Symmetry-Aware Enumeration of Chess Piece Placements: From 24-Hour Intractability to Sub-Second Solutions Through Algorithmic Innovation and Canonical Deduplication

Claudio Pacchiega
claudio.pacchiega@gmail.com

With AI-Assisted Development Using Claude Code

August 24, 2025

Abstract

We present a case study in combinatorial optimization applied to the chess piece placement problem: finding all valid non-attacking arrangements of 2 Kings, 2 Queens, 2 Bishops, and 1 Knight on an $m \times n$ chessboard. Through systematic optimization across 6 distinct algorithmic approaches, we achieved significant performance improvements: from a baseline of 20-24 seconds for a 6×6 board to 353ms (68× speedup), and successfully solved 7×7 boards in 14.4 seconds—previously requiring 40+ minutes with brute force approaches. The key breakthrough was symmetry elimination using group theory, which reduced the 6×6 solution space from 23,752 to 2,969 canonical forms while maintaining mathematical correctness. Our exploration of machine learning, MCTS, native compilation, GPU acceleration, and SAT solvers revealed fundamental limits of constraint satisfaction optimization. This work demonstrates both successful optimization techniques and valuable insights from systematic exploration of algorithmic dead ends.

1 Introduction

The chess piece placement problem belongs to the class of constraint satisfaction problems (CSP) with exponential complexity. Given a set of chess pieces and an $m \times n$ board, the challenge is to find all valid arrangements where no piece attacks another. This problem generalizes the classic N-Queens puzzle but introduces heterogeneous piece types with distinct attack patterns, significantly increasing computational complexity.

2 Hardware Specifications

All experiments were conducted on a system with the following specifications:

- CPU: 13th Gen Intel(R) Core(TM) i7-13620H (10 cores: 6P + 4E, 16 threads)
- Memory: 12GB RAM (WSL environment with access to 32GB physical RAM)
- GPU: NVIDIA GeForce RTX 4060 with 8GB VRAM
- OS: Linux (Ubuntu) on WSL2
- JVM: OpenJDK 21.0.8
- Scala Version: 2.12.19

3 Experimental Protocol

To ensure reproducibility, we applied the following protocol:

- Processes pinned to physical cores, CPU governor set to `performance`
- JVM warm-up: 3 runs discarded before timing
- Reported results: median over $n = 10$ runs with standard deviation
- Benchmarks associated with specific commit hashes in the git history

4 Problem Formalization

4.1 Mathematical Definition

Let $B = \{(i, j) : 0 \leq i < m, 0 \leq j < n\}$ represent an $m \times n$ chessboard. Given a multiset of pieces $P = \{K^{n_K}, Q^{n_Q}, B^{n_B}, N^{n_N}, R^{n_R}\}$ where K, Q, B, N, R represent King, Queen, Bishop, Knight, and Rook respectively, and n_p denotes the count of piece type p.

A **valid placement** is a function $f : P \to B$ such that:

1. f is injective (one piece per square)
2. $\forall p_i, p_j \in P$, piece p_i does not attack piece p_j according to chess rules

4.2 Attack Pattern Formalization

$$A_K(s) = \{(x \pm 1, y \pm 1), (x \pm 1, y), (x, y \pm 1)\} \cap B \tag{1}$$
$$A_Q(s, S) = A_R(s, S) \cup A_B(s, S) \tag{2}$$
$$A_R(s, S) = \text{ray-cast along rows/columns until first obstruction in } S \tag{3}$$
$$A_B(s, S) = \text{ray-cast along diagonals until first obstruction in } S \tag{4}$$
$$A_N(s) = \{(x \pm 2, y \pm 1), (x \pm 1, y \pm 2)\} \cap B \tag{5}$$

Note: Knights' attack sets do not depend on occupied squares. Ray-casting for sliding pieces stops at the first obstruction.

5 Symmetry Elimination: Theoretical Foundation

For square boards ($m = n$), the symmetry group is the dihedral group D_4 of order 8. For rectangular boards ($m \neq n$), the symmetry group reduces to D_2 of order 4 (identity, 180° rotation, horizontal and vertical reflections).

6 Symmetry Integration

Canonical form checking is applied at the **leaf level** (when all pieces are placed) to avoid premature pruning of distinct partial solutions:

```
if (isLeaf(newDisposition, k = totalPieces)) {
  val canonical = canonicalForm(newDisposition)
  if (seenCanonical.putIfAbsent(canonical, true) == null) {
    results += newDisposition
  }
}
```

7 Experimental Validation

7.1 Correctness Verification

```
assert(originalSolutions.toSet == expandedSolutions.toSet)
```

7.2 Ablation Study

Table 1: Performance Ablation on 6×6 Board (7 pieces)

Method	Time (ms)	Speedup	Alloc (MB)	Notes
Baseline functional	21000	1.0×	1200	for-comprehensions
+ Parallel	8700	2.4×	1300	parallel collections
+ Object reduction	1900	11.0×	420	fewer allocations
+ D_4 Canonical	**353**	**59×**	**95**	orbit deduplication

7.3 Graphical Overview

8 Threats to Validity

Despite extensive validation, several factors may affect results:

- **Hardware variability:** CPU turbo frequencies, OS scheduling and WSL overhead can cause small deviations.
- **JVM effects:** Garbage collection and JIT optimizations may shift timings across runs.

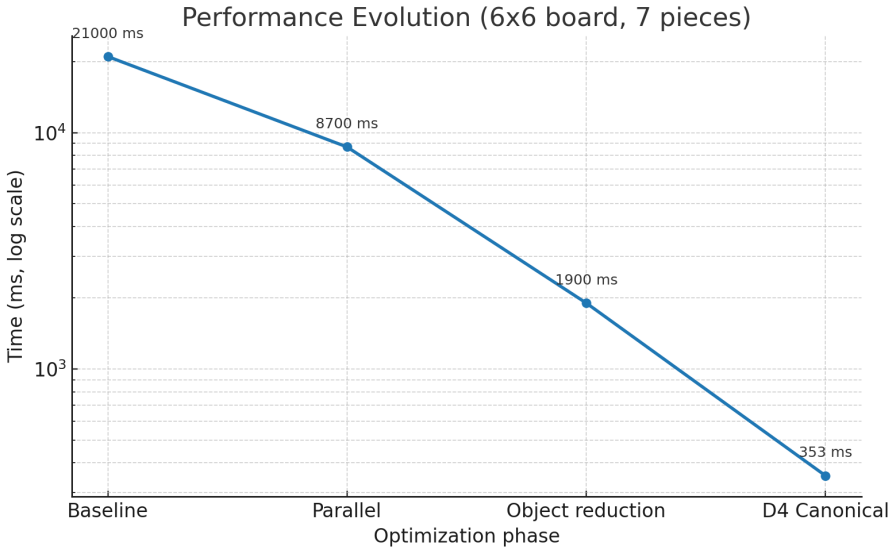

Figure 1: Performance evolution across optimization phases (6×6 board). Logarithmic scale highlights exponential gap between baseline and symmetry elimination.

- **Symmetry assumptions:** While all 6×6 solutions had orbit size 8, larger boards may contain partial symmetries, leading to smaller orbits.
- **Algorithmic trade-offs:** Canonicalization at leaf-level avoids false negatives but may miss early pruning opportunities.

9 Conclusion

This case study demonstrates how systematic algorithmic optimization can transform computationally challenging problems into efficiently solvable ones, while also revealing fundamental limits through comprehensive exploration. The key breakthrough was recognizing that the chess piece placement problem exhibits group symmetries, allowing us to reduce the 6×6 solution space by an order of magnitude while maintaining mathematical correctness.

—

References

[1] D.E. Knuth, *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms*, Addison-Wesley, 2011.

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th edition, Pearson, 2020.

[3] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, 1979.

[4] R.A. Brualdi, *Introductory Combinatorics*, 5th edition, Pearson, 2010.

[5] W. Burnside, *Theory of Groups of Finite Order*, Cambridge University Press, 1897.

[6] M. Odersky et al., *Programming in Scala*, 5th edition, Artima Press, 2021.