14) is an exponential number of constraints. Therefore, adding them all to the model is not an option. Instead, we can solve the master problem and, at nodes of the branch-and-bound tree, use special-purpose code to detect the disconnected eggs. For each connected component of a disconnected egg, we add the corresponding constraint from (14) as a lazy constraint and continue with the solve. The hope is that significantly fewer constraints will be needed before we discover a good solution.

It turns out that we can do even better by disaggregating the cuts. The following family of constraints also guarantees that all eggs are contiguous:

$$x_{sk} \leqslant \sum_{s' \in N(C)} x_{s'k} \quad \forall k \in K \text{ with } k \geqslant 2, \quad \forall C \in \mathcal{C}_k, \quad \forall s \in C. \tag{15}$$

A formal proof of this, too, may be found in the Appendix. The constraints state that each individual cell in the disconnected region is deactivated unless at least one more cell on its boundary has the same type. Note that the "aggregated" cuts are obtained by summing the disaggregated cuts over $s \in C$. We will add each cut as a separate lazy constraint, as required.

3.1. Results

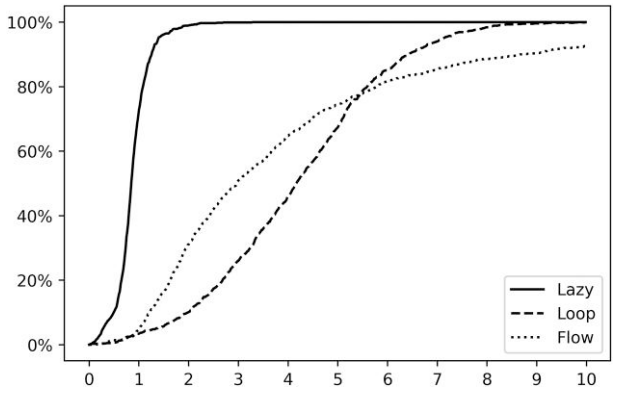
Integer programming solvers are well-known to be chaotic systems; runtimes can vary dramatically with the path taken through the branch-and-bound tree. This is even more pronounced with feasibility problems, where with good luck we could stumble upon a solution very quickly. Therefore, to do rigorous benchmarking, we need to test the proposed approaches on many instances. We would like to compare the following approaches to the 10×10 puzzle:

- Solving (1–8) once with (15) added as lazy constraints; Lazy
- Solving (1–8) from scratch in each iteration, with (15) added as regular constraints; Loop,
- (1–13); Flow.

We solved the 10×10 instance 1,000 times for each approach using distinct seeds, which impact the path taken through the tree. The models were programmed in Python 3.8.12 via the Anaconda distribution (4.11.0) and solved using Gurobi 9.1.2 (Gurobi Optimization LLC 2022). Jobs were run in parallel on a computing cluster operating 2.5 GHz CPUs. Each job was given a time limit of 10 minutes and a single thread. Performance profiles can be found in Figure 9.

The horizontal axis represents time in minutes. The vertical axis represents the percentage of 1,000 instances solved within that time. Although some of the fastest runs of the flow model outperform the slowest runs of the lazy model, the general trend is clear. The lazy model solves all instances within 3 minutes, and many of them much faster than that. On the other hand, the flow model fails to solve some instances even within 10 minutes. The intersection of the Loop and Flow curves is also interesting. Minimum, maximum, mean, and standard deviations of the runtimes are given in Table 2; note that the Flow column is skewed in its own favor because 68 instances timed out before finding a solution.

Figure 9. Performance Profiles for Three Methods



3.2. Improvements

Students often suggest improvements to the model in class. In this section, we outline two of these suggestions.

Let $\mathcal{S}$ be the set of all 2×2 connected squares of the form

$$C = \{(i,j),(i+1,j),(i,j+1),(i+1,j+1)\} \subset S \quad \text{for some } (i,j) \in S.$$

Then,

$$\sum_{s \in C} x_{s0} \leqslant 3 \quad \forall C \in \mathcal{S} \tag{16}$$

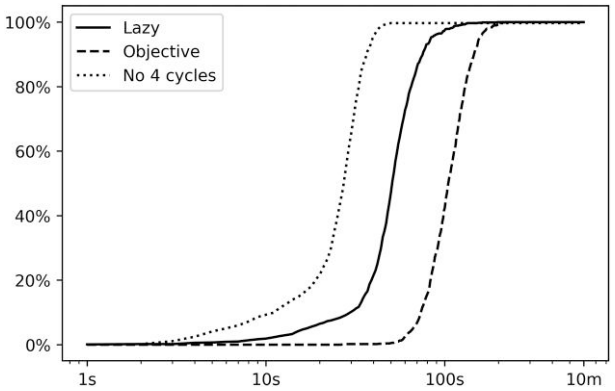
are valid inequalities. To see why this might help, note that for the snake, (15) are equivalent to classical cycle elimination constraints. Constraints (16) cut off all snake cycles of length 4 in advance. The student had observed that many of the incumbent solutions had several 4-cycles.

A second suggestion from students came before the lazy constraints had been introduced. Having asked the

Table 2. Comparing Three Formulations

Time (s)	Lazy	Loop	Flow
Minimum	0.37	0.39	7.9
Maximum	196.32	585.92	616.79
Mean	52.19	250.56	202.37
SD	20.62	103.31	129.89

Figure 10. Performance Profiles for Suggested Improvements



students to think of ways to enforce the connectivity of the snake eggs, one suggestion was to introduce an objective function maximizing the number of connections between adjacent egg cells. Although a novel and very reasonable suggestion, we were able to confirm that it does not produce a valid solution to the puzzle; the optimal solution includes a disconnected loop in the snake. What we can do is incorporate the objective function into the lazy model and simply terminate when a valid solution is found. The hope is that the objective will encourage the solver to favor connected eggs.

Figure 10 contains performance profiles for the standard lazy model, the lazy model with an objective function, and the lazy model with the valid inequalities.

This time we have plotted the profiles on a logarithmic scale. We see that including an objective function actually hinders the model. On the other hand, the valid inequalities approximately halved the runtime. See the table of results (Table 3).

4. Conclusion

In this paper, we have shown how up to a 10×10 instance of the Snake Egg puzzle with nine eggs can be solved using lazy constraints. Instead of providing the complete model at the outset, we stepped through the puzzle in a manner suitable to the classroom. Advantages of this include (i) the early detection of errors and (ii) early visualization of partial solutions. We benchmarked several approaches and found the lazy approach to be superior with respect to runtime. The lazy approach we have described is, in fact, an example of so-called Logic-Based Benders decomposition; see Hooker (2000) and Hooker and Ottosson (2003). In that context, the lazy constraints are called feasibility cuts. We believe that the snake egg puzzle is an effective pedagogical prelude to both classical and logic-based Benders decomposition.

Table 3. Comparing Two Potential Improvements

Time (s)	Lazy	No-4	Obj.
Minimum	0.37	1.25	26.98
Maximum	196.32	49.1	218.93
Mean	52.19	25.7	107.84
Dev	20.62	9.4	28.52

Acknowledgments

We thank the editors, reviewers, and all involved for their consideration of our work.

Appendix. Correctness of the Lazy Constraints

Theorem 1. *Fix $k \geqslant 2$ and an integer solution x' to (1–8), and let $C = \{s \in S : x'_{sk} = 1\}$. Then, x' satisfies every constraint of (14) indexed by k if and only if C is connected.*

Proof. $\Leftarrow$) Suppose C is connected. Choose $C' \in \mathcal{C}_k$. If $N(C') \cup C \neq \emptyset$, then

$$\sum_{s \in C'} x_{sk} \leqslant |C'| \leqslant |C'| \sum_{s' \in N(C')} x_{s'k}.$$

If $N(C') \cup C = \emptyset$, then

$$0 = \sum_{s \in C'} x_{sk} \leqslant |C'| \sum_{s' \in N(C')} x_{s'k}.$$

Because C' was arbitrary, every constraint in (14) indexed by k is satisfied by x'.

$\Rightarrow$) Conversely, suppose C is not connected. Without loss of generality, there exists $C_1 \in \mathcal{C}_k \setminus \{C\}$ and $C_2 \in \mathcal{C}_k \setminus \{C, C_1\}$ with $C = C_1 \cup C_2$ and $N(C_1) \cup C_2 = N(C_2) \cup C_1 = \emptyset$. Therefore,

$$1 \leqslant \sum_{s \in C_i} x_{sk} \leqslant |C_i| \sum_{s' \in N(C_1)} x'_{sk} = 0$$

for $i \in \{1,2\}$, which is a contradiction. □

Theorem 2. *Fix $k \geqslant 2$ and an integer solution x' to (1–8), and let $C = \{s \in S : x'_{sk} = 1\}$. Then, for $s \in C$, x' satisfies every constraint of (15) indexed by k and s if and only if C is connected.*

Proof. $\Leftarrow$) Suppose C is connected. Choose $C' \in \mathcal{C}_k$ and $s \in C'$. If $N(C') \cup C \neq \emptyset$, then

$$x_{sk} \leqslant 1 \leqslant \sum_{s' \in N(C')} x_{s'k}.$$

If $N(C') \cup C = \emptyset$, then

$$0 = x_{sk} \leqslant \sum_{s' \in N(C')} x_{s'k}.$$

Because C' and s were arbitrary, x' satisfies every constraint in (15).

$\Rightarrow$) Suppose C is not connected. Without loss of generality, there exists $C_1 \in \mathcal{C}_k \setminus \{C\}$ and $C_2 \in \mathcal{C}_k \setminus \{C, C_1\}$ with $C = C_1 \cup C_2$ and $N(C_1) \cup C_2 = N(C_2) \cup C_1 = \emptyset$. Therefore, for each $s \in C$,

$$1 = x_{sk} \leqslant \sum_{s' \in N(C_i)} x_{s'k} = 0$$

for $i \in \{1,2\}$, which is a contradiction. □

References

Benders J (1962) Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* 4:238–252.

Chlond M (2015a) Puzzle—chess avoidance puzzles. *Trans. Educ.* 15(3):254–256.

Chlond M (2015b) Puzzle—ip in the i. *Trans. Educ.* 16(1):39–41.
Conforti M, Cornuéjols G, Zambelli G (2014) *Integer Programming*, volume 271. (Springer). https://link.springer.com/book/10.1007/978-3-319-11008-0.
Dantzig G, Fulkerson R, Johnson S (1954) Solution of a large-scale traveling-salesman problem. *J. Oper. Res. Soc. Am.* 2(4):393–410.
Gurobi Optimization LLC (2022) Gurobi Optimizer Reference Manual. https://www.gurobi.com.
Hartmann S (2018) Puzzle-solving smartphone puzzle apps by mathematical programming. *Trans. Educ.* 18(2):127–141.
Hooker JN (2000) *Logic-based Methods for Optimization: Combining Optimization and Constraint Satisfaction.* (Wiley, New York).
Hooker JN, Ottosson G (2003) Logic-based benders decomposition. *Math. Programming* 96(1):33–60.
Pearce RH, Forbes MA (2017) Puzzle-the fillomino puzzle. *Trans. Educ.* 17(2):85–89.