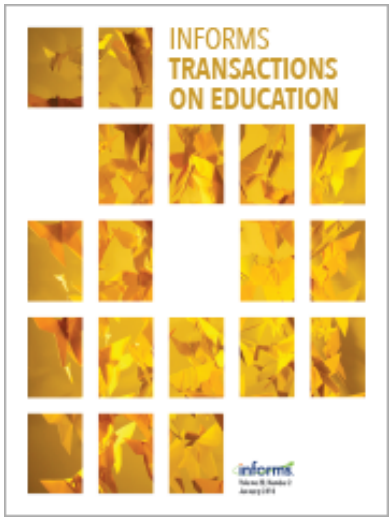


INFORMS Transactions on Education

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

Puzzle—Queen Puzzles: An Educational Approach to Integer Programming Techniques

Natalie DiMichele, Michael Forbes

To cite this article:
Natalie DiMichele, Michael Forbes (2026) Puzzle—Queen Puzzles: An Educational Approach to Integer Programming Techniques. INFORMS Transactions on Education 26(3):247-253. https://doi.org/10.1287/ited.2024.0122

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

INFORMS TRANSACTIONS ON EDUCATION
Vol. 26, No. 3, May 2026, pp. 247–253
ISSN 1532-0545 (online)

Puzzle

Queen Puzzles: An Educational Approach to Integer Programming Techniques

Natalie DiMichele,[a,*] Michael Forbes[a]

[a] School of Mathematics and Physics, The University of Queensland, Brisbane, St. Lucia, Queensland 4072, Australia
*Corresponding author
Contact: n.dimichele@uq.edu.au, https://orcid.org/0009-0008-3361-2747 (ND); m.forbes@uq.edu.au, https://orcid.org/0000-0002-2240-7245 (MF)

Received: November 5, 2024
Revised: January 28, 2025; February 25, 2025
Accepted: March 3, 2025
Published Online in Articles in Advance: August 1, 2025

https://doi.org/10.1287/ited.2024.0122

Abstract. Chessboard puzzles involving queens are valuable teaching examples for integer programming. We present the queens domination and peaceable queens problems in an educational format and describe our classroom experience with teaching these problems. We discuss techniques for improving efficiency, such as symmetry breaking, valid inequalities, bounds, and parameters.

Funding: This work was supported by the Australian Government (RTP Scholarship).

Keywords: integer programming • queens • puzzles • symmetry breaking • valid inequalities • solver parameters

1. Introduction

Logic puzzles are a great introduction to linear and integer programming (IP). They make for fun problems to solve and can be used to teach introductory and more advanced modelling techniques. Recent examples of integer programming formulations to solve puzzles include the snake eggs puzzle (Harris and Forbes 2023) and the fillomino puzzle (Pearce and Forbes 2017). These two examples give an introduction to some advanced techniques such as Benders decomposition, lazy constraints, and row/column generation. In common with many logic puzzles, they do not have an objective .

Chess puzzles are also a popular choice to solve with integer programming. Chess puzzles often have an objective function, which can make them more interesting and instructive. Examples include the chess avoidance puzzle, in which the aim is to place a set of chess pieces on a board so that none attack each other (Chlond 2015), and the knights exchange puzzle, in which the aim is to find the minimum number of moves to swap the black and white knights placed in the corners of a 3×3 chessboard (Iranpoor 2021).

Queens and knights stand out as the most intriguing pieces, showcasing distinctive mobility compared with other chess pieces. In this paper, chess problems involving queens will be modelled and solved using integer programming. The first of these problems is queens domination, in which the aim is to use the smallest number of queens to attack all the squares on the chessboard. The second is peaceable queens, where we find the largest size for two nonattacking armies of queens on a given chessboard.

These problems make for excellent teaching examples, as they are easy to describe and have the potential for extensions and alternative models. Our main objectives when teaching these problems are to provide interesting modelling examples, cover strategies for improving integer programming efficiency, and provide examples of how to report performance. The problems are taught in a third-year mathematics course for advanced operations research at the University of Queensland.

All problems are solved on a queens graph. For further information on graph theory, refer to West (2001). The queens graph, Q_n, corresponds to the $n \times n$ chessboard. The vertex set is each of the n^2 squares on the chessboard, and an edge exists between two vertices if a queen can move between the corresponding squares in one move. We say that a square is attacked if there is an edge between it and a queen.

The models in this paper are implemented and taught using the commercial solver Gurobi (Gurobi Optimization 2024). Gurobi is freely available to teachers and students for academic purposes. The features discussed later, including solver parameters and callbacks, specifically refer to Gurobi. A comparable solver,

CPLEX, offers similar functionality and is also freely accessible for academic use.

In each of the next two sections, we describe a specific problem: queens domination and peaceable queens. Each problem will be described in detail and then formulated as an integer program. Further alternative methods for solving the problem will then be investigated, along with other optimisation techniques. Finally, results for each problem will be compared with different approaches. We finish with some concluding remarks.

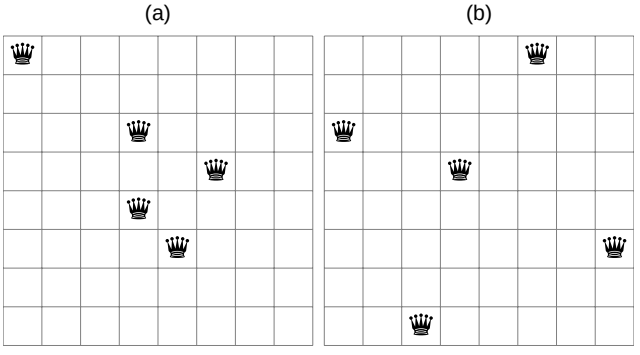
2. Queens Domination Problem

Domination puzzles on chessboards are a traditional problem that can be formulated with integer programming (Chlond and Toase 2002). The aim of these domination puzzles is to find the minimum number of pieces to put on a chessboard so that every square is attacked. Chlond (2010) looked at the knights domination problem using integer programming and a spreadsheet approach and, interestingly, considered extending the problem to a chessboard on any two-dimensional surface.

The queens domination problem is simple to understand. On an $n \times n$ chessboard, what is the minimum number of queens needed to attack every square? This is the general case, but the case where queens are independent of each other is often considered. Two solutions to the queens domination problem on the 8×8 chessboard are shown in Figure 1, with the general case on the left and the independent case on the right.

This problem is linked to dominating and independent sets in graphs. Let $G = (V, E)$ be a graph. A subset of the vertex set, $S \subseteq V$, is a dominating set if every vertex not in S is adjacent to a vertex in S. The domination number, $\gamma(G)$, is the size of the minimum dominating set of graph G. An independent set in a graph is a set of pairwise nonadjacent vertices. The size of the minimum independent dominating set of graph G is denoted $i(G)$. The minimum number of queens needed to attack every square is then the domination number. $\gamma(Q_n)$ for the general case and $i(Q_n)$ for the independent case.

Figure 1. 8×8 Chessboard with Dominating Queens



Notes. (a) General case. (b) Independent case.

This problem has been addressed before. Weakley (2018) provides an overview of work in the past 25 years. Bounds have been found on the domination number through a variety of combinatorial techniques.

The On-Line Encyclopaedia of Integer Sequences (OEIS) has a sequence for the queens domination number (A075458) and the independent domination number (A075324) (Sloane 2024). These can be seen in Tables 1 and 2.

Parra Inza et al. (2024) use integer programming to find a dominating set for any graph, not specifically the queens graph. Their formulation is used as the base formulation in this paper. They also have created heuristic and exact algorithms that solve the problem faster in some cases.

2.1. Base Formulation

Consider a size n chessboard. Let $N = \{1, \ldots, n\}$, and let S be the set of $n \times n$ squares on the chessboard. Each element in S is a pair, $(s_0, s_1), s_0 \in N, s_1 \in N$, that describes the row and column position. The adjacency matrix A has entries $A_{s,t} = 1$ if a queen can move from square s to t or $s = t$, and zero otherwise. The binary variables x_s are equal to one if a queen is placed on square $s \in S$, and zero otherwise.

The queens domination problem can be formulated as an integer programming problem as follows:

$$\text{minimise} \sum_{s \in S} x_s \tag{1}$$

subject to

$$\sum_{t \in S} A_{s,t} x_t \geq 1, \quad \forall s \in S. \tag{2}$$

The Objective Function (1) minimises the number of queens. Constraints (2) ensure that every square has at least one queen adjacent to it in the graph theoretic sense of adjacent; that is, every square is attacked or occupied by a queen.

2.2. Independence

To extend the problem to independent queens, the following constraints are added to the base formulation:

$$\sum_{s \in S \mid s_0 = i} x_s \leq 1, \qquad \forall i \in N \tag{3}$$

$$\sum_{s \in S \mid s_1 = j} x_s \leq 1, \qquad \forall j \in N \tag{4}$$

$$\sum_{s \in S \mid s_0 - s_1 = k} x_s \leq 1, \qquad \forall k \in (-N, N) \tag{5}$$

$$\sum_{s \in S \mid s_0 + s_1 = k} x_s \leq 1, \qquad \forall k \in 2(N-1). \tag{6}$$

Constraints (3) ensure that there is no more than one queen in any row. Similarly, Constraints (4) ensure that there is no more than one queen in any column,

Table 1. Domination Number

N	4	5	6	7	8	9	10	11	12	13	14	15
$\gamma(Q_n)$	2	3	3	4	5	5	5	5	6	7	8	9
N	16	17	18	19	20	21	22	23	24	25	26	
$\gamma(Q_n)$	9	9	9	10	11	11	12	12	13	13	13–14	

whereas Constraints (5) and (6) do the same for the diagonals. This is a stronger version of independence constraints, which are specific to queens on a chessboard. Independence can be ensured more generally with the following constraint:

$$\sum_{t \in S \mid A_{s,t}=1} x_t \leq M_s(1 - x_s), \quad \forall s \in S,$$

where M_s is the number of vertices adjacent to s.

2.3. Feasibility Approach

Let q be a potential domination number. In many cases of the queens domination problem, it is known that the solution may be q or $q+1$. When this is the case, a feasibility problem can be used to find whether a solution exists. From the base formulation, the objective is modified to potentially speed up this process, and constraints are added. The new objective and constraint are as follows:

$$\text{maximise} \sum_{s \in S} \left(x_s \times \sum_{t \in S} A_{s,t} \right) \tag{7}$$

subject to (2),

$$\sum_{s \in S} x_s = q. \tag{8}$$

The objective function (7) aims to maximise the number of squares that are being attacked to speed up finding a feasible solution. Constraint (8) forces the solution to have q queens. Constraints (2) are still used to ensure every square is attacked.

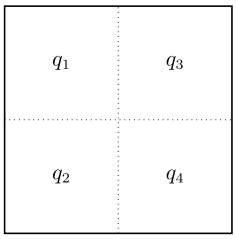
2.4. Symmetry Breaking

An approach to speed up the model is symmetry breaking, where we want to break the symmetries of rotation and reflection. This can be done by adding variables and constraints to force parts of the board to have more queens than others. Let q_1, q_2, q_3, q_4 be integer variables that denote the number of queens in quadrants 1, 2, 3 and 4, respectively. Refer to Figure 2 for a visualisation of these regions.

Table 2. Independent Domination Number

N	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$i(Q_n)$	3	3	4	4	5	5	5	5	7	7	8	9	9	9
N	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$i(Q_n)$	10	11	11	11	12	13	13	13	14	15	15	16	16	17

Figure 2. Regions of the Chessboard



Let $H1$ be the first half of N, that is, the set $\{0, 1, \ldots, \lfloor N - 1/2 \rfloor\}$, and $H2$ be the second half, that is, $\{\lfloor N - 1/2 \rfloor + 1, \ldots, N\}$. The following constraints are added to the formulation to break symmetry:

$$q_1 = \sum_{s \in S \mid s_0 \in H1, s_1 \in H1} x_s, \tag{9}$$

$$q_2 = \sum_{s \in S \mid s_0 \in H2, s_1 \in H1} x_s, \tag{10}$$

$$q_3 = \sum_{s \in S \mid s_0 \in H1, s_1 \in H2} x_s, \tag{11}$$

$$q_4 = \sum_{s \in S \mid s_0 \in H2, s_1 \in H2} x_s, \tag{12}$$

$$q_1 \leq q_2, \tag{13}$$

$$q_1 \leq q_3, \tag{14}$$

$$q_1 \leq q_4, \tag{15}$$

$$q_2 \leq q_3. \tag{16}$$

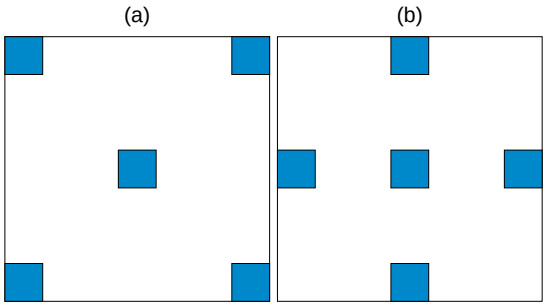
Constraints (9)–(12) count the number of queens in each specified region to populate the new integer variables. Constraints (13)–(16) break symmetry by forcing areas of the board to have at least as many queens as other areas.

2.5. Valid Inequalities

The next speedup we consider is valid inequalities—note this is only relevant for the independent case. Valid inequalities are constraints that are implied by the integer solution but may cut off solutions to linear relaxations of the problem. They potentially reduce solution time by cutting solutions earlier in the branch-and-bound tree. In this problem, if the solution has more than one queen in particular sets of squares, then the solution is cut, as it does not satisfy independence. These sets are the four corners of any-sized square within the board, plus the centre square if it exists, and the four corners of any-sized diamond within the board, plus the centre. These two sets can be seen in Figure 3.

Let the set of squares in Figure 3(a) be V_1 and the set of squares on the right be V_2. Then, we add the following constraints when there is more than one square

Figure 3. Valid Inequality Visualisation



used in one of these sets:

$$\sum_{s \in V_1} x_s \leq 1, \qquad \forall V_1 \subset S$$
$$\sum_{s \in V_2} x_s \leq 1, \qquad \forall V_2 \subset S.$$

We use the callback feature of modern solvers to do this. As an example, for $n = 8$, we find a state where squares (2,2), (5,2) and (2,5) have variable values of 0.5, 0.375 and 0.25, respectively; see Figure 4 for the solution at this state. Because these form part of a square and their values add up to greater than one, the valid inequality constraint is violated, and this solution is cut off in the callback.

2.6. Bounds and Parameter Tuning

The next speedup that was implemented was the use of lower and upper bounds. The values for lower and upper bounds are taken from Table 1. Let L be the lower bound. Then, the following constraint ensures that there are at least as many queens as the lower bound:

$$\sum_{s \in S} x_s \geq L.$$

The upper bound is used as a cutoff value, so the model will only look for solutions where the objective value is no worse than the upper bound.

Finally, we experimented with other solver parameters in Gurobi (Gurobi Optimization 2024). The parameters that could potentially make an impact were BranchDir, MIRCuts and MIPFocus. We set BranchDir = 1, forcing the solver to explore the up branch first. We set MIRCuts = 2, so the solver adds mixed-integer rounding cuts more aggressively. We set MIPFocus = 2 to force the solution strategy to focus on finding an optimal solution.

Figure 4. Valid Inequality Violation Visualisation

0	0	0	0	0	0	0	0
$\frac{1}{8}$	0	0	0	0	$\frac{3}{8}$	0	0
0	$\frac{5}{8}$	$\frac{1}{4}$	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{2}$
0	0	$\frac{1}{2}$	0	0	$\frac{3}{8}$	0	0
$\frac{3}{8}$	0	0	0	$\frac{3}{8}$	0	0	0
0	0	0	0	0	0	0	0

2.7. Student Experience

Students found this problem straightforward and easy to follow. They suggested using a big M constraint for independence but quickly grasped the tightness of Constraints (3)–(6). They also proposed several insightful efficiency improvements. In particular, they identified similar ideas for symmetry breaking (force a queen to be in the top left corner) and valid inequalities (cutoff solutions with more than one queen in a row). These suggestions provided a natural segue into discussing the symmetry-breaking techniques and valid inequalities used in our formulation. Additionally, students considered partitioning the board into smaller sections for larger instances to further reduce symmetry.

When asked to predict the most impactful technique, students expected symmetry breaking to have the greatest effect on computation time. They were surprised by the actual results (Figure 5 in the following section). We also discussed how to compare these methods, particularly in cases where performance differences between individual runs are minor. Students gained an appreciation for percentage solved versus time graphs as a useful tool for such comparisons.

2.8. Results

In this section, models with the different speedups are compared with the base model. The results in Tables 1 and 2 have been verified up to a point.

Students are often surprised to learn that integer programming solvers are chaotic. That is, even very small changes in the specification of the model, for example, reordering of constraints, can lead to completely different solution trajectories and runtimes. Therefore, to investigate the effectiveness of the different approaches

Figure 5. Percentage Solved vs. Time, Different Approaches on a 14 × 14 Chessboard for the General Case

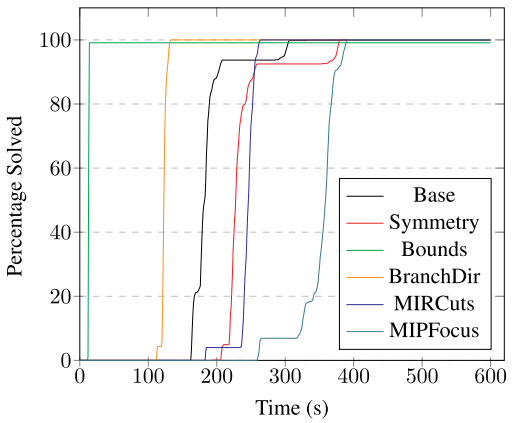


Table 3. Comparing Timing of Approaches on a 14 × 14 Chessboard for General Case

Times	Base	Symmetry	Bounds	BranchDIR	MIRCuts	MIPFocus
Minimum	198.39	403.10	9.06	148.50	248.42	267.18
Maximum	230.13	511.99	12.54	170.79	294.53	323.04
Mean	209.10	463.95	11.59	159.63	262.25	295.01
Standard deviation	9.49	21.01	0.66	5.35	13.10	14.23

and speedups, each approach needs to be tested with many different values of the solver parameter Seed, which changes the solution trajectory. This is done separately for the general and independent cases. We compare the following approaches for the general case:

- base formulation
- base + symmetry
- base + upper and lower bounds
- base + branching
- base + mixed-integer rounding (MIR) cuts
- base + mixed-integer programming (MIP) focus

The 14 × 14 chessboard was solved with 1,000 different seeds for each solution approach. The models were programmed in Python 3.11.7 via the Anaconda distribution 23.11.0 and solved using Gurobi 11.0.1 (Gurobi Optimization 2024). Jobs were run on a computing cluster with 2.5 GHz CPUs. Figure 5 shows the percentage of these 1,000 instances that are solved against runtime for each of the different approaches. We can see that the two approaches that speed up the base formulation are adding in lower and upper bounds and setting the branching direction. The graphs also demonstrate the variability in run time for the same approach when only the seed is varied. Table 3 displays the minimum, maximum, mean and standard deviation of the runtimes for the different approaches. The same process was carried out to compare the independent case with and without valid inequalities. It was determined that the valid inequalities do not speed up the program, as the independent case had an average runtime of 76 seconds compared with 398 seconds with the addition of valid inequalities. It is likely that Gurobi already implements symmetry breaking and valid inequalities on its own, so the manual addition of these is not helpful. For teaching purposes, this can be run prior to the lesson and presented in class. For smaller examples, students can divide and conquer problems using their laptops.

3. Peaceable Queens Problem

Ainley (1977) first introduced the peaceable queens problem in his book of mathematical puzzles. This appeared alongside many other chess puzzles. Bosch (1999) posed this as an integer programming problem.

The goal of this problem is to find the largest size, $m(n)$, of two equal sized armies of queens on an $n \times n$ chessboard so that queens from opposing armies cannot attack each other. A solution for the 8×8 chessboard can be seen in Figure 6. The peaceable queens problem can demonstrate the effect of different integer programming formulations of the same problem, which makes a valuable teaching example.

In graph theory, a vertex colouring on a graph is an assignment of colours to vertices in the graph with some conditions. Peaceable armies on a graph correspond to a vertex colouring so that no two adjacent vertices have different colours.

Peaceable queens is in the OEIS as A250000 (Sloane 2024) with solutions for $1 \le n \le 15$; these values can be seen in Table 4.

Smith et al. (2004) have also presented a constraint programming formulation along with a tighter version, which uses row and column variables. These ideas are used in this paper's formulation of the problem.

3.1. Formulation

We use the same notation as previous sections. Additionally, let M be the maximum number of squares that a queen can attack, namely, $4(n-1)$. The binary variables x_s are equal to one if a white queen is placed on square $s \in S$, and zero otherwise. The binary variables y_s are equal to one if a black queen is placed on square $s \in S$, and zero otherwise.

The peaceable queens problem can be formulated as an integer programming problem as follows:

$$\text{maximise} \quad \sum_{s \in S} x_s \tag{17}$$

$$\text{subject to} \quad \sum_{s \in S} x_s = \sum_{s \in S} y_s, \tag{18}$$

$$\sum_{t \in S \mid A_{s,t}=1} x_t \le M(1 - y_s), \qquad \forall s \in S \tag{19}$$

$$\sum_{t \in S \mid A_{s,t}=1} y_t \le M(1 - x_s), \qquad \forall s \in S. \tag{20}$$

Figure 6. Peaceable Queens on an 8 × 8 Chessboard

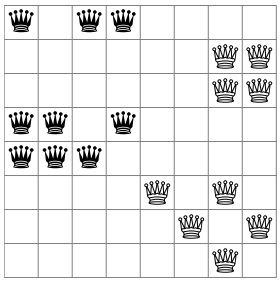


Table 4. Maximum Peaceable Army Sizes

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$m(n)$	0	0	1	2	4	5	7	9	12	14	17	21	24	28	32

The Objective Function (17) maximises the number of queens in the white army. Constraint (18) ensures that the armies are the same size. Constraints (19) and (20) ensure that black and white queens cannot attack each other.

3.2. Tighter Formulation

We can tighten the formulation considerably as follows. Add binary variables r_i, c_j, d_k, e_l, which are equal to one (zero) if the relevant row, column or diagonal can be occupied by a white (black) queen.

The peaceable queens problem can be formulated as a tighter integer programming problem as follows:

$$\text{maximise} \sum_{s\in S} x_s \tag{21}$$

$$\text{subject to} \sum_{s\in S} x_s = \sum_{s\in S} y_s, \tag{22}$$

$$x_{i,j} \le r_i, \qquad \forall i,j \in N \tag{23}$$

$$y_{i,j} \le 1 - r_i, \qquad \forall i,j \in N \tag{24}$$

$$x_{i,j} \le c_j, \qquad \forall i,j \in N \tag{25}$$

$$y_{i,j} \le 1 - c_j, \qquad \forall i,j \in N \tag{26}$$

$$x_{i,j} \le d_{i-j}, \qquad \forall i,j \in N \tag{27}$$

$$y_{i,j} \le 1 - d_{i-j}, \qquad \forall i,j \in N \tag{28}$$

$$x_{i,j} \le e_{i+j}, \qquad \forall i,j \in N \tag{29}$$

$$y_{i,j} \le 1 - e_{i+j}, \qquad \forall i,j \in N. \tag{30}$$

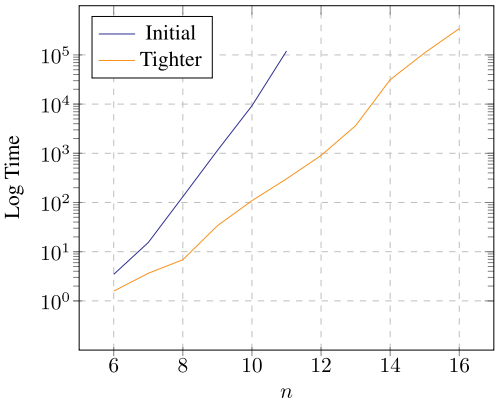
Constraints (23)–(30) ensure that each row, column or diagonal, respectively, is restricted to be occupied by either white or black.

Table 5. Peaceable Queens Timing Results

N	$m(n)$	Initial(s)	Tighter(s)
4	2	1	≤1
5	4	1	1
6	5	3	2
7	7	15	4
8	9	131	7
9	12	1,138	34
10	14	9,237	109
11	17	119,514	302
12	21	≥48 hours	902
13	24	≥48 hours	3,645
14	28	≥48 hours	31,287
15	32	≥48 hours	110,407
16	**37**	≥48 hours	3,430,623

Note. The result in bold is a new confirmed optimal solution.

Figure 7. Peaceable Queens Approaches: Log Time vs. n



3.3. Student Experience

Students understood this problem and its formulations without difficulty. After being prompted for ideas for a faster formulation, the students suggested composite variables and column generation. This was a great suggestion, particularly in following with the content of the course. However, it is not obviously applicable in this context.

Within the tighter formulation, we experimented with changing parameters such as BranchPriority, MIPFocus and Cuts. To determine if any of these changes were beneficial for efficiency, we used the Wilcoxon and Mann–Whitney U tests. The results of a Mann–Whitney U test applied to 10 different seeds for $n = 9$ showed that adding branching priority to the row and column variables was faster, with a p-value of 0.028. This demonstrated to students a meaningful way to compare timings of approaches.

3.4. Results

In this section, we present the results of the peaceable queens problem. Table 5 presents solution values $m(n)$ and their runtimes (in seconds) for $n = 4, \ldots, 16$, using the initial and tighter formulations, respectively. This can also be seen graphically in Figure 7. Note that the tighter formulation for $n = 16$ was given a time limit larger than 48 hours. It is clear that the tighter formulation outperforms the initial formulation, which is notable because the tighter formulation has a larger number of variables and constraints. For example, the 9×9 chessboard has 210 variables and 641 constraints in the tighter formulation, whereas the initial has only 162 variables and 163 constraints.

4. Conclusion

In this paper, many instances of queens domination and peaceable queens have been solved using integer programming. Initial approaches and upgrades have been stepped through in a manner suitable for educational purposes. The upgrades introduced have demonstrated improvements in computational efficiency.

These findings underscore the value of queens problems as teaching tools in optimisation. There is potential for further exploration in more graph theory and combinatorial problems.

Acknowledgments

We extend our gratitude to Lachlan W. J. McBeath for his contributions to the figures in this paper. We thank the operations research students for their valuable participation in the discussion of these problems. We also thank the editors, reviewers and everyone involved for their thoughtful consideration of our work.

References

Ainley S (1977) *Mathematical Puzzles* (G. Bell & Sons Ltd., London).

Bosch RA (1999) Peaceably coexisting armies of queens. *Optima: Math. Programming Soc. Newsletter* 62:6–9.

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

Chlond MJ (2015) Puzzle—Chess avoidance puzzles. *INFORMS Trans. Ed.* 15(3):254–256.

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

Gurobi Optimization (2024) Gurobi optimizer reference manual. Accessed July 24, 2025, https://docs.gurobi.com/projects/optimizer/en/current/index.html.

Harris M, Forbes M (2023) The snake eggs puzzle: Preparing students for Benders decomposition. *INFORMS Trans. Ed.* 23(3):210–217.

Iranpoor M (2021) Knights exchange puzzle—Teaching the efficiency of modelling. *INFORMS Trans. Ed.* 21(2):108–114.

Parra Inza E, Vakhania N, Sigarreta Almira JM, Hernández Mira FA (2024) Exact and heuristic algorithms for the domination problem. *Eur. J. Oper. Res.* 313(3):926–936.

Pearce RH, Forbes MA (2017) Puzzle—The fillomino puzzle. *INFORMS Trans. Ed.* 17(2):85–89.

Sloane NJA (2024) On-line encyclopedia of integer sequences. Accessed July 24, 2025, https://oeis.org/.

Smith BM, Petrie KE, Gent IP (2004) Models and symmetry breaking for 'peaceable armies of queens'. Régin JC, Rueher M, eds. *Integration AI OR Techniques Constraint Programming Combin. Optim, Problems. CPAIOR 2004*, Lecture Notes in Computer Science, vol. 3011 (Springer, Berlin, Heidelberg), 271–286.

Weakley WD (2018) Queens around the world in twenty-five years. Gera R, Haynes T, Hedetniemi S, eds. *Graph Theory*, Problem Books in Mathematics (Springer International Publishing, Cham, Switzerland), 43–54.

West D (2001) *Introduction to Graph Theory*, Featured Titles for Graph Theory (Prentice Hall, Hoboken, NJ).