Course: Seminar on Computer Science III
(4810-1206)
Instructors: CODOGNET Philippe,
SUPPAKITPAISARN Vorapong
Semester: A1A2 2025

Comparison of Three Computational Models using the N-Queens Problem as Benchmark

Oliver Odhe

Human and Engineered Environmental Studies

Graduate School of Frontier Sciences

The University of Tokyo

January 30, 2026

Abstract

The N-Queens problem is a well-known benchmark in combinatorial optimisation and constraint programming. This report compares three different ways of modelling and solving the problem: an integer constraint programming model using `alldifferent` constraints, a Boolean constraint programming model with pseudo-Boolean constraints, and a QUBO model solved using an annealing-based method. The models are implemented and tested on problem instances of increasing size. The results show clear differences in performance and scalability, highlighting how modelling choices and solver paradigms affect runtime behaviour and solution reliability.

1 Introduction

The ”N-Queens problem” is a typical benchmark problem in artificial intelligence (AI) and combinatorial optimisation. It consists of placing N queens on an $N \times N$ chessboard such that no two queens attack each other, meaning that no two queens share the same row, column, or diagonal. Although the problem may appear relatively straightforward, it has deep internal relationships, symmetries, and patterns, making it suitable for modelling and optimisation in computer science.

From a computational perspective, the N-Queens problem is well suited to illustrating the strengths and weaknesses of different modelling choices. It can be expressed naturally as a constraint satisfaction problem (CSP), where feasibility is determined solely by the satisfaction of constraints, but it can also be defined as an unconstrained optimisation problem through the use of penalty functions. As such, it provides an ideal test case for comparing exact constraint-based methods with alternative optimisation approaches such as quadratic unconstrained binary optimisation (QUBO) and annealing-based solvers.

The aim of this report is to study and compare three different modelling and solving approaches for the N-Queens problem. First, an integer constraint programming (CP) model is using global `alldifferent` constraints is considered as baseline, reflecting a compact and propagation-efficient model. Second, a Boolean CP model is introduced, using N^2 binary variables and pseudo-Boolean constraints, which represents the same problem but with a significantly different modelling structure. Finally, the problem is defined as a QUBO model and solved using an annealing-based solver via the Fixstars Amplify framework, providing a penalty-driven, stochastic optimisation approach.

By implementing and experimentally evaluating these three approaches, this report seeks to highlight the impact of modelling choices on solver performance, scalability, and solution reliability. In particular, it investigates how problem structure, variable representation, and solver paradigms influence runtime behaviour and solution quality as the problem size increases. Through this comparison, the report aims to provide practical insights into the trade-offs between expressive constraint models and flexible optimisation-based models for combinatorial problems.

2 Models and Methods

This section describes the modelling approaches and solution methods use to solve the N-Queens problem. Three unique models are considered: an integer CP model, a Boolean CP model, and a QUBO model solved using an annealing-based approach. Although all three models represent the same underlying problem, they differ significantly in variable representation, constraint encoding, and solving techniques.

2.1 Problem Definition

The N-Queens problem consists of placing N queens on an $N \times N$ chessboard such that no two queens attack each other. A queen attacks another queen if they are located on the same row, the same column, or the same diagonal. A valid solution must therefore satisfy the following constraints:

- exactly one queen is placed in each row;
- exactly one queen is placed in each column;
- at most one queen is placed on each diagonal (in both directions).

The goal of the problem in this report is to find any configuration that satisfies all constraints. Furthermore, no optimisation objective beyond feasibility is required.

2.2 Integer CP Model

The first model is an integer CP formulation, which serves as a baseline approach. In this model, the chessboard is represented using a single array of integer decision variables. For each row $r \in \{1,...,N\}$, a variable q_r denotes the column index in which the queen is placed in that row.

To enforce that no two queens share the same column, a global `alldifferent` constraint is imposed on the array of variables $\{q_1, q_2, ..., q_N\}$. Diagonal conflicts are prevented by applying additional `alldifferent` constraints to the expressions $q_r + r$ and $q_r - r$, which uniquely identify the two diagonal directions. These constraints ensure that no two queens are located on the same diagonal.

This model is compact, using only N variables, and relies on strong global constraints that enable efficient constraint propagation. The model is implemented in MiniZinc as described Listing 1 and solved using the Gecode solver.

Listing 1: MiniZinc code implementation of the integer CP model

```
int: n;
array[1..n] of var 1..n: q;

include "alldifferent.mzn";

constraint alldifferent(q);
constraint alldifferent([ q[r] + r | r in 1..n ]);
constraint alldifferent([ q[r] - r | r in 1..n ]);

solve satisfy;
```

```
output [
  "n=", show(n), "\n",
  "q=", show(q), "\n"
];
```

2.3 Boolean CP Model

The second model approaches the problem using Boolean decision variables. In this approach, a binary variable $x_{r,c} \in \{0,1\}$ is introduced for each cell of the chessboard, where $x_{r,c} = 1$ indicates that a queen is placed at row r, column c, and $x_{r,c} = 0$ otherwise. This results in N^2 decision variables.

The constraints are expressed using pseudo-Boolean linear expressions. For each row r, the sum of variables in that row is constrained to equal one, ensuring that exactly one queen is placed in each row. Similarly, for each column c, the sum of variables in that column is constrained to equal one. Diagonal constraints are enforced by requiring that, for each diagonal, the sum of the variables on that diagonal is less than or equal to one, which prevents multiple queens from occupying the same diagonal.

Although this model represents the same problem as the integer CP model, it differs significantly in size and structure. The Boolean model involves a much larger number of variables and constraints, and it does not benefit from specialised global constraints such as `alldifferent`. As a result, constraint propagation is typically weaker, and the solver may require more search effort. This model is also implemented in MiniZinc and solved using the Gecode solver, allowing for a direct comparision with the integer CP approach. The MiniZinc code implementation is described in Listing 2.

Listing 2: MiniZinc code implementation of the Boolean CP model

```
int: n;
array[1..n, 1..n] of var 0..1: x;

% Exactly one queen per row
constraint forall(r in 1..n) (
  sum(c in 1..n)(x[r,c]) = 1
);

% Exactly one queen per column
constraint forall(c in 1..n) (
  sum(r in 1..n)(x[r,c]) = 1
);
```

```
% Diagonals: r - c constant
constraint forall(d in -(n-1)..(n-1)) (
  sum(r in 1..n, c in 1..n where r - c = d)(x[r,c]) <= 1
);

% Anti-diagonals: r + c constant
constraint forall(s in 2..(2*n)) (
  sum(r in 1..n, c in 1..n where r + c = s)(x[r,c]) <= 1
);

solve satisfy;

output [
  "n=", show(n), "\n",
  "board=\n"
] ++
[
  if fix(x[r,c]) = 1 then "Q " else ". " endif ++
  if c = n then "\n" else "" endif
  | r in 1..n, c in 1..n
];
```

2.4 QUBO Model

The third approach models the N-Queens problem as a QUBO problem. As in the Boolean CP model, a binary variable $x_{r,c}$ is used to represent whether a queen is placed at position (r,c). However, instead of explicitly enforcing constraints, all problem requirements are encoded into a single objective function using penalty terms.

Row and column constraints are enforced using squared penalty functions of the form $(\sum x - 1)^2$, which penalise any deviation from having exactly one queen per row or column. Diagonal constraints are handled using pairwise penalty terms that penalise simultaneous placement of queens on the same diagonal. These penalty terms ensure that solutions violating the problem constraints cause a higher objective value.

The overall objective function is a weighted sum of these penalty terms. A solution with objective value zero corresponds to a valid solution of the N-Queens problem. Appropriate penalty weights are chosen to ensure that constraint satisfaction dominates the search process.

The QUBO model is constructed and solved using the Fixstars Amplify framework, which provides an interface to an annealing-based solver. Unlike CP, this approach is stochastic and

does not guarantee feasibility in a single run. Therefore, multiple independent trials are performed for each problem size, and solution validity and success rates are measured. This model illustrates a fundamentally different solving paradigm compared to the CP models, trading exactness and determinism for modelling flexibility and solver generality. A pseudocode implementation of the QUBO model is provided in Listing 3.

Listing 3: Pseudocode for the QUBO model solved by annealing

```
Algorithm QUBO_NQueens(
    N, w_row, w_col, w_diag, num_reads, time_limit
)
Input: N, penalty weights (w_row, w_col, w_diag),
    annealing params (num_reads, time_limit)
Output: X_best (N x N binary board), valid, energy

Create binary variables X[r,c] in {0,1} for r,c = 1..N
objective <- 0

// Row and column penalties (exactly one queen)
for r = 1..N do
    objective <- objective + w_row *
                        ( sum_{c=1..N} X[r,c] - 1 )^2
end for
for c = 1..N do
    objective <- objective + w_col *
                        ( sum_{r=1..N} X[r,c] - 1 )^2
end for

// Diagonal penalties (at most one queen)
for each main diagonal D (constant r-c) do
    objective <- objective + w_diag *
                        sum_{(i,j) in pairs(D)} X[i]*X[j]
end for
for each anti-diagonal A (constant r+c) do
    objective <- objective + w_diag *
                        sum_{(i,j) in pairs(A)} X[i]*X[j]
end for

Solutions <- Anneal(objective, num_reads, time_limit)
X_best <- BestSolution(Solutions)
```

```
energy <- objective(X_best)
valid <- IsValidNQueens(X_best)
return (X_best, valid, energy)
```

3 Experimental Setup

This section describes the experimental methodology used to evaluate and compare the three modelling approaches introduced in the previous section. The experiments were designed to provide both a fair runtime comparison and an assessment of practical scalability, while accounting for differences in solver behaviour and determinism.

3.1 Design and Rationale

The experimental evaluation was divided into two complementary parts. In the first part, runtime performance was compared over a range of problem sizes for which all three approaches were able to successfully produce valid solutions. This ”common-success interval” ensures a fair comparison, as runtimes are only compared when each method solves the same instances under comparable conditions.

In the second part, a scalability study was conducted to assess how each approach performs as the problem size increases. For this study, each model was evaluated on progressively larger values of N until it failed to produce a valid solution within the imposed time and computational limits. Once a model failed for a given problem size, it was not evaluated further for larger instances, while the remaining approaches continued to be tested. This procedure reflects practical usage scenarios, where solvers are typically abandoned once they become unreliable or computationally impractical.

3.2 Parameters and Evaluation

For the CP models, implemented in MiniZinc and solved using the Gecode solver, a single solver run was performed for each problem size, as these approaches are deterministic. A run was considered successful if a valid N-Queens configuration was produced within the time limit.

For the QUBO model, implemented in Python using the Fixstars Amplify framework and solved using the Amplify Annealing Engine, each problem instance was solved using 10 independent trials. Each trial employed 100 annealing reads and a time limit of 1 second. A QUBO instance was considered successful if all trials produced valid solutions. Otherwise, the approach was deemed to have failed for that problem size. Penalty weights in the QUBO objective function were fixed across all experiments to ensure consistency.

Furthermore, four evaluation metrics were recorded for each experiment: runtime, representing the time it takes to find a solution; solution validity, indicating whether the returned solution satisfies all N-Queens constraints; success rate (QUBO only), determining the proportion of trials that produced a valid solution; and objective value (QUBO only), representing the minimum objective value obtained, with a value of zero indicating full constraint satisfaction.

4 Results

This section presents the experimental results obtained for the three modelling approaches. The results are organised into three parts: a runtime comparison under comparable conditions, an analysis of scalability and failure behaviour, and an examination of the reliability of the QUBO approach.

4.1 Runtime Comparison in the Common-Success Interval

We first compare the runtime of the three approaches over a range of problem sizes for which all methods successfully produced valid solutions. This comparison focuses on problem sizes $N = 4$ to $N = 20$, which is the interval of N before scaling becomes an issue.

Figure 1 shows the runtime of the integer CP model, the Boolean CP model, and the QUBO model. For the CP models, runtime corresponds to a single deterministic solver run, while for QUBO, runtime is reported as the median runtime per trial, reflecting the cost of a single annealing attempt.

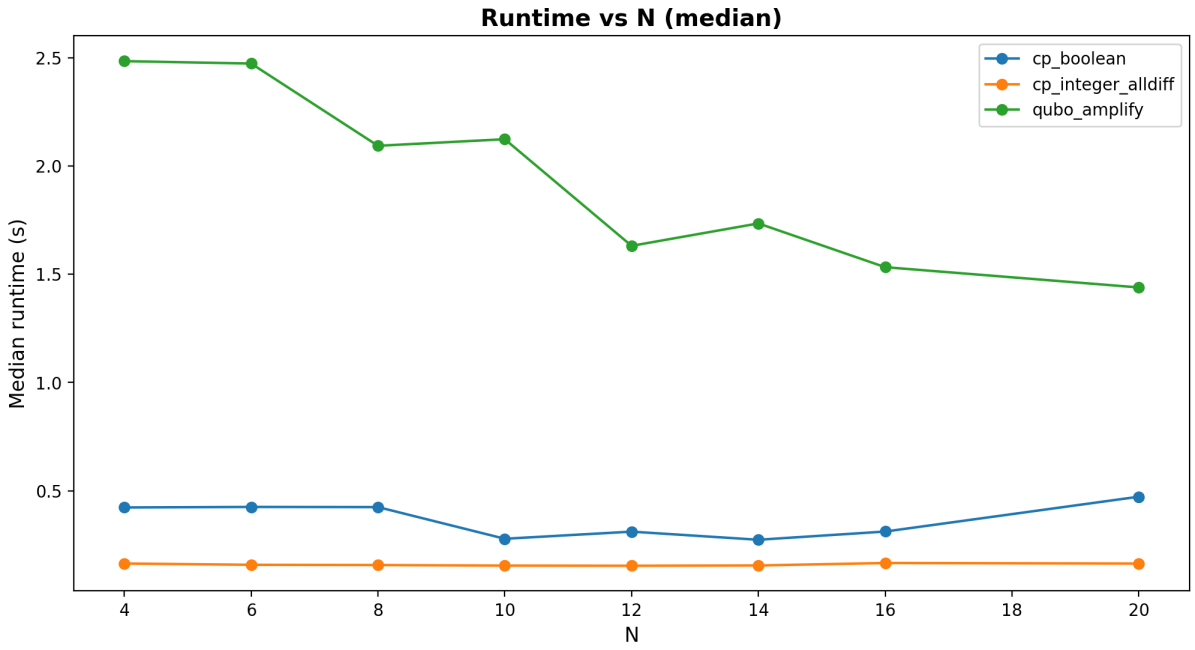

Figure 1: Models runtime comparison in comparable conditions

4.2 Scalability and Failure Behaviour

To evaluate scalability, the problem size was increased until each approach failed under the predefined experimental criteria. Once a method failed for a given value of N, it was not evaluated further, while the remaining approaches continued to be tested. Figure 2 displays the scalability of three approaches, showing the maximum problem sizes solved by each model before satisfying failure criterias.

The Boolean CP model showed a sharp degradation in performance. While it successfully solves instances up to $N = 24$, runtime increases dramatically thereafter, exceeding the 60-second time limit at $N = 30$ and failing to produce a valid solution. This behaviour is consistent with the quadratic number of variables and the absence of strong global constraints.

The integer CP model demonstrates substantially better scalability. It remains fast and reliable for a wide range of problem sizes, successfully solving instances up to $N = 300$. However, at $N = 500$, the solver fails to find a valid solution within the time limit. The observed runtime behaviour is non-monotonic, reflective the search-based nature of CP and the influence of solver heuristics.

The QUBO model shows a different scaling pattern. Under a fixed annealing budget, it consistently produces valid solutions up to $N = 50$. At $N = 60$, the success rate drops below 100%, which was defined as the failure criteria. This indicates a reliability degradation rather an abrupt failure, contrasting with the CP models.

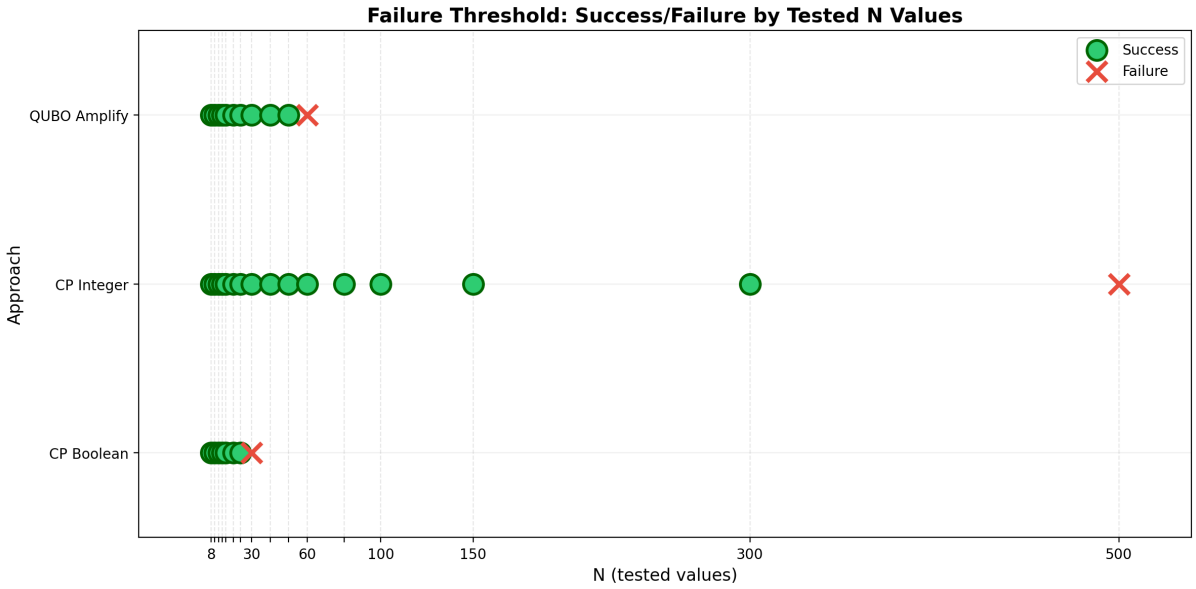

Figure 2: Scalability of the three approaches, showing the maximum problem sizes solved before failure under the experimental conditions.

4.3 Reliability of the QUBO approach

Unlike the CP models, the QUBO approach is stochastic and may produce different outcomes across repeated runs. To account for this behaviour, each QUBO instance was solved using multiple independent trials, and solution validity was measured as a success rate.

For small and moderate problem sizes the QUBO model achieves a success rate of 100%, with all trials producing valid solutions and an objective value of zero. As the problem size increases, the success rate begins to decline under the fixed time budget, as observed in the transition between $N = 50$ and $N = 60$. Despite $N = 60$ causing failure under the constraints defined in this study (success rate $< 100\%$), it still produces valid solutions in 80% of trials. This indicates that the model may still be able to find solutions at $N > 60$, however it can no longer be considered reliable as success is no longer guaranteed. This differs greatly from CP models that either finds a solution every time or never, for any given N.

5 Discussion

The experimental results reveal clear differences between the three modelling approaches, both in terms of runtime efficiency and scalability behaviour. These differences can largely be explained by the underlying modelling choices and solver paradigms.

The integer CP model consistently outperforms the other approaches in terms of runtime across all problem sizes where it succeeds. This behaviour is expected, as the model uses only N decision variables and relies on strong global `alldifferent` constraints, which enable efficient constraint propagation of the search space. Even for large problem sizes, the solver is often able to find a solution with minimal search effort. However, the observed failure at very large N highlights the worst-case exponential nature of search-based methods. The non-monotonic runtime behaviour observed for large instances reflects the sensitivity of constraint solvers to branching heuristics and problem structure, which is common for CP models.

The Boolean CP model has significantly poorer scalability. Although it successfully solves small and moderate instances, its performance degrades rapidly as N increases, eventually failing due to timeouts. This behaviour is linked to the quadratic number of binary variables and the absence of strong global constraints. While the Boolean model is conceptually straightforward and mirrors the chessboard representation, it demonstrates that naive encodings can be inefficient when solved by powerful constraint solvers. This result highlights the importance of careful modelling in CP.

The QUBO model shows a fundamentally different scaling behaviour. Unlike the CP models, it does not fail through rapidly increasing runtime, but instead shows a gradual degradation in reliability under a fixed computational budget. For small and moderate problem sizes, the annealing-based solver consistently finds valid solutions. As the problem size increases, the probability of success decreases, even though the runtime per trial remains relatively stable.

This behaviour reflects the stochastic nature of annealing-based optimisation and the reliance on penalty terms to enforce constraints. Under a fixed time and read budget, the solver may no longer explore the solution space sufficiently to guarantee feasibility.

An important observation is that the QUBO approach degrades gracefully rather than catastrophically. Even when the strict reliability criterion is no longer met, valid solutions are still obtained in a majority of trials. This contrasts with the Boolean CP model, which fails abruptly once the problem size exceeds a certain threshold. The QUBO results therefore illustrate a trade-off between deterministic guarantees and robustness under limited computational budgets.

Overall, the results demonstrate that no single approach is universally superior. The integer CP model is highly effective for this problem when strong global constraints are available, but may eventually encounter worst-case behaviour. The Boolean CP model highlights the cost of inefficient modelling choices. The QUBO model offers flexibility and robustness, but requires careful consideration of computational budgets and penalty parameters.

6 Conclusion

This report investigated three modelling approaches for the N-Queens problem: an integer CP model, a Boolean CP model, and a QUBO model solved using an annealing-based method. Through a systematic experimental evaluation, the impact of modelling choices and solver paradigms on performance and scalability was examined.

The results show that the integer CP model provides the best overall performance, achieving fast runtimes and strong scalability due to its compact representation and use of global constraints. The Boolean CP model, while intuitive, scales poorly and fails for larger problem sizes, demonstrating that modelling efficiency is a critical factor in CP. The QUBO model performs reliably for small and moderate problem sizes and degrades gradually under increasing problem size when evaluated under a fixed computational budget.

These findings reinforce key concepts in combinatorial optimisation, particularly the importance of expressive constraints, the trade-offs between exact and heuristic methods, and the role of stochastic optimisation in combinatorial problem solving. While CP remains highly effective for structured problems such as N-Queens, QUBO models offer a flexible alternative that may be advantageous in contexts where traditional constraint solvers or global constraints are unavailable.

Future work could explore adaptive annealing budgets, alternative QUBO penalty schemes, or hybrid approaches that combine CP with local search or annealing techniques. Such extensions may further illuminate the strengths and limitations of different optimisation paradigms when applied to combinatorial problems.