Information Processing Letters 41 (1992) 253–256
North-Holland

3 April 1992

A dynamic programming solution to the n-queens problem

Igor Rivin

NEC Research Institute, 4 Independence Way, Princeton, NJ 08540, USA

Ramin Zabih

Computer Science Department, Stanford University, Stanford, CA 94305, USA

Communicated by D. Dolev
Received 10 December 1990
Revised 8 December 1991

Abstract

Rivin, I. and R. Zabih, A dynamic programming solution to the n-queens problem, Information Processing Letters 41 (1992) 253–256.

The n-queens problem is to determine in how many ways n queens may be placed on an n-by-n chessboard so that no two queens attack each other under the rules of chess. We describe a simple $O(f(n)8^n)$ solution to this problem that is based on dynamic programming, where $f(n)$ is a low-order polynomial. This appears to be the first nontrivial upper bound for the problem.

Keywords: Combinatorial problems, design of algorithms, dynamic programming, n-queens problem, search problems

1. Introduction

The n-queens problem is to determine $Q(n)$, the number of different ways in which n queens may be placed on an n-by-n chessboard so that no two queens are on the same row, column or diagonal. The problem has a long history. The 8-by-8 case was posed in an anonymous problem in [2] in 1848. This was later attributed to Max Bezzel [1, p. 221]. In the years that followed, the problem has attracted the attention of a number of famous mathematicians, including Gauss [3], Lucas [5] and Pólya [7]. It is of current interest mostly as a benchmark problem for backtracking algorithms; [4] is representative of a large literature. Yet there appears to be no algorithm whose complexity is known to be better than the brute-force approach.

A related problem is the *toroidal* n-queens problem. This problem is identical to the (regular) n-queens problem, except that all diagonals are of length n and wrap as if the chessboard were on a torus. If we denote the number of solutions to the toroidal problem as $T(n)$, it is obvious that $T(n) \leqslant Q(n)$.

In this paper we describe a simple algorithm for computing $Q(n)$ or $T(n)$ in time $O(f(n)8^n)$, where $f(n)$ is a low-order polynomial. The algorithm is based on dynamic programming. It is a special case of an approach developed in [9], which views certain search problems as integer linear programming problems and solves them via polynomial multiplication.

Correspondence to: R. Zabih, Computer Science Department, Stanford University, Stanford, CA 94305, USA.

2. Preliminaries

A *line* is a maximal co-linear set of squares on the chessboard (in other words, a row, column or diagonal). If one of the squares in a line has a queen in it, that line is said to be *closed*.

For the toroidal problem, we will define a line to "wrap" off the edge of the chessboard. All lines in the toroidal problem thus consist of n squares.

Lemma 2.1. *The number of lines in the toroidal n-queens problem is* $4n$.

Lemma 2.2. *The number of lines in the (regular) n-queens problem is* $6n - 2$.

A *candidate* C consists of a placement of at most n queens on an n-by-n chessboard. A candidate is *feasible* if no two queens attack each other, i.e., there is no line containing two queens. A *completion* of a candidate is a placement of the remaining queens that results in a solution.

Lemma 2.3. *Two feasible candidates with the same number of closed lines contain the same number of queens.*

Proof. A candidate is feasible just in case the number of closed lines is the number of queens placed multiplied by 4. □

We are now ready to present the theorem on which our results are based.

Theorem 2.4. *Suppose* C_1 *and* C_2 *are feasible candidates having the same set of closed lines. Then any completion of* C_1 *is also a completion of* C_2.

Proof. Suppose that the set of closed lines is S, and that the number of queens placed by C_1 is m. Let C be a completion of C_1. By Lemma 2.3, C_1 and C_2 contain the same number of queens, so adjoining C to C_2 will result in a candidate with n queens. C did not place a queen on any line in S, which consisted of the closed lines in C_2 by assumption. So the result of adjoining C, a completion of C_1, to C_2 is a solution. □

3. Algorithm and analysis

Our algorithm consists of performing dynamic programming using Theorem 2.4. We will perform a breadth-first search of the feasible candidates, where candidates with the same set of closed lines will be placed in an equivalence class. Theorem 2.4 guarantees that we will not overlook any solutions by doing this.

The data structure we need is a set of tuples $\langle S, i\rangle$, where S is a set of (closed) lines and i is an integer. This will be used to represent an equivalence class of i feasible candidates whose set of closed lines is S.

ALGORITHM **Queens**

1. [Initialization] Set QUEUE to $\{\langle \emptyset, 1\rangle\}$.
2. [Square selection] Choose an unexamined square. Let T be the set of four lines containing this square.
3. [Iteration] For every tuple $\langle S, i\rangle \in$ QUEUE such that $S \cap T = \emptyset$ do the following.
   3.1 [Compaction] If $\langle S \cup T, j\rangle \in$ QUEUE for some j, replace j by $i + j$.
   3.2 [Creation] Otherwise, add $\langle S \cup T, i\rangle$ to QUEUE.
4. [Termination] If an unexamined square remains, go to step 2. Otherwise, terminate.

At the end of this algorithm we can easily determine the number of solutions from QUEUE. First, define M to be the set of $2n$ lines that are either rows or columns. A solution places a queen on each line in M. Define the projection function σ on tuples by

$$\sigma(\langle S, i\rangle) = \begin{cases} i & \text{if } M \subseteq S, \\ 0 & \text{otherwise.} \end{cases}$$

After our algorithm has terminated, the number of solutions is

$$\sum_{\langle S, i\rangle \in \text{QUEUE}} \sigma(\langle S, i\rangle).$$

3.1. Analysis

The space requirements of our algorithm depend on the maximum size of QUEUE at any point in the algorithm, which is the possible number of sets of closed lines. If there are α possible closed lines, we will need to store 2^α equivalence classes of feasible candidates. QUEUE can be represented as an array of 2^α elements. The number of possible candidates is $O(2^{n^2})$, so the size of an element in the array is $O(n^2)$, and the space necessary for our algorithm is $O(n^2 2^\alpha)$. Lemmas 2.1 and 2.2 provide an upper bound on α.

The time requirements of our algorithm also depend on α. Step 3 is executed n^2 times, once for each square on the board. Each execution of this step takes time $O(g(n)2^\alpha)$, where $g(n)$ comes from the complexity of integer arithmetic. So the total running time of our algorithm is $O(f(n)2^\alpha)$, where $f(n) = n^2 g(n)$ is a low-order polynomial.

We have thus proved the following theorem.

Theorem 3.1. *$T(n)$ can be computed in time $O(f(n)16^n)$ and space $O(n^2 16^n)$. $Q(n)$ can be computed in time $O(f(n)64^n)$ and space $O(n^2 64^n)$.*

3.2. Improving the algorithm

The number of possible equivalence classes can be further reduced. First, note that any fixed line l consists of at most n squares. Hence the number of times that, in step 2 of our algorithm, we can select a T such that $l \in T$ is bounded by n.

3.2.1. Exhausting lines

In our breadth-first search of the space of feasible candidates, at some point all the squares in l will have been selected. We will refer to this as *exhausting* the line l. Whenever this occurs we have the opportunity to reduce the size of QUEUE. There are two cases to handle; in the first case l is either a row or column, while in the second case it is a diagonal.

If l is a row or column, then any $\langle S, i\rangle$ such that $l \notin S$ can be deleted from QUEUE. This follows from the fact that a solution must place a queen on every row and column. When l has been exhausted, any candidate that fails to place a queen on l cannot be extended to a solution.

The diagonal case is slightly more complicated. Define two candidates to *differ* on a line l if l is closed in one candidate but open in another. Two candidates that differ on exhausted diagonals may be placed in the same equivalence class. This is a result of the following theorem.

Theorem 3.2. *Let C_1 and C_2 be feasible candidates that occur at the same depth in a breadth-first search. If every line l on which C_1 and C_2 differ is an exhausted diagonal, then C_1 and C_2 have the same completions.*

Proof. Let C be a completion of C_1. C_1 and C_2 place the same number of queens, because they have the same set of closed rows (and columns, for that matter). However, C cannot place any queens on exhausted diagonals, so C must be a completion of C_2 as well. □

3.2.2. Using exhaustions

We can modify our algorithm by adding a new step, after step 3 has been completed. The new step loops over all lines (if any) that become exhausted when a square is selected in step 2. For every such l, we do the following:

- If l is a row or a column, then delete every tuple $\langle S, i\rangle \in$ QUEUE such that $l \notin S$.
- If l is a diagonal, then replace every tuple $\langle S, i\rangle \in$ QUEUE such that $l \in S$ by the tuple $\langle S - \{l\}, i\rangle$. If there was another tuple $\langle S - \{l\}, j\rangle \in$ QUEUE, then merge the tuples together (i.e., replace them by $\langle S - \{l\}, i + j\rangle$).

Adding this step increases the efficiency of our algorithm considerably, because an exhausted line will either appear in all tuples (if it was a row or column) or not appear in any tuple (if it was a diagonal).

Let us define a line that has had at least one square selected in the breadth-first search, but has not yet been exhausted, to be *semi-exhausted*. Before this step, our algorithm was exponential in the number of lines. Now our algorithm is exponential in the maximum number of semi-exhausted lines at any point in the search. All that

remains is to produce a bound on the number of semi-exhausted lines. This depends on the order in which squares are selected. However, there is a simple ordering under which the maximum number of semi-exhausted lines is small.

Theorem 3.3. *If the chessboard is processed in row-major order, the number of semi-exhausted lines is bounded by* $3n + c$, *for constant* c.

Proof. Consider the toroidal n-queens problem first. In the first row, the first square contributes four semi-exhausted lines (namely, the four lines passing through it), and the remaining lines in the row contribute three new-exhausted lines (namely, all the lines passing through each except the row line). The last square in the row exhausts the row, hence deleting one row. So after the first row, there are $3n$ semi-exhausted lines. Subsequent rows add a new semi-exhausted line with their first square, and exhaust it with their last square. (Of course, the final row behaves differently by exhausting various lines, but that does not affect our worst-case analysis.)

The nontoroidal case is very similar. In the first row, the first square adds four new lines, and subsequent squares add three new semi-exhausted lines apiece. The last square in the row exhausts two lines: the row, and one of its diagonals (namely, the short single-element diagonal). So after the first row there are $3n - 1$ semi-exhausted lines. The first square in the second row adds a new row line and a new diagonal line, and exhausts the other (two-element) diagonal, for a net gain of one line. The remaining squares in the second row have no effect until we get to the last square, which adds a new diagonal and exhausts the row line and a short (two-element) diagonal, for a net loss of one line. Subsequent rows behave like the second row. Again, the final row exhausts various lines, but that does not concern us. □

By taking advantage of line exhaustion and proceeding in row-major order, we have improved the running time of our algorithm considerably.

Theorem 3.4. *$Q(n)$ and $T(n)$ can be computed in time $O(f(n)8^n)$ and space $O(n^2 8^n)$.*

4. Conclusions

We have described an $O(f(n)8^n)$ algorithm for the n-queens problem. There is some evidence that the number of solutions to the problem is super-exponential [8]. If this is true, then our algorithm is superior to any approach (such as backtracking) that explicitly constructs all solutions to the problem.

Acknowledgment

The technique described in Section 3.2 is due to John Lamping. We wish to thank Ilan Vardi for the use of his bibliography and for various helpful suggestions. We also acknowledge the assistance of two anonymous reviewers. Ramin Zabih is supported by a fellowship from the Fannie and John Hertz Foundation.

References

[1] W. Ahrens, *Mathematische Unterhaltungen und Spiele*, Vol. 1 (Teubner, Leipzig, 1921) 211–284.
[2] Berliner Schachgesellschaft, 1848, Vol. 3, p. 363.
[3] C.F. Gauss, Letter to H.C. Schumacher, September 12, 1850; in: *Werke*, Vol. 12, commissioned by Julius Springer, Berlin, 1929, pp. 20–21.
[4] R.M. Haralick and G.L. Elliott, Increasing tree search efficiency for constraint satisfaction problems, *Artificial Intelligence* **14** (1980) 263–313.
[5] E. Lucas, *Récréations Mathématiques*, "Les carrés magiques", 1891, Vol. 1, Note III, p. 41.
[6] F. Nauck, *Illustrierten Zeitung* **14** (1890) 352.
[7] G. Pólya, Über die "doppelt-periodischen" Lösungen des n-Damenproblems, in: *Collected Works*, Vol. 5, pp. 237–247; originally in: W. Ahrens, *Mathematische Unterhaltung und Spiele*, Zweiter Band (Teubner, Leipzig, 1918) 363–374.
[8] I. Rivin, I. Vardi and P. Zimmermann, The n-queens problem, Preprint, 1990.
[9] I. Rivin and R. Zabih, An algebraic approach to constraint satisfaction problems, in: *Proc. Internat. Joint Conf. on Artificial Intelligence* (IJCAI-89), Detroit, MI, 1989.