QUEEN DOMINATION BY SAT SOLVING

Taha Rostami
University of Luxembourg
taha.rostami@uni.lu

Curtis Bright
University of Windsor
cbright@uwindsor.ca

ABSTRACT

The queen domination problem asks for the minimum number of queens needed to attack all squares on an $n \times n$ chessboard. Once this optimal number is known, determining the number of distinct solutions up to isomorphism has also attracted considerable attention. Previous work has introduced specialized and highly optimized search procedures to address open instances of the problem. While efficient in terms of runtime, these approaches have not provided proofs that can be independently verified by third-party checkers. In contrast, this paper aims to combine efficiency with verifiability. We reduce the problem to a propositional satisfiability problem (SAT) using a straightforward encoding, and solve the resulting formulas with modern SAT solvers capable of generating proof certificates. By improving the SAT encoding with a novel literal ordering strategy, and leveraging established techniques such as static symmetry breaking and the Cube-and-Conquer paradigm, this paper achieves both performance and trustworthiness. Our approach discovers and corrects a discrepancy in previous results for $n = 16$ and resolves the previously open case $n = 19$.

1 Introduction

Given an $n \times n$ chessboard, the queen domination problem asks for the minimum number of queens, denoted $\gamma(Q_n)$, required to cover all squares of the board [1]. A queen in chess is a piece that covers all squares in the same row, column, and diagonals from its position. For over a century and a half, mathematicians have studied not only the queen domination problem itself but also various related variants [1]. For example, in addition to determining the optimal value of $\gamma(Q_n)$, researchers have been interested in enumerating all distinct solutions of an $n \times n$ chessboard up to isomorphism [2]. The latter is the primary focus of this paper. Figure 1 provides a concrete illustration of the minimum queen domination problem on a 4×4 chessboard. It presents all 12 optimal solutions and highlights their classification into three isomorphism classes based on board symmetries.

Before the advent and widespread use of computers in mathematics, early efforts to study the problem focused on establishing tighter theoretical bounds [3, 4, 5] and solving small open cases manually [6]. Analytical work continues to this day [7]. However, with the rise of computational methods, previously open cases have increasingly been tackled using algorithmic and computer-based search [2, 8, 9]. This has led to a fruitful synergy between theoretical and practical approaches: theorists provide tighter bounds to narrow the search space, while practitioners solve specific instances and generate data that inform further theoretical insights. A more detailed discussion of prior contributions is provided in Section 5.

Two theoretical observations are particularly useful for constraining the queen domination problem: the inequality $\gamma(Q_n) - \gamma(Q_{n-1}) \leq 1$ restricts how quickly the minimum number of queens can increase as n grows, and the lower bound $\gamma(Q_n) \geq \lceil (n-1)/2 \rceil$ further narrows the range of candidate values [3]. Together, these bounds often reduce the number of values that need to be checked for a given n to just one or two. Recently, the tool UNIDOM [2] was used to determine $\gamma(Q_n)$ for all $n \leq 25$, as well as all (non-isomorphic) solutions using exactly $\gamma(Q_n)$ queens for boards with $n \leq 18$.

Despite the value of computational methods, they come with inherent risks regarding correctness. Since results depend on software (typically custom-written code), the potential of bugs can undermine confidence in reported outcomes. This concern is not hypothetical: discrepancies in computational results have been documented in computer-assisted proofs of other problems, such as in Lam's problem of proving the nonexistence of a projective plane of order ten [10]. Even

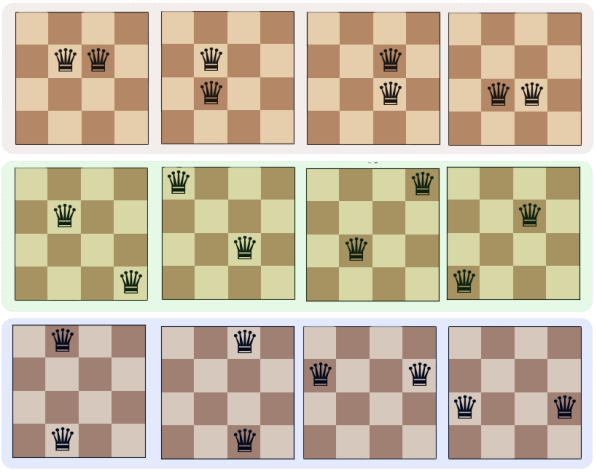

Figure 1: A minimum of two queens is necessary to dominate all squares of a 4×4 chessboard. While there exist 12 distinct configurations that achieve this, many are symmetric under rotation and reflection. Up to isomorphism, these reduce to only three essentially different solutions, as highlighted in the figure.

though Lam's problem was solved by two independent computational implementations [11, 12], discrepancies in the intermediate results of both implementations were later discovered, including the discovery of partial planes that were asserted to not exist but actually do [13]. Unfortunately, it is simply a reality of software development that even the most well-tested programs have bugs [14]. This is particularly important for code that purports to find *all* solutions of a mathematical problem: how can we have confidence that all solutions were indeed found?

One promising direction is the use of Boolean satisfiability problem (SAT) solvers. SAT solvers have already been successfully applied to a number of difficult combinatorial problems in recent years, including the Boolean Pythagorean Triples problem [15], a case of the Erdős Discrepancy Conjecture [16], Lam's problem [10], the enumeration of Williamson matrices [17], and Ramsey number computations [18]. Rather than developing a custom search algorithm—an error-prone and time-consuming endeavor—researchers instead encode the problem in propositional logic. Many combinatorial problems have natural and compact Boolean encodings, and these are often simpler and more reliable to implement compared to custom software. These logical formulations are then passed to SAT solvers which are highly optimized and extensively tested search engines. Modern SAT solvers also output formal resolution proofs in the case of unsatisfiability, enabling independent certification of unsatisfiability results.

UNIDOM is a state-of-the-art search tool written by W. Bird that was used to resolve open instances of the queen domination problem, including the largest board sizes reported in the literature [2]. During the course of our investigation, we discovered a bug that causes UNIDOM to not enumerate all solutions of the queen domination problem when run under certain configurations. Details of a failing testcase are provided in Appendix A.

In our work, we adopt a SAT-based approach to first certify the correctness of all previously reported queen domination results for chessboard sizes $n \leq 18$ and also solve the open instance $n = 19$ under the model enumeration setting—i.e., enumerating all optimal solutions up to isomorphism. Our results are detailed in Section 3.

Even with highly efficient modern SAT solvers, the task of enumerating all solutions of the queen domination problem remains computationally expensive. Domain-specific solvers written for the domination problem, such as UNIDOM, in some cases can even outperform SAT solvers by incorporating custom heuristics and stronger domain-specific reasoning. Modern SAT solvers, commonly known as conflict-driven clause learning (CDCL) solvers, operate on a general-purpose Boolean representation and rely primarily on unit propagation and clause learning. As such, a non-trivial part of our work lies in carefully optimizing the SAT encoding to make the most effective use of the SAT solver's capabilities. Our SAT approach is detailed in Section 2, while Section 4 presents experiments comparing its performance with other encoding methods.

The main contributions of this paper are:

1. We develop a SAT encoding of the queen domination problem and investigate various cardinality constraint encodings for this problem. We propose a novel literal ordering based on the Hilbert curve that significantly outperforms traditional literal orderings. We also enhance our SAT encoding using symmetry breaking and

adopt a parallelization strategy (called Cube-and-Conquer [19]) to further reduce runtime and solve larger instances.

2. We uncover and correct a discrepancy reported in the literature—the number of solutions for $n = 16$ was previously reported to be 43, but our SAT-based approach found 371 non-isomorphic solutions.
3. We solve the previously open case for $n = 19$ and determine there are exactly 11 non-isomorphic solutions in this case.
4. Our results decrease the amount of trust on computational code when compared with the previous approaches used for the queen domination problem, because all our results were accomplished by a proof-producing SAT solver that generated completeness certificates that were subsequently checked by an independent proof verifier. In other words, our results only rely on the correctness of our SAT encoding and the correctness of the proof verifier, not the correctness of the code that did the search.
5. We make our source code and the obtained solutions publicly available.

2 SAT-Based Methodology

As the core of our work relies on SAT solving, we begin with a brief background on propositional satisfiability, interleaved with a description of our problem encoding.

2.1 SAT Preliminaries

We work with propositional formulas expressed in conjunctive normal form (CNF), which is a standard representation used in SAT solving. A formula F is said to be in conjunctive normal form when F consists of a conjunction (AND) of clauses, where each clause is a disjunction (OR) of literals. A literal refers to either a Boolean variable x (known as a positive literal) or its negation $\neg x$ (a negative literal). A truth assignment α assigns each variable a value of 1 (true) or 0 (false). A literal is satisfied under α if the assignment makes it true—specifically, a positive literal is satisfied when the variable is assigned true, and a negative literal is satisfied when the variable is assigned false. A clause is considered satisfied if at least one of its literals is satisfied by α, and a formula is satisfied when all of its clauses are satisfied. Such an assignment is called a model of the formula. If at least one model exists, the formula is deemed satisfiable; otherwise, it is unsatisfiable.

For a satisfiable formula F, one may be interested in enumerating all models that satisfy it. This can be done by repeatedly invoking a SAT solver on F, and each time a model is found adding a "blocking clause" to F that blocks the found model from being a solution. This process continues until the extended formula F' becomes unsatisfiable, indicating that all solutions have been enumerated. For a comprehensive treatment of SAT and SAT solving, refer to the *Handbook of Satisfiability* [20].

2.2 Encoding Queen Domination in SAT

In our case, we consider the queen graph of an $n \times n$ chessboard, with a vertex for each square on the board, and edges connecting squares that share a rank, file, or diagonal. Our goal is to encode the following decision problem in SAT: is there a subset of at most γ vertices that dominate all vertices of the graph?

To express this, let $Q(i)$ be a Boolean variable indicating whether a queen is placed on square i, assuming squares are indexed from 1 to n^2 in row-major order. For each vertex $1 \leq i \leq n^2$, our encoding uses the clause

$$\bigvee_{j \in N(i)} Q(j)$$

where $N(i)$ is the inclusive neighborhood of vertex i. These clauses enforce that each square is either occupied by a queen or threatened by one.

Next, our encoding imposes a cardinality constraint: at most γ of the $Q(i)$ variables can be true. This constraint is not naturally expressible in pure CNF. However, several encoding techniques have been proposed to handle such constraints, including sequential counters [21], sorting networks [22], cardinality networks [23], modulo totalizer [24], and modulo totalizer for k-cardinality [25]. We found that, for this particular problem, the modulo totalizer (mTotalizer) encoding performs best. A comparative empirical analysis supporting this conclusion is presented in Section 4.

2.3 Literal Ordering

One optimization that significantly improves SAT solver performance relates to the ordering of literals in the cardinality constraint. Considering Boolean variables as $\{0, 1\}$ integers, the cardinality constraint has the form

$$Q(1) + Q(2) + \cdots + Q(n^2) \leq \gamma.$$

The ordering of the $Q(i)$ variables impacts the structure of the encoding and the behavior of the SAT solver and thus the solver's efficiency.

Inspired by the work of Reeves et al. [26] that proposed variable ordering strategies such as ordering variables by decreasing clause frequency, we introduce a novel ordering tailored to the queen domination problem. While clause frequency and centrality are useful heuristics, they may overlook important properties of the chessboard such as locality. To better preserve locality, our approach orders literals according to their position in a Hilbert curve [27], a space-filling curve that provides a locality-preserving 1D-to-2D mapping. This ordering better reflects the board's structure and significantly improves performance in our experiments (see Section 4).

2.4 Symmetry Breaking

To reduce the search space and only search for solutions up to isomorphism, our encoding incorporates symmetry breaking constraints based on a lexicographic ordering encoding proposed by Warwick Harvey (see [28]).

Excluding the identity symmetry, the $n \times n$ chessboard has seven symmetries: rotations by $90°$, $180°$, and $270°$, as well as reflections across the horizontal axis, vertical axis, main diagonal, and anti-diagonal. We define a Boolean vector X of size $n \times n$ representing the identity (original) variable ordering. For each of the seven symmetry transformations, we define a corresponding Boolean vector Y of the same size, representing the variable ordering under that transformation, and then add the constraint that X is lexicographically equal or less than Y, written $X \leq Y$.

To encode the lexicographic ordering constraint $[x_1, \ldots, x_N] \leq [y_1, \ldots, y_N]$, we introduce the auxiliary variables a_0, $a_1, \ldots, a_N$. Note $N = n^2$ in our application since there are n^2 board squares. The auxiliary variable a_i has the meaning that the suffix bitvector $[x_{i+1}, \ldots, x_N]$ is lexicographically equal or less than the suffix bitvector $[y_{i+1}, \ldots, y_N]$, and this happens exactly when $x_{i+1} < y_{i+1} + a_{i+1}$. Altogether, Harvey's encoding uses the $3N$ ternary clauses

$$a_{i+1} \vee y_{i+1} \vee \neg a_i, \qquad a_{i+1} \vee \neg x_{i+1} \vee \neg a_i, \qquad \text{and} \qquad y_{i+1} \vee \neg x_{i+1} \vee \neg a_i$$

for each index $0 \leq i < N$, in addition to fixing the first and last auxiliary variables a_0 and a_N to both be true. We use such constraints for all seven nontrivial symmetries of the $n \times n$ chessboard.

3 Main Results

This section presents the main results of our study. For each instance (excluding the trivial cases $n = 1$ and $n = 2$), we encode the problem using our SAT-based method, and perform model enumeration twice—once with UNIDOM under a configuration that avoids a bug (see Appendix A for details) and once with the state-of-the-art SAT solver CADICAL [29]. Once the list of solutions has been determined, we then make a final call to CADICAL to generate a proof of completeness (i.e., that no more solutions exist) by using our encoding described in Section 2 along with clauses blocking each solution from being a model of the SAT instance. For example, if $m_1, \ldots, m_\gamma$ are the true variables representing the positions of the γ queens in a model, we add the blocking clause $\neg Q(m_1) \vee \cdots \vee \neg Q(m_\gamma)$ to the instance.

To handle larger instances, parallelization is necessary. We adopt the Cube-and-Conquer (CnC) approach using `march_cu`, a state-of-the-art lookahead solver [19]. Given a CNF formula, `march_cu` partitions it into many non-overlapping subproblems (cubes), which are evenly distributed across the available CPU cores and solved by instances of CADICAL running in parallel. Each CPU core receives a set of cubes stored in an `.icnf` containing the clauses used in our encoding along with sets of assumptions specifying the subproblems to solve. When verification is required, proofs are generated in the LRAT format and are checked using the proof verifier `lrat-trim` [30]. The running time spent on cubing is negligible compared to the total runtime. For example, the longest cubing time was 1072 seconds, with most cases taking between 200 and 400 seconds—insignificant compared to the total computation time (e.g., for $n = 19$ approximately 3.6 months of total compute time was required).

These experiments are conducted for all $3 \leq n \leq 19$. All runs were executed on Cedar, a heterogeneous high-performance computing cluster provided by Compute Canada mostly using Intel CPUs running at 2.1 GHz. A summary of our results appears in Table 1. Our results reveal a discrepancy in the $n = 16$ problem, where a previous study [2] reported the number of models up to isomorphism as 43, while our experiments found 371 non-isomorphic solutions.

Table 1: Experimental results for model enumeration and verification for instances with $n \leq 19$. Running times are in seconds.

n	# sols.	UNIDOM (sec)	# cubes	SAT Enumeration (sec)	Proof Generation & Verification (sec)	Proof Size
3	1	0	0	0	0	0 MB
4	3	0	0	0	0	0 MB
5	37	0	0	0	0	0.1 MB
6	1	0	0	0	0	0.2 MB
7	13	0	0	0	0	0.5 MB
8	638	0	0	0	1	6 MB
9	21	0	0	0	1	10 MB
10	1	0	0	0	1	10 MB
11	1	0	0	0	1	14 MB
12	1	4	32	6	21	145 MB
13	41	94	32	103	410	1.6 GB
14	588	2,440	750	1,934	18,597	14 GB
15	25,872	69,043	5,000	63,382	353,407	377 GB
16	371	41,275	1,500	22,840	117,298	115 GB
17	22	29,213	5,000	75,373	446,847	444 GB
18	2	30,403	10,000	66,949	604,652	588 GB
19	11	1,023,499	95,187	1,729,621	9,657,395	8.6 TB

The number of solutions for $n = 19$ is new to the best of our knowledge. For all other values of n, our results match those reported in the literature.

For double-checking, we ran experiments using two completely different toolchains—one based on UNIDOM and one based on CADICAL—and both approaches produced consistent results. The generation and verification of proofs further strengthens our findings, because as a result our results do not rely on trusting any form of search code. We do need to trust the code that generates our SAT encoding and the code of `lrat-trim`, but in both cases the code is relatively self-contained and simpler than code implementing a search algorithm.

In some instances UNIDOM outperforms the SAT solver, while in other instances the reverse is true. Interestingly, this performance difference does not depend solely on the parameter n, but likely on a combination of the problem size and the number of models for that instance. It is also possible that our CnC approach, despite some fine-tuning efforts, could benefit from further optimization. An effective parallelization strategy is already supported by UNIDOM, and when parallelization was necessary we called UNIDOM with a `split-depth` parameter of 1.

As a conclusion, the general trend is that for model enumeration followed by proof verification, the combination of UNIDOM for enumeration plus SAT solving for proof generation achieves the best overall performance. As an optimization, the pure SAT-based approach could be used to generate a proof during the model enumeration step. This would remove the need to call a SAT solver again once the enumeration was complete in order to generate the completeness proof, though we did not pursue this avenue.

4 Evaluating Cardinality Constraints and Literal Orderings

This section provides an empirical comparison of our proposed Hilbert curve literal ordering against state-of-the-art methods. Experiments ran on a Linux system with a 13th Gen Intel Core i7–13700H CPU, managed using Python and Bash scripts.

Our implementation relies on third-party tools—CADICAL [29] for SAT solving, `march_cu` [19] for cubing, and Python libraries PySAT [31] and `hilbertcurve`[1] for encoding. To mitigate risks from potential bugs in these dependencies, we applied multiple validation layers, including code reviews and sanity checks. For example, we verified that each literal ordering produces a valid permutation and visually confirmed the correctness of Hilbert curves.

Experiments focused on chessboards sized 12×12 to 15×15, as smaller boards are trivial, while larger ones become computationally expensive for benchmarking. Given a formula S encoding the question "Is there a configuration with at most γ queens dominating an $n \times n$ board?", S is satisfiable when γ equals the optimal number of queens—a case often too easy for the evaluated methods. To better assess solver performance, we instead consider the unsatisfiable variant with $\gamma - 1$ queens, which requires a complete search in order to show that there are no dominating sets of size $\gamma - 1$.

[1] `https://pypi.org/project/hilbertcurve/`

Table 2: Solver performance in seconds for unsatisfiable instances (i.e., instances with $\gamma - 1$ queens). The timeout was 5 hours.

n	$\gamma - 1$	Default order + SeqCard	Default order + mTot.	Occurence order + mTot.	Hilbert order + mTot.
12	5	33	7	3	**2**
13	6	1427	174	59	**19**
14	7	Timeout	11849	3092	**376**
15	8	Timeout	Timeout	Timeout	**14327**

Several cardinality constraint encodings and literal ordering methods have been proposed in the literature. We investigated which combination is most effective for the queen domination problem. For cardinality encodings, we selected well-established methods: sequential counter (SeqCard), sorting network, cardinality network, modulo totalizer (mTot.), and modulo totalizer for k-cardinality. For literal ordering, we considered the default order (i.e., the declaration order), random, occurrence-based, proximity-based, graph-based, Proximity with At-Most-One constraints (PAMO), PAMO combined with occurrence, and finally our proposed Hilbert curve ordering. We evaluated all combinations with both symmetry breaking and Cube-and-Conquer disabled.

Table 2 summarizes our results. In this table, the second column contains the running times for the most straightforward configuration of the default literal ordering with a sequential counter cardinality encoding. The third column contains the times using the default ordering but with the best performing cardinality encoding (mTotalizer) of all the cardinality encodings that we mentioned above. The fourth column gives the times for the best-performing cardinality encoding with the best-performing literal ordering from the literature, the occurence-based ordering from Reeves et al. [26]. Finally, the last column gives the times using the Hilbert curve ordering proposed in this paper with the best-performing cardinality encoding. The results show that our proposed ordering method outperforms all others in terms of runtime.

5 Related Work

In the literature, various variants of the domination problem on chessboards have been explored, differing in board shapes (e.g., rectangular, square, diamond-shaped), types of chess pieces involved (e.g., queens, bishops), placement conditions (e.g., unrestricted positioning, restricted to the main diagonal), and the so-called chain of inequalities that characterize solution sets (e.g., minimal dominating sets, minimum dominating sets, redundant dominating sets). Among these, the minimum queen domination problem on a square chessboard without positional restrictions on the queens has received considerable attention [1]. Research has focused not only on finding the optimal number of queens needed to dominate the board but also on enumerating the number of distinct dominating sets, often considered up to board symmetries or isomorphisms [2].

Previous work in minimum queen domination can be roughly divided into two categories, namely theoretical and practical. In theoretical research, researchers have focused on establishing strong upper bounds on the optimal number of queens needed to cover a chessboard while also deriving stronger lower bounds, either for chessboards of arbitrary size or for specific sizes [32, 33, 34]. This dual approach provides insight into the potential optimal number of queens with a margin of error often limited to one queen. Theoretical studies, however, have yet to determine whether this problem is NP-hard or not and the exact complexity class of the problem remains unknown [2, 35].

On the practical side, researchers have been working to expand the available data by resolving numerous open cases of the problem. For example, Gibbons and Webb [8] incorporated isomorphism rejection within their backtracking search approach to address certain unresolved cases. In subsequent work, Kearse and Gibbons [9] explored three distinct approaches. The first method simply integrated isomorphism rejection into the backtracking search. The second method involved identifying undominated squares during the search process and placing pieces in all positions that attack an undominated square, rather than backtracking on queen placements. The third method centered on backtracking based on the set of queens attacking a given position, rather than their positions. They then conducted empirical evaluations to determine the most effective methods for different partitions of the search space. Following this optimization, they divided the search space into three portions, assigning each subspace to the method expected to perform best based on their experiments, and subsequently solved some new open cases.

More prominently, the specialized highly optimized search tool UNIDOM has recently been proposed and employed to solve several open cases of the queen domination problem [2]. UNIDOM employs an efficient backtracking search implementation with several improvements tailored to the minimum domination set problem. Consider an arbitrary state in the search tree of the minimum domination set problem: a graph, a set of candidate nodes with potential inclusion in the optimal solution, and nodes assumed to be part of the optimal solution are given. UNIDOM utilizes a heuristic

function to select an undominated square, followed by another heuristic function to choose a node capable of dominating the selected undominated square. Next, it adopts a novel bounding strategy—a function that accepts a partial solution and rejects it if it can guarantee that the partial solution will never lead to an optimal solution. Whether rejecting or passing the current partial solution, UNIDOM recursively performs the same procedure for other states in the search tree until finding a solution or reaching exhaustion. Additionally, UNIDOM benefits from distributed computing, specialized data structures, and various implementation optimizations. Leveraging these capabilities, UNIDOM has been applied to solve open cases of different queen domination problems, including model enumeration for board sizes $n \leq 18$.

6 Conclusion

In this paper, we complete a model enumeration of the queen domination problem up to isomorphism for all $n \leq 19$. We achieved this through a logic-based approach of encoding the problem into propositional logic. Given the straightforward nature of expressing this problem as a SAT instance, this strategy is less error-prone than implementing a specialized solver from scratch. Furthermore, our logic-based approach generates correctness certificates that our model enumerations for the queen domination problem are complete, i.e., not missing any models up to isomorphism. These correctness certificates are verifiable by third-party proof checkers and we checked the correctness of them using the tool `lrat-trim` [30]. Thus, to believe in the correctness of our results, you only need to trust the correctness of `lrat-trim`, not the SAT solver that we used.

We further improved our approach by introducing a novel literal ordering based on the Hilbert curve, which preserves spatial locality and substantially outperforms existing ordering strategies. When combined with symmetry breaking and a Cube-and-Conquer workflow, our optimized SAT encoding achieves results competitive with the leading specialized domination set solver, UNIDOM.

Finally, our work corrects some errors in the literature on the queen domination problem. Our studies detected a bug in the set solver UNIDOM previously used to exhaustively find all solutions of the queen domination problem for $n \leq 18$. For the $n = 16$ queen domination problem, we found optimal solutions previously missed by UNIDOM.

For future work, it would be valuable to explore alternative literal orderings that exploit the spatial locality of the chessboard, such as Z-ordering, block-based orderings, or hybrids that combine spatial locality with domination-based heuristics. Another potential optimization is to use partial symmetry breaking in order to reduce encoding size while retaining most of the benefits of full symmetry elimination. Furthermore, we believe that additional cubing refinements may lead to significant further performance improvements. Finally, it would be valuable to investigate whether techniques developed for this problem—such as leveraging spatial locality through Hilbert curve-based ordering—can be generalized to other problems that share similar spatial characteristics.

Code and Data Availability

The code implementation and the minimum data required to apply our method and reproduce the results are available at `https://github.com/TahaRostami/Gamma`. Additional data supporting the main experiments can be found at `https://zenodo.org/records/16683945`.

Acknowledgement

We would like to thank Bill Bird for generously sharing the UNIDOM source code and for patiently addressing first author's inquiries. We would also like to thank Ali Gholami and Wendy Myrvold for their valuable comments and for kindly responding to the first author's questions.

References

[1] J.T. Hedetniemi and S.T. Hedetniemi. Domination in chessboards. In *Structures of Domination in Graphs*, pages 341–386. Springer International Publishing, 2020.

[2] W.H. Bird. *Computational methods for domination problems*. PhD thesis, University of Victoria, 2017.

[3] E.J. Cockayne. Chessboard domination problems. *Discrete Mathematics*, 86:13–20, 1990.

[4] V. Raghavan and S.M. Venkatesan. On bounds for a board covering problem. *Information Processing Letters*, 25:281–284, 1987.

[5] C.M. Grinstead, B. Hahne, and D. Van Stone. On the queen domination problem. *Annals of Discrete Mathematics*, 48:21–26, 1991.

[6] C. F. de Jaenisch. *Traité des applications de l'analyse mathématique au jeu des échecs*. Dufour & Cie, 1862.

[7] W.D. Weakley. Queens around the world in twenty-five years. In *Graph Theory: Favorite Conjectures and Open Problems – 2*, pages 43–54. Springer International Publishing, 2018.

[8] P.B. Gibbons and J.A. Webb. Some new results for the queens domination problem. *Australasian Journal of Combinatorics*, 15:145–160, 1997.

[9] M.D. Kearse and P.B. Gibbons. Computational methods and new results for chessboard problems. Technical report, CDMTCS Research Report Series, 2000.

[10] C. Bright, K.K.H. Cheung, B. Stevens, I. Kotsireas, and V. Ganesh. A SAT-based resolution of Lam's problem. In *Proc. AAAI Conf. on Artificial Intelligence*, volume 35, pages 3669–3676, 2021.

[11] C.W.H. Lam, L. Thiel, and S. Swiercz. The non-existence of finite projective planes of order 10. *Canadian Journal of Mathematics*, 41:1117–1123, December 1989.

[12] Dominique J. Roy. Confirmation of the non-existence of a projective plane of order 10. Master's thesis, Carleton University, 2011.

[13] Curtis Bright, Ilias Kotsireas, and Vijay Ganesh. When satisfiability solving meets symbolic computation. *Communications of the ACM*, 65:64–72, June 2022.

[14] C.W.H. Lam. Opinion: How reliable is a computer-based proof? *The Mathematical Intelligencer*, 12:8–12, December 1990.

[15] M.J.H. Heule, O. Kullmann, and V.W. Marek. Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In *Proc. SAT 2016*, volume 9710 of *LNCS*, pages 228–245. Springer, 2016.

[16] B. Konev and A. Lisitsa. A SAT attack on the Erdős discrepancy conjecture. In *Proc. SAT 2014*, volume 8561 of *LNCS*, pages 219–226. Springer, 2014.

[17] Curtis Bright, Ilias Kotsireas, and Vijay Ganesh. Applying computer algebra systems with SAT solvers to the Williamson conjecture. *Journal of Symbolic Computation*, 100:187–209, September 2020.

[18] Z. Li, C. Duggan, C. Bright, and V. Ganesh. Verified Certificates via SAT and Computer Algebra Systems for the Ramsey $R(3,8)$ and $R(3,9)$ Problems, 2025. To appear at IJCAI 2025.

[19] M.J.H. Heule, O. Kullmann, S. Wieringa, and A. Biere. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In *Haifa Verification Conference*, pages 50–65. Springer, 2011.

[20] A. Biere, M.J.H. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2nd edition, 2021.

[21] C. Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. In *Proc. CP 2005*, pages 827–831, 2005.

[22] K.E. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computing Conference*, pages 307–314, 1968.

[23] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality networks and their applications. In *Proc. SAT 2009*, pages 167–180, 2009.

[24] T. Ogawa, Y. Liu, R. Hasegawa, M. Koshimura, and H. Fujita. Modulo based CNF encoding of cardinality constraints and its application to MaxSAT solvers. In *ICTAI 2013*, pages 9–17, 2013.

[25] A. Morgado, A. Ignatiev, and J. Marques-Silva. MSCG: Robust core-guided MaxSAT solving. *JSAT*, 9:129–134, 2015.

[26] J.E. Reeves, J. Filipe, M.C. Hsu, R. Martins, and M.J.H. Heule. The impact of literal sorting on cardinality constraint encoding. In *Proc. AAAI Conf. on Artificial Intelligence*, volume 39, pages 11327–11335, 2025.

[27] D. Hilbert. Über die stetige abbildung einer linie auf ein flächenstück. In *Gesammelte Abhandlungen, Band 3*, pages 1–2. Springer, 1935.

[28] A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Propagation algorithms for lexicographic ordering constraints. *Artificial Intelligence*, 170:803–834, 2006.

[29] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt. CaDiCaL 2.0. In *International Conference on Computer Aided Verification*, pages 133–152. Springer, 2024.

[30] F. Pollitt, M. Fleury, and A. Biere. Faster LRAT checking than solving with CaDiCaL. In *Proc. SAT 2023*, volume 271 of *LIPIcs*, pages 21:1–21:12. Schloss Dagstuhl, 2023.

[31] A. Ignatiev, A. Morgado, and J. Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *Proc. SAT 2018*, pages 428–437. Springer, 2018.

[32] W.D. Weakley. Domination in the queen's graph. In *Graph Theory, Combinatorics, and Algorithms*, pages 1223–1232. Wiley-Interscience, 1995.

[33] W.D. Weakley. A lower bound for domination numbers of the queen's graph. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 42:231–254, 2002.

[34] A.P. Burger, C.M. Mynhardt, and E.J. Cockayne. Domination and irredundance in the queen's graph. *Discrete Mathematics*, 170:13–24, 1997.

[35] H. Fernau. Dynamic programming for queen domination. In *CTW 2007*, pages 43–48, 2007.

A Bug in UNIDOM

UNIDOM is a fast, specialized solver for domination problems that has recently been used to resolve several previously open cases, including the largest board sizes reported in the literature to date [2]. While UNIDOM is a highly sophisticated and well-optimized tool, it lacks support for proof generation. As a result, using it without caution introduces the risk that undetected bugs may compromise the correctness of its output.

In this section, we present a test case that demonstrates such a bug in UNIDOM under certain settings. This example illustrates how subtle implementation flaws may lead to incorrect results and underscores the importance of close ties to proof complexity in computational investigations of combinatorial problems.

UNIDOM supports distributed computation based on a divide-and-conquer paradigm, which is frequently used to handle larger instances (e.g., $n \geq 15$). In this mode, nodes at the upper levels of the search tree (up to a given cutoff depth) are duplicated across all processes, meaning that each process independently explores this part of the tree. Beyond this depth, the remaining subtrees are partitioned and assigned to specific processes. This distributed mode is parameterized by three user-specified values: the number of jobs, the job ID, and the cutoff depth. While the first two depend on the available computational resources, the cutoff depth must be manually selected.

Our tests revealed a bug in UNIDOM's distributed mode. We instructed the tool to compute all solutions for a specific board size, then aggregated the solutions into a *frequency matrix*, where each entry indicates how often a queen appears at that square across all solutions, normalized by the total number of solutions found. Given the inherent symmetry of the chessboard, the resulting matrix should itself be symmetric. Any deviation from this symmetry indicates a flaw in the computation.

Our attempts to fix the bug were unsuccessful, but experiments revealed that it is specifically tied to UNIDOM's strongest bounding strategy when used with larger cutoff depths. When the cutoff depth was set to small values such as 0, 1, or 2—even with the same bounding strategy—UNIDOM passed our symmetry test. For example, we tested combinations with 1, 5, and 10 processes and cutoff depths of 0 and 1, and in all cases obtained consistent, symmetric results.

Figure 2 illustrates this contrast for $n = 16$: the left heatmap displays the asymmetric frequency matrix resulting from the bug when using a large cutoff depth, while the right heatmap shows the symmetric frequency matrix obtained with a small cutoff depth where UNIDOM operates correctly.

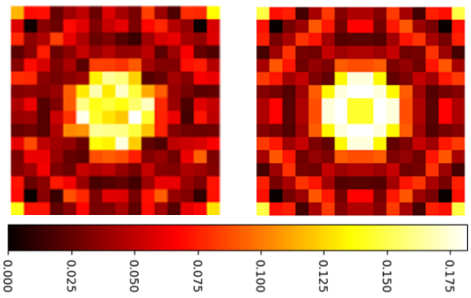

Figure 2: Heatmaps of a 16×16 board: **Left** shows incorrect behavior due to a bug in UNIDOM; **Right** shows correct output with a small cutoff depth.