

Las Vegas Does N-Queens

Timothy J. Rolfe

Eastern Washington University
 Computer Science Department
 319F Computing & Engineering Bldg.
 Cheney, Washington 99004-2493 USA
<http://penguin.ewu.edu/~trolfe/>
 trolfe@ewu.edu

Abstract

This paper presents two Las Vegas algorithms to generate single solutions to the n -queens problem. One algorithm generates and improves on random permutation vectors until it achieves one that is a successful solution, while the other algorithm randomly positions queens within each row in positions not under attack from above.

Keywords: n -queens problem; Las Vegas algorithm; probabilistic algorithm

1. Introduction

Finding a single solution for the problem of placing n queens on an n by n grid can be done in $O(n)$ time by the deterministic algorithm reported by Hoffman, Loessi and Moore in their article in the 1969 *Mathematics Magazine* [1]. It was referenced in 1991 by Bernhardsson in a note to the *ACM SIGART Bulletin* [2]. Bernhardsson was responding to a 1990 article by Sosič and Gu in the *ACM SIGART Bulletin* [3], presenting a probabilistic algorithm that was reported to run in polynomial time. Sosič and Gu later published an expansion of their work in *IEEE Transactions on Systems, Man, and Cybernetics* [4].

This paper presents alternative probabilistic algorithms that are simpler than Sosič and Gu's algorithm and thus may be useful in teaching probabilistic algorithms. One is based on starting from a random permutation vector but then doing a controlled swap of row positions within the vector based on collisions without explicitly computing and tracking the collisions within the present working vector. The other is based on retaining a boolean matrix indicating row positions under attack from above, and then positioning successive queens in random positions not under attack.

The first algorithm was developed in the context of building a problem set for the Pacific Northwest regional ACM International Collegiate Programming Contest [5] and provided one of the two independent judges' solutions to the problem. It was correctly solved by four out of 87 teams.

2. Algorithm One

The board is represented by a permutation vector giving the positions of the n queens in the n rows. This means that there are necessarily no attacks either horizontally or vertically; one only needs to check for the two diagonal

attacks. These can be tracked by the vectors first introduced by Niklaus Wirth in his 1971 article in the *Communications of the ACM* [6] and later included in his *Algorithms and Data Structures* [7]. One vector flags the diagonals of constant (row-col); the other flags the diagonals of constant (row+col).

```
repeat
  set valid to true
  shuffle the current permutation vector
  set the two diagonal vectors to all false
  mark the row 0 queen in the two diagonal
  vectors
  for row = 1 to n-1
    set test = row+1
    while this queen is on an in-use diagonal
      if test = n
        set valid to false
        break out of the while loop
      else
        swap positions row and test
        increment test
      end if/else
    end while
    if not valid
      break out of the for loop
    mark this row's queen in the diagonal
    vectors
  end for
loop until valid
```

The implementation of this algorithm (in the context of the ACM ICPC problem statement) is available on the author's web site [8]. The following table reflects computations on a Dell desktop computer with Intel Pentium-4 3.80GHz CPU under Windows-XP Professional and Java version 1.5.0_04.

Computation of 1000-Queens Solutions				
Time (seconds)	12.016	37.937	2.047	30.094
Permutation Vectors Used	55,518	175,805	9,490	138,681

3. Parallel Execution

This algorithm also supports parallel execution. One simply needs to insure that the parallel threads or processes are using different seeds for their random number generators. Then one needs to establish a method of communicating among the threads the fact that one of them has obtained a valid solution. In the author's Java implementation, this was achieved by having a static variable within the thread class initialized to null, and then set to the first successful solution. All threads in their computations check the state of that variable and exit immediately when it is found to be non-null. The parallel code is in the same location on the author's web site [8] as the single-processor code.

4. Algorithm Two

This is a significantly slower algorithm than the first, and was suggested by the start-up of Sosič and Gu's Queen Search 3 algorithm in their *IEEE Transactions* article. Retain information about cells within the rows of the board that are under attack from above. Starting from the top, randomly position the queen in each row in one not currently under attack. If there are no such positions, discard that trial solution and begin a new trial solution.

Though significantly slower than Algorithm One, Algorithm Two (nicknamed GridSearch) does execute rapidly enough that it meets the timing criterion from the regional programming contest mentioned above (problem size up to 300, and program completion within 120 seconds). Indeed, it is possible that this may have been among the solutions submitted.

References

- [1] E. J. Hoffman, J. C. Loessi and R. C. Moore, "Constructions for the Solution of the m Queens Problem", *Mathematics Magazine*, Vol. 42, No. 2 (Mar. 1969), p. 66-72. The construction is that of A.M.Yaglom and I.M.Yaglom, *Neelementarnye Zadachi v Elementarnom Izlozhenii* (Moscow, 1954), translated by James McCawley, Jr., revised and edited by Basil Gordon, *Challenging Mathematical Problems with Elementary Solutions: Volume I, Combinatorial Analysis and Probability Theory* (1964, reprinted by Dover 1987), pp. 92-98.
- [2] Bo Bernhardsson, "Explicit solutions to the n -queens problems for all n ", *ACM SIGART Bulletin*, Vol. 2, No. 2 (Apr. 1991), p.7.
- [3] Rok Sosič and Jun Gu, "A Polynomial Time Algorithm for the N-Queens Problem", *ACM SIGART Bulletin*, Vol. 1, No. 3 (xx, 1990), pp. 7-11.
- [4] Rok Sosič and Jun Gu, "Fast Search Algorithms for the N-Queens Problem", *IEEE Transactions on Systems, Man, and Cybernetics*, Vo. 21, No. 6 (Nov/Dec 1991), pp. 1572-76.
- [5] <http://www.acmcontest-pacnw.org/ProblemSet/2005/ Problem I>.
- [6] Niklaus Wirth, "Program Development by Stepwise Refinement", *Communications of the ACM*, Vol. 14, No. 4 (April 1971), pp. 221-27, specifically p. 224.
- [7] Niklaus Wirth, *Algorithms and Data Structures* (Prentice-Hall, 1986), pp. 153-57.
- [8] <http://penguin.ewu.edu/~trolfe/QueenLasVegas/>

```
repeat
  set valid to true
  boolean matrix blocked[n][n] set to all false
  for row = 0 to n-1
    randomly select col as an unblocked cell
    if there is NO unblocked cell
      set valid to false and break from for
      loop
    save col as part of the solution vector
    for all rows below the current row
      mark as blocked the col cell
      also diagonal cells blocked by
        [row][col]
    end for
  end for
loop until valid
```

The C implementation of this algorithm is also on the above mentioned web site [8]. A Java implementation would also support parallel execution in threads similar to that for Algorithm One.

The following table reflects computations on a Dell computer with Intel Pentium-4 3.00GHz CPU under Ubuntu Linux version 2.6.12-8-686-smp and gcc version 3.4.5.

Computation of 1000-Queens Solutions				
Time	39.250	426.600	1508.570	563.920
Trials Required	1,323	14,025	49,466	18,574

Acknowledgement

The author heard of a probabilistic algorithm to obtain a single solution to the n -queens problem involving row swaps from Dr. Ray Hamel. Algorithm One was developed based on that, though it turned out differing significantly from the algorithm that Ray described.