Information Processing Letters 34 (1990) 173–178
North-Holland

24 April 1990

# AN ALMOST PERFECT HEURISTIC FOR THE $N$ NONATTACKING QUEENS PROBLEM

L.V. KALÉ

*Department of Computer Science, University of Illinois at Urbana Champaign, 1304 W. Springfield Avenue, Urbana, IL 61801, USA*

Communicated by G.R. Andrews
Received 13 October 1989
Revised 29 December 1989

We present a heuristic technique for finding solutions to the $N$ nonattacking queens problem that is almost perfect in the sense that it finds a first solution without any backtracks in most cases. In addition to previously known variable-ordering heuristics and their extensions, it uses a value-ordering heuristic, which contributes dramatically to its success. Using these heuristics, solutions have been found for all values of $N$ between 4 and 1000.

*Keywords*: Combinatorial problems, backtracking, heuristic search, $N$ queens

## 1. Introduction

The $N$ nonattacking queens problem is defined as follows: Find a placement for $N$ queens on a generalized $N \times N$ chessboard such that no two queens attack each other. A queen is a chesspiece that is said to attack another queen if they are both in the same row, column, or diagonal. I.e., if one queen is at position $(x_1, y_1)$, and the other one is at $(x_2, y_2)$, then they attack each other if either $x_1 = x_2$ or $y_1 = y_2$ or $|x_1 - x_2| = |y_1 - y_2|$.

Although a "toy" problem, this problem has received much attention in computer science literature. It is widely used as a canonical example of a state-space search problem, and of the backtracking search technique. Many new general heuristic techniques are tested and demonstrated on this problem.

A few closed-form solutions are known for the $N$ queens problem [3]. They give a method that works for any value of $N$, for constructing one solution that satisfies the constraints. However, complete search procedures that look for all solutions are of interest. A heuristic search procedure is useful for binding the first $K$ solutions quickly, for any arbitrary value of $K$ that is possible.

## 2. Prior work

A useful observation for this problem is that each row and each column must contain exactly one queen in any solution. So, for example, one may formulate the algorithm as a row-by-row placement of queens. Bitner and Reingold [1] described a few techniques that are likely to be useful for backtracking search in general. These include pruning based on early detection of impossible partial states, and search rearrangement (also called variable rearrangement) based on the most-constrained rule. As a simple application of early detection, one may prune the search when the partial set of queens is seen already to contain a pair of attacking queens. Stone and Stone [5] conducted an extensive empirical study of the $N$ queens problem, and developed the most-constrained heuristics for this problem. This heuristic chooses the next row in which to place a queen as that row which has the fewest possible free posi-

tions left. Using this heuristic, they demonstrated dramatic gains over the lexicographic search, which places queens in rows from 1 to $N$ in the natural sequence. They report that the lexicographic search, using the pruning techniques mentioned above, was not able to find any solutions for $N > 30$ "in a reasonable time", and that their algorithm, called MIN-search, was able to find the first 10 solutions for values of $N \leqslant 96$ using the IBM PC/AT computer. The worst-case time was 1100 seconds for $N = 93$. Their method was unable to find solutions for $N > 96$, and they conjectured that the exponential nature of the search-space may manifest itself beyond that value of $N$.

In this paper we report on a set of heuristic techniques we used to find solutions for all values of $N$ between 4 and 1000, each in less than a minute, on a SUN 3/60.

## 3. The heuristic techniques and performance

Some of the techniques described below have been used and described by others for this problem. We include them here to give a complete presentation of the algorithm. The most important technique is the specific value-ordering heuristic we used, which contributed dramatically to the success of the algorithm. This heuristic is new and not described elsewhere.

We first formulated the algorithm as a backtracking search procedure in the following simplified form (in pseudo-code).

```
extend(State)
  if solution(State)
    then RecordSolution(State)
  else
  if not(impossible(State))
    then
      row = selectRow(State);
      /* see generalization later */
      for col := 1 to N do
        if (free(row, col)) then
          update(State, row, col, Changes);
          /* remember changes */
          extend(State);
          unUpdate(State, Changes);
          /* so they can be undone. */
```

The state representation included the number of placed queens, the set of placed queens, counts for each row and column indicating how many places in that row or column are still "free" and 4-bit vectors for rows, columns, and the two sets of diagonals to record whether each row, column or diagonal is unoccupied. A position is free if no placed queen attacks it. With the diagonal bit vectors, it now takes constant time to decide if a position is free: it is enough to check if the row, column, and the two diagonals it is on are unoccupied, and they can each be tested with a single bit-vector operation.

The impossibility test checks if there is any row or column without any free positions, using the counts kept for each row and column. The update procedure sets the appropriate bits in the bit vectors to indicate the specific row, column, and the two diagonals that are taken, and updates the counts for each row and column, an $O(N)$ operation.

This algorithm, coded in C, performed in rough accord with the results reported by Stone and Stone. It also failed to find a solution to the 97 queens problem even after running for 20 hours on a SUN 3/60.

The next heuristic we tried was to make forced placements whenever possible. A row with only one free position—a forced placement—is already handled by the above algorithm, as this row will be selected before any other row with more than one free positions. However, columns with a single free position are not detected, and this was added to the new algorithm. Although this yielded solutions for many more values of $N$, there were many values of $N$, in the range 1..150, that defied solution within 15 minutes of CPU-time on a SUN 3/60. Some of the failed searches ran for more than 24 hours without a solution. This indicated that there were regions of the search space with very low densities of solutions, and suggested the form of the next heuristic.

Given a row to place the queen in, which columns should we try first? It is clear from the above that the subtree beneath each choice had drastically different solution densities. So, rather than following a standard left-to-right order, some other "intelligent" ordering of the values this col-

umn variable may take should be tried.

As observed by Stone and Stone [5], there are fewer solutions with a queen placed at a corner compared to placing it in the middle of the first row. In fact, the solution densities can be seen to decrease monotonically from the center to either end, with a few exceptions, in the case reported by them. So, instead of left-to-right or reverse order, a middle-out ordering seemed to be better. However, this ordering is justified only for the first row, by this reasoning. How can we apply it to the other rows? Note that the middle elements in the first row have their influence only in the top half of the board, as both the diagonals starting from there end near the middle row. The corner element, in addition to blocking position in its column, also blocks positions in all the rows because of the main diagonal it is on. Empirical evidence (examined for $N \leqslant 14$) suggested that this influence translates into solution densities. So, it is worthwhile using the middle-out order in the rows close to the first row, and by symmetry, those close to the last row. In the middle row, the middle squares extend deeper influence than the edges squares (which extend influence only up to the middle column), and so outside-to-middle ordering would be suitable for the middle row, and those close to it. This intuition led to the following heuristic:

*While placing a queen in the first* $1/3$ *or the last* $1/3$ *rows, start from the middle position, and on backtracking, try outward positions. For the middle* $1/3$ *rows, try placing a queen at the edge (in the first or last column) first, and on backtracking, try inward columns.*

If the count of free positions in a *column* is smaller than this count for all other columns and rows, the "most-constrained" heuristic suggests that the column should be chosen as the next "variable" to be set: i.e., alternate queen positions in that column should be tried. Mixing column-choice-points with row-choice-points does not affect the completeness of the search procedure. However, this symmetric variant did not work satisfactorily with the above value-ordering heuristic (although it solved all instance up to $N = 467$, it failed to find solutions for $N = 468$ even after 20 hours of running time. Many other instances with $N > 468$ failed similarly). We chose instead to select column-placement only when there are three or fewer free positions in that column, and there was no row with fewer free positions. The value ordering used for column-placement was the same as that for the row-placement.

This combination of variable and value ordering was immediately successful. Solutions for all values of $N$ between 4 and 1000 were obtained with very few backtracks. In a majority (750 out of 997) cases, the first solution was obtained without any backtracking at all! Table 1 shows the number of instances that were solved with different number of backtracks. Only 7 out of the 997 instances required more than 29 backtracks. The largest number of backtracks for the first solution, 123, were needed for the 720 queens problem.

To emphasize that this is a complete search procedure, as opposed to a closed-form solution method which produces only one solution (and its symmetric variants), we ran the program to produce the first one hundred solutions for each value of $N$. This is also useful for comparing results with that of Stone and Stone [5]. The total number of backtracks for the first ten solutions for each value of $N$ are plotted in Fig. 1, while those for the first 100 solutions are plotted in Fig. 2. (The average number of backtracks per solution is the number plotted divided by 10 and 100 respectively). It is clear from the plots that a vast major-

Table 1
Number of backtracks for the first solution.

| Number of Backtracks | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10–19 | 20–29 | ≥30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of Instances | 750 | 33 | 46 | 52 | 23 | 15 | 14 | 11 | 8 | 9 | 22 | 7 | 7 |

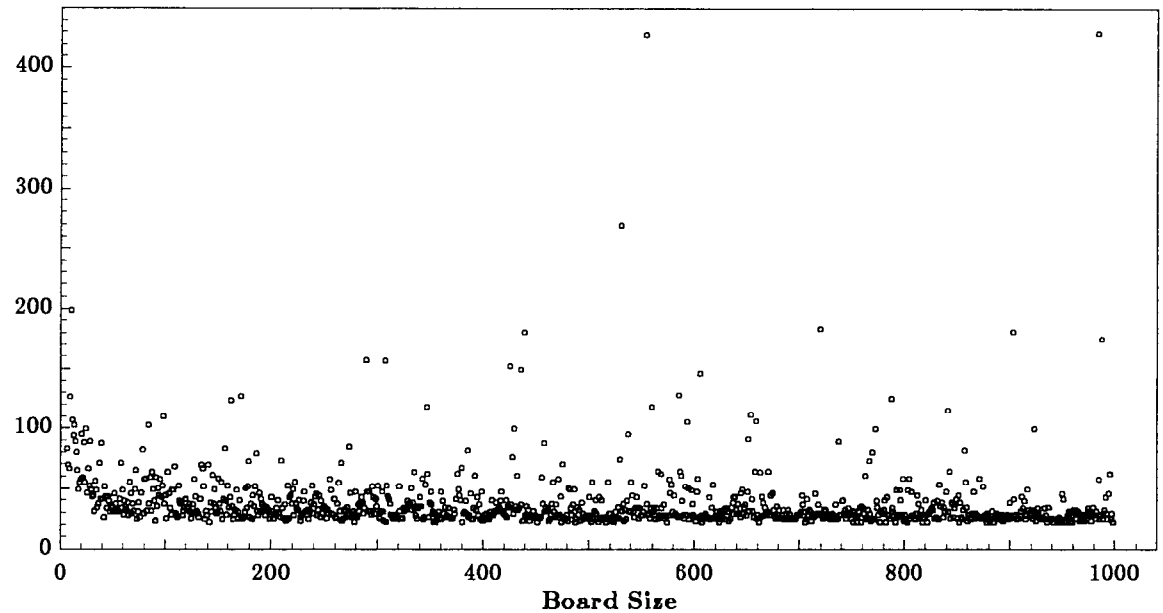


Fig. 1. Number of backtracks for first 10 solutions.

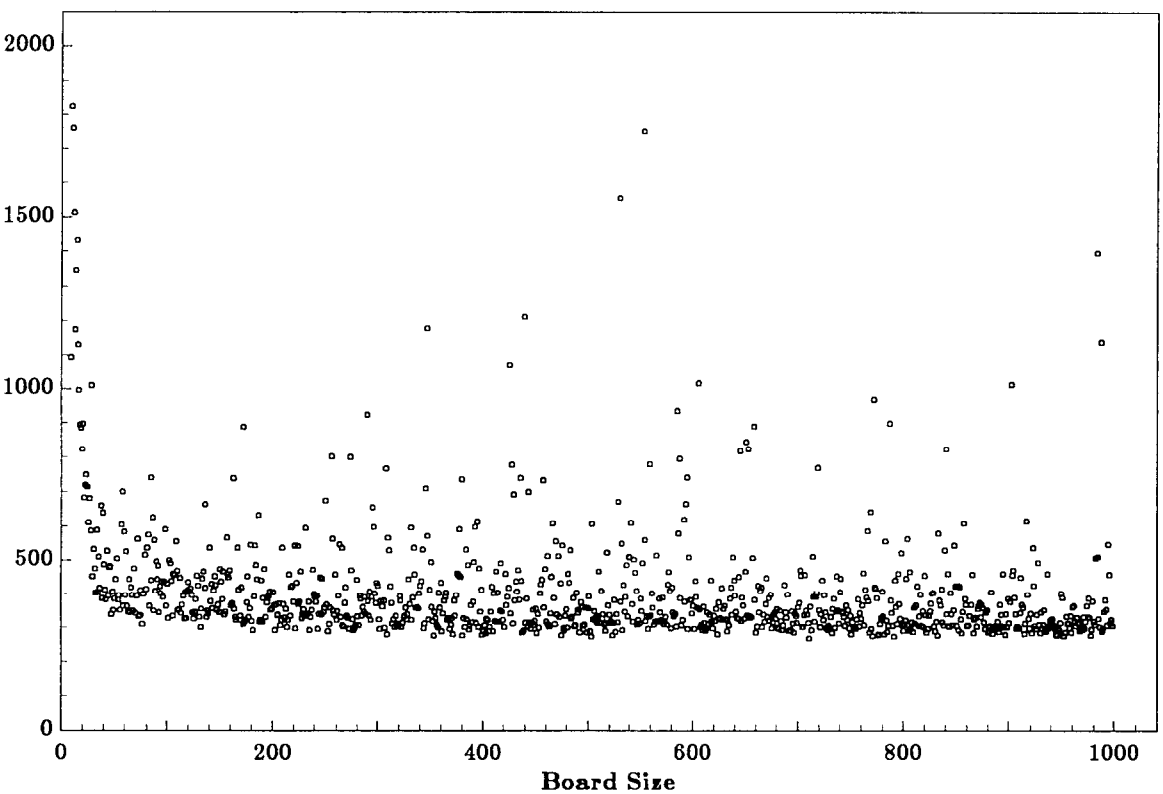


Fig. 2. Number of backtracks for first 100 solutions.

ity of instances yield solutions with only 3–5 backtracks per solution in this range: for 852 out of the 997 instances the first hundred solutions were reported in less than 500 backtracks. The maximum number of backtracks for the first ten solutions is 430, required for $N = 984$. Only about 30 instances required more than 100 backtracks for the first ten solutions. For the first hundred solutions, the largest number of backtracks was 1827, required for $N = 10$. A "boundary effect" can be seen at the lower end. Up to $N = 16$, more than 1000 backtracks were needed for the first 100 solutions. This is partly due to the fact that a larger fraction of the search space needs to be explored (thus having to get into low density regions), and partly because the heuristic itself is less effective in these situations. Only 18 instances required more than 1000 backtracks for the first hundred solutions, (i.e., more than 10 backtracks per solution) of which 8 were for $N \leqslant 16$.

On a SUN 3/60, each of the instances with $N < 1000$ took less than one minute to report the first hundred solutions. The time required for finding the first hundred solutions for an $N$ queens instance is $O(N^2 + BN)$, where $B$ is the number of backtracks required. The time is largely spent in updating the count information for each row and column. The memory requirement of this program is $O(N^2)$, dominated by the cost of remembering the "*Changes*" at each level so they can be done by the *unUpdate* procedure during backtracking. More concretely, for $N = 1000$, it requires 4.02 Megabytes of memory.

## 4. Discussion

A simple empirical result was presented: A specific value-ordering heuristic combined with a variable-ordering (most-constrained) heuristic was shown to be instrumental in producing the first 100 solutions for the $N$ queens problem for all values of $N$ between 4 and 1000.

Table 2
Distribution of queen placements in all solutions for the 11 queens problem.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 96 | 219 | 209 | 295 | 346 | 350 | 346 | 295 | 209 | 219 | 96 |
| 2 | 219 | 220 | 260 | 251 | 277 | 226 | 277 | 251 | 260 | 220 | 219 |
| 3 | 209 | 260 | 250 | 247 | 239 | 270 | 239 | 247 | 250 | 260 | 209 |
| 4 | 295 | 251 | 247 | 238 | 208 | 202 | 208 | 238 | 247 | 251 | 295 |
| 5 | 346 | 277 | 239 | 208 | 178 | 184 | 178 | 208 | 239 | 277 | 346 |
| 6 | 350 | 226 | 270 | 202 | 184 | 216 | 184 | 202 | 270 | 226 | 350 |
| 7 | 346 | 277 | 239 | 208 | 178 | 184 | 178 | 208 | 239 | 277 | 346 |
| 8 | 295 | 251 | 247 | 238 | 208 | 202 | 208 | 238 | 247 | 251 | 295 |
| 9 | 209 | 260 | 250 | 247 | 239 | 270 | 239 | 247 | 250 | 260 | 209 |
| 10 | 219 | 220 | 260 | 251 | 277 | 226 | 277 | 251 | 260 | 220 | 219 |
| 11 | 96 | 219 | 209 | 295 | 346 | 350 | 346 | 295 | 209 | 219 | 96 |

The number of backtracks for the first 100 solutions does not increase significantly with $N$. This suggests the question: Is the average solution density (or its inverse: the number of backtracks per solution) for the $N$ queens problems a constant, independent of $N$? The overall search space certainly increases exponentially with $N$, but the number of solutions is also seen to increase with $N$ empirically. We do not know the answer to this question. The solution densities for $10 \leqslant N \leqslant 14$ vary between 22.7 and 26 backtracks per solution, without a clear monotonic increase with $N$.

Even if the solution densities were constant, the difficulty of solving the $N$ queens problem is not trivialized: for large $N$, there can be (and are, as indicated by our experiments with the failed heuristics) very large regions of the search space without any solution, but the number of solutions in the rest of the space can easily compensate for this to bring the average density to its usual high value. This is possible because of the exponential size of the search space.

The consequences of this result extend beyond the $N$ queens problem. The empirical result underscores the importance of value-ordering heuristics, which have received considerably less attention in the literature than the variable-ordering heuristics. A related point is the importance of estimated solution densities in designing heuristic techniques. Table 2 shows the solution densities, by showing for each position of a $11 \times 11$ board, the number of solutions with a queen in that position. Our value-ordering heuristic is in rough accord with the variation of solution densities shown here. Observing such solution-density variation may lead to an effective heuristic in other search domains as well.

The work presented here arose in the context of a parallel search method [4]. Before the heuristic described here was developed, the parallel algorithm produced solutions to many instances for which the sequential algorithm had failed to solve within its allocated time. This corroborates a recent result by Rao and Kumar [2] that in the presence of nonuniform solution densities, searching many branches in parallel leads to "superlinear" speed-ups over searching one branch at a time, *on the average*. However, a "perfect" heuristic such as the one presented here obviates the need for any parallel search. In problems where such "perfect" heuristics are not known, our result suggests that good value-ordering heuristics should be adhered to even in a parallel search scheme.

## References

[1] J. Bitner and E.M. Reingold, Backtrack programming techniques, *Comm. ACM* **18** (1975) 651–655.

[2] V.N. Rao and V. Kumar, Superlinear speedup in state-space search, in: *Proc. Conference on Foundation of Software Technology and Theoretical Computer Science* (1988).

[3] M. Reichling, A simplified solution to the $n$ queens problem, *Inform. Process. Lett.* **25** (1987) 253–255.

[4] V. Saletore and L.V. Kalé, Parallel state-space search for a first solution, Rept. UIUCDCS-R-89-1549, Dept. of Computer Science, University of Illinois, Urbana, IL (1989).

[5] H. Stone and J. Stone, Efficient search techniques—An empirical study of the $N$-queens problem, *IBM J. Res. Develop.* **31** (4) (1987).