

# Chasing First Queens by Integer Programming

Matteo Fischetti and Domenico Salvagnin

Department of Information Engineering (DEI), University of Padova  
{matteo.fischetti, domenico.salvagnin}@unipd.it

February 26, 2018

**Abstract.** The  $n$ -queens puzzle is a well-known combinatorial problem that requires to place  $n$  queens on an  $n \times n$  chessboard so that no two queens can attack each other. Since the 19th century, this problem was studied by many mathematicians and computer scientists. While finding any solution to the  $n$ -queens puzzle is rather straightforward, it is very challenging to find the lexicographically first (or smallest) feasible solution. Solutions for this type are known in the literature for  $n \leq 55$ , while for some larger chessboards only partial solutions are known. The present paper was motivated by the question of whether Integer Linear Programming (ILP) can be used to compute solutions for some open instances. We describe alternative ILP-based solution approaches, and show that they are indeed able to compute (sometimes in unexpectedly-short computing times) many new lexicographically optimal solutions for  $n$  ranging from 56 to 115.

**Keywords:**  $n$ -queens problem, mixed-integer programming, lexicographic simplex.

## 1 Introduction

The  $n$ -queens puzzle is a well-known combinatorial problem that requires to place  $n$  queens on an  $n \times n$  chessboard so that no two queens can attack each other, i.e., no two queens are on the same row, column or diagonal of the chessboard. Initially stated for the regular  $8 \times 8$  chessboard in 1848 [5], it was soon generalized to the  $n \times n$  case [17], and has attracted the interest of many mathematicians (including Carl Friedrich Gauss) and, more recently, by Edsger Dijkstra who used it to illustrate a depth-first backtracking algorithm. As a decision problem, the  $n$ -queens puzzle is rather trivial, as a solution exists for all  $n > 3$ , and there are closed formulas to compute such solutions; see, e.g., the survey in [4]. On the other hand, the counting version of the problem, i.e., to determine the number of different ways to put  $n$  queens on a  $n \times n$  chessboard turns out to be extremely challenging. The sequence, labelled A000170 on the Online Encyclopedia of Integer Sequences (OEIS) [20], is currently known only up to  $n = 27$ . The related problem of finding all solutions to the problem was shown in [14] to be beyond the #P-class.

Another variant of the problem, which is somewhat related to the one addressed in this paper, is the  $n$ -queens completion problem, in which some queens are already placed on the chessboard and the solver is required to place the remaining ones, or show that it is not possible. The  $n$ -queens completion problem is both NP-complete and #P-complete, as proved in [10].

Following a suggestion of Donald Knuth [16], in this paper we study another very challenging version of the  $n$ -queens problem, namely, finding the lexicographically-first (or smallest) feasible solution. This is sequence A141843 on OEIS. Solutions for this variant are known only for  $n \leq 55$  [19], while for some larger chessboards only partial solutions are known.

It is worth noting that the lexicographically optimal solution is known for the case of a chessboard of infinite size. Indeed, such a sequence can be easily computed by a simple greedy algorithm that iterates over the anti-diagonals of the chessboard and places a queen in each anti-diagonal in the first available position (this is sequence A065188 on OEIS). Interestingly, as the size of the chessboard increases, its lexicographically optimal solution overlaps more and more with this greedy sequence.

The outline of the paper is as follows. In Section 2 we describe the basic Integer Linear Programming (ILP) formulation for the  $n$ -queens model, as well as potential families of valid inequalities. In Section 3 we describe the different methods developed to solve the instances to lexicographic optimality, while computational results are given in Section 4. Conclusions and future directions of research are drawn in Section 5. Finally, we list in Appendix all the new optimal solutions we found for  $n$  ranging from 56 to 115.

## 2 An ILP model

A basic ILP model for the  $n$ -queens problem can be obtained by introducing the binary variables  $x_{ij} = 1$  iff a queen is placed in row  $i$  and column  $j$  of the chessboard, for each  $i, j = 1, \dots, n$ . Constraints in the basic model stipulate that (i) there is exactly one  $x_{ij} = 1$  in each row  $i$ ; (ii) there is exactly one  $x_{ij} = 1$  in each column  $j$ ; and (iii) there is at most one  $x_{ij} = 1$  in each diagonal of the chessboard. Note that all such constraints are *clique* constraints.

In principle, it would be possible to encode the (row-wise) lexicographically minimum requirement by just adding the objective function:

$$\sum_{i=1}^n \sum_{j=1}^n 2^{ni+j} x_{ij} \tag{1}$$

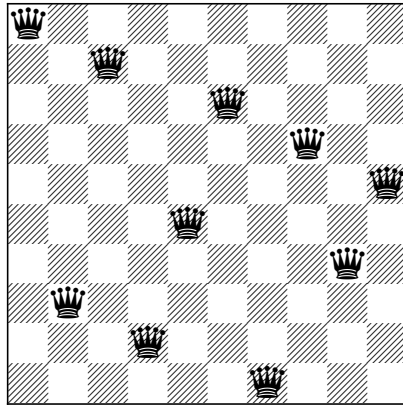
and solve the problem with a black-box ILP solver. However, the size of the coefficients makes such a method practical only for the smallest chessboards. Still, this simple model, without the objective (1), is the basis of all the methods that will be discussed in Section 3.

A compact way to represent a feasible solution is to use a permutation  $\pi = (\pi_1, \dots, \pi_n)$  of the integers  $1, \dots, n$  defined as follows:

$$\pi_i := \sum_{j=1}^n j x_{ij}, \quad i = 1, \dots, n. \quad (2)$$

Among all permutations  $\pi$  that correspond to a feasible  $x$ , we then look for the lexicographically smallest one. For example, the lex-optimal solution for  $n = 10$ , depicted in Figure 1, can be described as

$$(1, 3, 6, 8, 10, 5, 9, 2, 4, 7).$$



**Fig. 1.** Lexicographically optimal solution for  $n = 10$ .

The  $n$ -queens problem can also be easily reformulated as a maximum independent set problem, as noted for example in [8]. Indeed, one just needs to construct a graph in which there is a node for each square of the chessboard and an edge for each pair of conflicting squares, i.e., for any two squares in the same row, column or diagonal. Then any independent set of cardinality  $n$  is a solution to the puzzle. The independent set reformulation immediately suggests classes of valid inequalities for the  $n$ -queens problem, namely all that are valid for the stable set polytope, such as *clique* and *odd-cycle* [13] inequalities.

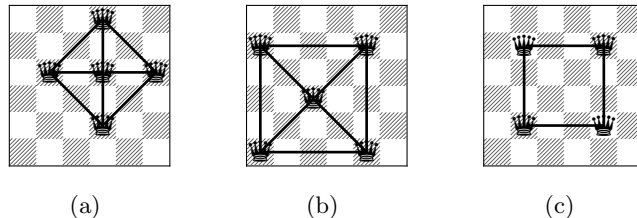
Among clique inequalities, the following (polynomial in  $n$ ) family is particularly relevant for our problem:

$$x_{ij} + x_{i,j+h} + x_{i+h,j} + x_{i-h,j} + x_{i,j-h} \leq 1 \quad (3)$$

$$x_{ij} + x_{i+h,j+h} + x_{i-h,j+h} + x_{i-h,j-h} + x_{i+h,j-h} \leq 1 \quad (4)$$

$$x_{ij} + x_{i+h,j} + x_{i+h,j+h} + x_{i,j+h} \leq 1 \quad (5)$$

where  $i, j, h \in \{1, \dots, n\}$ ; of course, variables  $x_{uv}$  corresponding to a position  $(u, v)$  outside the  $n \times n$  chessboard are removed from the summations. The three different types of cliques in this family are depicted in Figure 2.



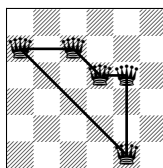
**Fig. 2.** Three different families of clique cuts for  $n$ -queens.

Clique inequalities (3)–(5) can be trivially separated in time that is polynomial in  $n$ . In addition, in preliminary experiments we implemented a general-purpose exact clique separator based on the solution of an auxiliary ILP model, and it never produced any additional violated clique inequality for the instances in our testbed.

A second class of inequalities contains the so-called *odd-cycle* inequalities. Given any odd cycle  $O$  in the graph, the following inequality:

$$\sum_{k \in O} x_k \leq \frac{|O| - 1}{2} \quad (6)$$

is valid for the stable set polytope. Odd-cycle inequalities can be easily separated as  $\{0, 1/2\}$ -cuts with the combinatorial procedures described in [6,7,2]. An example of odd-cycle inequality occurring in the  $n$ -queens problem is illustrated in Figure 3.



**Fig. 3.** Example of odd-cycle inequality for  $n$ -queens: no more than two of the five positions can be occupied by a queen.

### 3 Solution methods

We next describe the solution algorithms that we implemented.

#### 3.1 Using a Constraint Programming solver

The  $n$ -queens puzzle can be easily modeled as a Constraint Programming (CP) problem. Indeed, working directly on the variables  $\pi_i$ , the puzzle can be formulated by just three `alldifferent` [18,21] global constraints:

$$\text{alldifferent}(\pi_i, i = 1, \dots, n) \tag{7}$$

$$\text{alldifferent}(\pi_i + i, i = 1, \dots, n) \tag{8}$$

$$\text{alldifferent}(\pi_i - i, i = 1, \dots, n). \tag{9}$$

We implemented the model above with Gecode [9]. In order to enforce the model to find the lexicographically-smallest solution, we use Depth-First Search (DFS) as search strategy, always branching on the first unfixed variable  $\pi_i$  and picking values in increasing order—in Gecode terminology, that amounts to using a *brancher* specified by `INT_VAR_NONE()` and `INT_VALUES_MIN()`. In the following, we will refer to this solution method as CP.

#### 3.2 Using an exact ILP solver

A simple algorithm to compute the lex-optimal solution by iteratively using a black-box ILP solver is as follows: We scan all the chessboard positions  $(i, j)$  in lexicographical order, i.e., row by row. For each  $(i, j)$ , we are given the queens already positioned in the previous iterations (i.e., we have a number of fixed  $x$  variables), and our order of business is to decide whether a queen can be placed in  $(i, j)$  or not. This in turn requires solving the basic ILP model with some variables fixed in the previous iterations, by *maximizing*  $x_{ij}$ : if the final optimal solution has value 1, we place a new queen in position  $(i, j)$  by fixing  $x_{ij} = 1$ , otherwise we fix  $x_{ij} = 0$  and proceed with the next chessboard position<sup>1</sup>. This approach requires solving  $n^2$  ILPs.

In our actual implementation, a more effective scheme is used that exploits representation (2). To be specific, we scan the rows  $i = 1, \dots, n$ , in sequence. For each  $i$ , we have already fixed in the previous iterations the lex-optimal sequence  $\pi_1, \dots, \pi_{i-1}$  and the corresponding  $x$  variables, and we want to compute the *smallest* feasible integer  $\pi_i$ . To this end we solve the basic ILP model, with some variables fixed in the previous iterations, by *minimizing* the objective function (2), fix all the  $x_{ij}$  variables in row  $i$  accordingly, and proceed with the next row. In this way, only  $n$  ILPs need to be solved. In the following, we will refer to this solution method as ILP-ITER.

---

<sup>1</sup> Alternatively, one could fix  $x_{ij} = 1$ , check the resulting model for feasibility, and then move to the next position.

### 3.3 Using a truncated ILP solver

We also implemented an explicit depth-first backtracking algorithm to build the lex-optimal permutation  $\pi$ , very much in the spirit of the CP approach described in Subsection 3.1. At each iteration (i.e., at each node of the branching tree) we have tentatively fixed a lex-minimal, but possibly infeasible sequence,  $(\bar{\pi}_1, \dots, \bar{\pi}_{i-1})$  and the corresponding  $x$  variables, and we have to decide the next value in position  $i$ . This is in turn obtained by solving a *relaxation* of the current ILP with objective function (2), to be minimized, i.e., by applying the following three steps:

- i) invoke the ILP solver (with its default cutting-plane generation and preprocessing) for a limited number of nodes, say  $NN$ ;
- ii) define  $\bar{\pi}_i$  as the best *lower bound* available at the node limit (rounded up);
- iii) tentatively fix  $\bar{\pi}_i$ , along with the corresponding  $x$  variables, as the  $i$ -th value in the sequence.

As a lower bound (instead of the true value) is used, it may happen that, at a later iteration, the current ILP becomes infeasible, proving that the current tentative subsequence  $(\bar{\pi}_1, \dots, \bar{\pi}_k)$  till position  $k$  (say) is infeasible as well. In this case, a backtracking operation takes place, that consists in imposing that the  $k$ -th position must hold a value strictly larger than  $\bar{\pi}_k$ . The latter requirement can easily be enforced in the ILP model by setting  $x_{kj} = 0$  for  $j = 1, \dots, \bar{\pi}_k$ . The algorithm ends as soon as the first feasible complete permutation  $(\bar{\pi}_1, \dots, \bar{\pi}_n)$  is found.

After some preliminary tests, we decided to set  $NN = 0$ , i.e., to only solve the root node of the ILP at hand. Note that this is *not* equivalent to solving the LP relaxation of the ILP, as cutting planes and (most importantly) preprocessing play a crucial role here. According to our computational experience, solving just the LP relaxation is indeed mathematically correct and very fast, as the dual simplex can be used to reoptimize each LP, but the number of backtrackings becomes too large to have a competitive implementation. In the following, we will refer to this solution method as ILP-TRUNC.

### 3.4 An enumerative method based on lexicographic simplex

Finally, given the strong lexicographic nature of the problem at hand, we decided to implement a custom enumerative algorithm based on the lexicographic simplex method [11,12]. The lexicographic simplex method not only finds an optimal solution to a given LP, but it guarantees to return the lexicographically smallest (or greatest) one among all optimal solutions. The lexicographic variant of the simplex method can be implemented quite easily on top of a black-box regular simplex solver, as described for example in [3,22]. The idea is as follows. Given an ordered sequence of objective functions  $f_k$  to optimize lexicographically, at each step we impose to stay on the optimal face of the current objective by fixing all variables (including the artificial variables associated to inequality

constraints) with nonzero reduced cost, move to the next objective and reoptimize. Once all objectives have been optimized, in sequence, the original bounds for all variables are restored, which does not change the optimality status of the final basis, which is the lex-optimal one.

In our  $n$ -queens case, given our encoding of the permutation variables  $\pi$  as  $x_{ij}$ , we are interested in the lexicographically maximal solution in the  $x$  space or, equivalently, the sequence of objective functions to be minimized is  $-x_{ij}$ , for all  $i, j = 1, \dots, n$ .

Using a lexicographic simplex method within an enumerative DFS scheme, in which again we always branch on the first unfixed variable and explore the 1-branch first, provides the following advantages over using a “regular” simplex method:

- Whenever the LP relaxation turns out to be integer, i.e., there are no fractional variables, we are guaranteed that this is the lex-optimal integer solution within the current subtree, hence we can prune the node. Given our branching and exploration strategy, this also implies that we are done.
- If the first unfixed variable at the current node gets a value strictly less than one, then we can fix the variable to zero. This is easily proved using the lex-optimality of the LP solution as an argument. Being the first unfixed variable, this is the first objective to be considered by the lexicographic simplex at the current node, so a lex-optimal value  $< 1$  means that there is no feasible solution (in the current subtree) in which this variable takes value 1. Note that this reduction can be applied iteratively until the first unfixed variable gets a value of 1. We call this process *mini-cutloop*.

The basic scheme above can be improved with some additional modifications. First of all, we do not need to branch on single variables but we can branch directly on rows, again always picking the first row that contains an unfixed variable. For example, let the first unfixed variable be  $x_{ij}$ : instead of branching on the binary dichotomy  $x_{ij} = 1 \vee x_{ij} = 0$ , we use the  $n$ -way branching  $x_{i1} = 1 \vee x_{i2} = 1 \vee \dots \vee x_{in} = 1$ . Of course, variables that are already fixed are removed from the list. This basically mimics the branching that would have been done by working directly with the  $\pi$  variables, as done by the CP solver.

Note that, because of our rigid branching strategy, there is no need for a full lexicographic optimization at each node. Indeed, for the purpose of branching, we can stop the lexicographic optimization at the first fractional variable, as we will be forced to branch on its row, or on a previous one. For this very reason, and because of the  $n$ -queens structure, we implemented a specialized lexicographic simplex method, where instead of optimizing one variable at the time, we optimize row by row, also integrating the mini-cutloop in the process. In particular, we do the following:

1. Let  $i^*$  be the first row with an unfixed variable. Set the objective function to  $\sum_{j=1}^n jx_{i^*j}$  and minimize it.
2. Apply the mini-cutloop, by iteratively fixing the first unfixed variable in the row if its fractional value is  $< 1$  and by reoptimizing with the dual simplex.

3. If all variables in the current row are fixed this way, then we can move to the next row and go to step (1). Otherwise stop.

Note that the method above does not need to temporarily fix variables as the regular lexicographic simplex would. It is also important to note that, in the loop above, if the current fractional solution is integer, we are no longer guaranteed that this is the lexicographically optimal solution. In this (rare) case, we resort to a full-blown lexicographic simplex method to tell whether we can prune the node or need to branch.

The effectiveness of the node processing above greatly depends on the minicutloop, which in turn relies on being able to recognize fixed variables, i.e., to distinguish between a variable that happens to be zero or one in the current fractional solution, and a variable that is actually fixed at that value in the current node. For this purpose, we implemented a specialized propagator for the clique constraints of the basic model—while there is no need to propagate the clique constraints (3)–(5) as those can never lead to additional fixings.

Finally, separation of the clique inequalities (3)–(5) and odd-cycle inequalities has also been implemented and added to the node processing code. In the following, we will refer to this solution method as **LEX-DFS**.

## 4 Computational comparisons

We implemented our ILP models with the MIP solver **IBM ILOG CPLEX 12.7.1** [15], while we used **Gecode 5.1.0** [9] as the CP solver for model (7)–(9). All experiments were done on a cluster of 24 identical machines, each equipped with an Intel Xeon E3-1220 V2 quad-core PC and 16GB of RAM.

The testbed is made of all instances with  $n$  ranging from 21 to 60. A time limit of 2 days was given for each instance to each method. Detailed results are given in Table 2, where we report the running time, in seconds, for all of our methods. The last two rows of the table report the shifted geometric mean [1] of the computing time (with a shift of 10 sec.s) and the number of solved instances. According to the table, the CP model is able to solve models up to size 40 in a reasonable amount of time, after which it can no longer solve any model. Comparing with the numbers reported in [19], this can be already considered a good achievement, and a testament to how efficient Gecode’s implementation is. On the other hand, all methods based on ILP, while initially slower, turn out to be able to solve almost all models in the testbed. Among the ILP methods, **ILP-ITER**, while being the easiest to implement, is also the slowest method, while **ILP-TRUNC** and **LEX-DFS** are the fastest methods, with very similar average running times.

As already noted in [19], the size of the chessboard is not a direct indicator of instance difficulty, as some bigger chessboards can be solved significantly faster than smaller ones. This is true in particular for ILP-based methods, where for example  $n = 48$  is unsolved while  $n = 49$  can be cracked in a few seconds. Interestingly, chessboards with even  $n$  seem to be consistently harder than the ones with odd  $n$ .



As for the advanced techniques implemented in LEX-DFS, we have to admit that for some of them the overall effect was rather disappointing. In particular, the separation of clique and odd-cycle inequalities, while able to reduce the number of enumerated nodes by more than a factor of 2, does not lead to a faster algorithm overall. To the contrary, disabling cut separation leads to a slightly faster method with an average runtime of 246 sec.s. Note that this is not due to the complexity of separating cuts, separation being extremely fast for both classes of inequalities, but rather for the reduced node throughput.

## 5 Conclusions and future directions of work

Finding a lexicographically minimal (also called “first”) solution of the  $n$ -queens puzzle is a very difficult problem that attracted some research interest in recent years. Following a suggestion by Donald E. Knuth, we have developed new solution methods based on Integer Linear Programming, and have been able to provide the optimal solution for several open problems.

The two main outcomes of our research are as follows: (1) ILP has been able to solve many previously unsolved models for this problem, sometimes in unexpectedly-short computing times; (2) the yet-unsolved cases provide excellent benchmark examples on which to base the next advances in ILP technology. In addition, we think that improving our understanding on how to solve lexicographic variants of combinatorial problems is an interesting topic on its own.

Future research should address the unsolved cases, and in particular should try to better understand the reason why, in the ILP setting, the instances with even  $n$  seem to be much more difficult to solve than those with  $n$  odd.

## Acknowledgements

This research was partially supported by MiUR, Italy, through project PRIN2015 “Nonlinear and Combinatorial Aspects of Complex Networks”. We thank Donald E. Knuth for having pointed out the problem to us, and for inspiring discussions on the role of Integer Linear Programming in solving combinatorial problems arising in digital tomography.

## A New solutions

Here are the solutions we found for some open problems from the literature:

$n$	Solution
56	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 33 42 44 46 43 51 53 55 45 54 50 47 56 48 52 49 12 14 23 21 32 34 26 16 30 17 24 18 37 28 40 20 39 41 35 38 36

(continued on next page)

$n$	Solution
57	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 34 43 45 47 50 52 54 44 57 49 46 56 51 48 55 53 14 28 17 33 23 16 18 30 24 37 20 32 21 26 40 35 41 39 42 36 38
58	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 42 45 48 52 54 43 53 55 49 44 46 50 57 47 51 58 56 28 26 20 34 30 18 14 17 24 21 16 35 23 40 33 36 38 32 41 39 37
59	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 34 36 45 47 49 52 56 53 46 57 59 48 51 54 50 55 58 16 14 17 32 23 26 20 18 33 35 28 21 43 41 37 24 40 44 30 39 42 38
60	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 34 44 46 48 45 51 54 58 50 59 57 60 47 49 52 55 53 56 18 33 23 32 28 16 20 17 21 37 35 26 24 30 14 42 38 43 41 39 36 40
61	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 45 47 49 52 54 56 50 60 46 61 58 48 51 53 55 57 59 23 32 16 33 21 17 26 36 18 20 38 24 28 34 40 30 41 44 42 37 39 43
63	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 47 49 51 53 59 57 52 60 62 48 50 54 63 55 58 56 61 32 16 33 17 21 26 36 20 18 38 28 23 40 24 30 34 41 39 44 46 43 45 42
65	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 39 49 51 53 50 56 59 63 55 64 62 65 52 54 57 60 58 61 16 30 17 21 26 36 33 20 18 41 38 23 32 24 28 48 46 34 43 40 44 47 45 42
67	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 39 41 51 53 55 52 58 61 65 57 66 64 67 54 56 59 62 60 63 16 18 34 30 38 20 24 17 21 23 43 32 40 33 36 26 28 46 48 50 44 47 45 42 49
69	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 39 41 43 53 55 57 54 60 63 67 59 68 66 69 56 58 61 64 62 65 17 20 16 30 24 33 40 38 18 21 34 26 23 42 49 28 32 50 36 51 46 44 52 48 45 47
71	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 39 41 16 53 55 57 54 56 62 68 66 69 59 70 67 58 71 61 64 60 65 63 21 30 17 40 18 24 36 20 42 44 26 34 23 33 38 32 28 49 51 45 47 52 50 48 46 43
73	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 39 41 16 44 55 57 59 56 58 63 67 69 71 73 61 70 72 65 60 62 64 66 68 20 34 21 18 42 17 38 24 43 23 28 45 33 40 36 26 32 30 54 47 50 52 46 48 53 51 49
77	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 39 41 16 18 45 57 59 61 58 60 65 68 72 74 76 73 75 63 67 64 62 77 70 66 71 69 38 40 28 17 21 24 26 20 43 46 42 23 36 34 32 30 44 33 52 55 47 50 53 56 54 48 51 49

(continued on next page)

$n$	Solution
79	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 39 41 16 18 45 47 59 61 63 60 62 67 70 74 71 77 79 76 78 64 68 65 69 66 73 75 72 20 38 17 21 44 24 30 23 46 48 36 42 40 34 26 28 33 50 32 53 43 57 52 58 56 54 51 49 55
85	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 39 41 16 18 45 17 48 50 63 65 67 64 66 71 73 75 80 82 84 81 83 72 70 68 85 69 78 74 77 79 76 20 23 43 24 21 49 44 42 34 46 28 30 52 26 38 51 32 40 33 61 47 60 36 53 58 54 57 59 56 62 55
91	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 39 41 16 18 45 17 48 20 51 53 67 69 71 68 70 75 77 79 81 85 87 90 86 91 89 72 74 76 73 80 82 84 78 83 88 21 34 26 49 46 24 47 52 43 23 30 33 55 28 42 32 54 40 36 44 64 50 38 59 61 65 57 66 60 63 56 58 62
93	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 39 41 16 18 45 17 48 20 51 53 55 69 71 73 70 72 77 79 81 83 87 89 92 88 93 91 74 76 78 75 82 84 86 80 85 90 24 21 23 46 49 47 52 38 30 56 33 26 28 43 32 54 57 42 44 36 34 40 50 61 68 65 62 59 63 58 67 64 66 60
97	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 39 41 16 18 45 17 48 20 51 53 21 56 71 73 75 72 74 79 81 83 85 87 89 93 95 97 94 96 76 80 77 86 78 82 84 91 88 90 92 46 24 28 52 23 49 47 34 30 26 57 50 33 61 42 44 36 32 55 43 38 54 60 66 40 70 68 63 58 69 62 65 67 64 59
101	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 39 41 16 18 45 17 48 20 51 53 21 56 58 60 75 77 79 76 78 83 85 87 89 91 93 97 99 101 98 100 80 84 81 90 82 86 88 95 92 94 96 23 26 28 40 43 54 57 24 32 47 50 42 59 33 30 34 52 62 68 46 38 36 44 55 66 71 74 70 49 73 63 72 67 61 64 69 65
103	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 39 41 16 18 45 17 48 20 51 53 21 56 58 60 62 77 79 81 78 80 85 87 89 91 93 95 99 101 103 100 102 82 86 83 92 84 88 90 97 94 96 98 23 26 24 30 28 36 46 55 59 52 54 44 61 34 66 33 42 32 47 49 40 38 57 73 71 63 72 43 64 70 75 50 69 67 76 74 68 65
109	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 39 41 16 18 45 17 48 20 51 53 21 56 58 60 23 63 65 81 83 85 82 84 89 91 93 95 86 100 104 106 101 109 107 105 108 88 92 87 96 90 97 102 94 98 103 99 26 24 32 28 36 55 57 40 64 61 54 50 30 66 34 42 38 33 49 43 67 59 62 77 52 44 47 75 71 46 76 80 73 70 79 69 78 72 74 68

(continued on next page)

$n$	Solution
115	1 3 5 2 4 9 11 13 15 6 8 19 7 22 10 25 27 29 31 12 14 35 37 39 41 16 18 45 17 48 20 51 53 21 56 58 60 23 63 24 66 68 85 87 89 86 88 93 95 97 99 90 102 108 111 113 107 109 112 115 91 114 98 101 92 94 96 100 105 103 110 106 104 26 28 30 32 36 50 59 62 64 55 43 34 72 67 52 33 40 65 57 44 42 38 74 54 61 46 83 47 77 69 49 82 79 75 84 71 80 78 81 73 70 76

## References

1. Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007.
2. Giuseppe Andreello, Alberto Caprara, and Matteo Fischetti. Embedding  $\{0, 1/2\}$ -cuts in a branch-and-cut framework: A computational study. *INFORMS Journal on Computing*, 19(2):229–238, 2007.
3. Egon Balas, Matteo Fischetti, and Arrigo Zanette. On the enumerative nature of Gomory’s dual cutting plane method. *Mathematical Programming*, 125:325–351, 2010.
4. Jordan Bell and Brett Stevens. A survey of known results and research areas for  $n$ -queens. *Discrete Mathematics*, 309(1):1–31, 2009.
5. Max Bezzel. Proposal of 8-queens problem. *Berliner Schachzeitung*, 3:363, 1848.
6. Alberto Caprara and Matteo Fischetti.  $\{0, \frac{1}{2}\}$ -Chvátal-Gomory cuts. *Mathematical Programming*, 74:221–235, 1996.
7. Alberto Caprara and Matteo Fischetti. Odd cut-sets, odd cycles, and 0-1/2 Chvatal-Gomory cuts. *Ricerca Operativa*, 26:51–80, 1996.
8. L. R. Foulds and D. G. Johnston. An application of graph theory and integer programming: Chessboard non-attacking puzzles. *Mathematics Magazine*, 57:95–104, 1984.
9. Gecode Team. Gecode: Generic constraint development environment, 2017. Available at <http://www.gecode.org>.
10. Ian P. Gent, Christopher Jefferson, and Peter Nightingale. Complexity of  $n$ -queens completion. *Journal of Artificial Intelligence Research*, 59:815–848, 2017.
11. Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.
12. Ralph E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597, The RAND Cooperation, 1960.
13. Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.
14. Jieh Hsiang, D. Frank Hsu, and Yuh-Pyng Shieh. On the hardness of counting problems of complete mappings. *Discrete Mathematics*, 277(1-3):87–100, 2004.
15. IBM. ILOG CPLEX 12.7 User’s Manual, 2017.
16. Donald E. Knuth. Private communication, November 2017.
17. François J.E. Lionnet. Question 963. *Nouvelles Annales de Mathématiques*, 8:560, 1869.

$n$	methods			
	CP	ILP-ITER	ILP-TRUNC	LEX-DFS
21	0.01	0.30	0.45	0.08
22	0.95	1.63	16.67	9.20
23	0.02	0.40	0.60	0.11
24	0.20	0.60	2.95	0.82
25	0.03	0.49	0.79	0.12
26	0.16	0.84	1.59	0.42
27	0.17	0.59	0.90	0.09
28	0.84	1.13	2.08	1.06
29	0.39	1.05	1.36	0.32
30	15.80	13.16	77.05	16.35
31	2.86	1.50	3.31	0.87
32	19.45	4.97	42.73	5.23
33	29.82	28.47	56.76	13.83
34	593.60	342.32	4558.02	228.07
35	33.70	11.21	30.46	4.67
36	5199.27	1882.10	20901.43	1196.59
37	185.37	2.06	7.49	0.54
38	2485.20	101.30	151.86	130.16
39	1642.30	143.02	184.50	44.79
40	<i>t.l.</i>	9604.18	117591.20	7068.84
41	1543.84	20.91	105.47	5.69
42	<i>t.l.</i>	<i>t.l.</i>	<i>t.l.</i>	<i>t.l.</i>
43	23528.50	21.65	162.13	6.08
44	<i>t.l.</i>	1013.43	14838.95	2220.52
45	<i>t.l.</i>	3604.37	4560.69	1388.93
46	<i>t.l.</i>	<i>t.l.</i>	<i>t.l.</i>	<i>t.l.</i>
47	<i>t.l.</i>	1602.10	5057.63	601.54
48	<i>t.l.</i>	<i>t.l.</i>	<i>t.l.</i>	<i>t.l.</i>
49	<i>t.l.</i>	23.26	460.07	10.35
50	<i>t.l.</i>	28011.44	<i>t.l.</i>	61679.70
51	<i>t.l.</i>	4.63	874.16	0.88
52	<i>t.l.</i>	30306.67	27701.60	75659.40
53	<i>t.l.</i>	5.05	285.65	0.96
54	<i>t.l.</i>	64.09	19784.91	67031.40
55	<i>t.l.</i>	44.50	569.58	18.42
56	<i>t.l.</i>	28026.57	13386.85	101936.00
57	<i>t.l.</i>	10.03	3961.54	5.87
58	<i>t.l.</i>	<i>t.l.</i>	129596.69	<i>t.l.</i>
59	<i>t.l.</i>	49647.30	<i>t.l.</i>	18795.70
60	<i>t.l.</i>	<i>t.l.</i>	39143.49	<i>t.l.</i>
shmean	2945.76	780.20	251.85	263.79
#solved	21	35	35	35

**Table 2.** Comparison of different methods for  $n = 21, \dots, 60$ , with a time limit of 172800 sec.s (2 days).

18. Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Artificial Intelligence*, volume 1, pages 362–367, 1994.
19. Wolfram Schubert. Wolfram Schubert’s N-Queens page, <http://m29s20.vlinux.de/~wschub/nqueen.html>, accessed on December 2017.
20. Neil J.A. Sloane. The on-line encyclopedia of integer sequences, 2017.
21. Willem Jan van Hoeve. The alldifferent constraint: A survey. *CoRR*, 2001.
22. Arrigo Zanette, Matteo Fischetti, and Egon Balas. Lexicography and degeneracy: can a pure cutting plane algorithm work? *Mathematical Programming*, 130:153–176, 2011.