Comparison of Heuristic Algorithms for the N-Queen Problem

Ivica Martinjak
University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
ivica.martinjak@fer.hr

Marin Golub
University of Zagreb, Faculty of Electrical Engineering and Computing
Department of Electronics, Microelectronics, Computer and Intelligent Systems
Unska 3, 10000 Zagreb, Croatia
marin.golub@fer.hr

Abstract. This paper addresses the way in which heuristic algorithms can be used to solve the n-queen problem. Metaheuristics for algorithm simulated annealing, tabu search and genetic algorithm are shown, test results are demonstrated and upper bound complexity is determined. The efficiencies of algorithms are compared and their achievements are measured. Due to the reduction of the fitness function complexity to O(1) problem instances with large dimensions are solved.

Keywords: n-queen problem, heuristic algorithms, simulated annealing, tabu search, genetic algorithm

1. Introduction

The class P refers to the set of all decision problems\(^1\) for which polynomial-time algorithms exist (P stands for “polynomial”). Problems which have a property that, for any problem instance for which the answer is “yes” (in a decision problem) there exist a proof that this answer can be verified by a polynomial-time algorithm are called NP-class problems (NP stands for “non-deterministic polynomial”). A problem Q is said to be NP-hard if all problems in the NP-class are reducible to Q [5]. Due to the high complexity of NP-class problems (e.g. \(O(2^n), O(n^k)\)) they cannot be solved in a reasonable amount of time using deterministic techniques. Therefore, heuristic methods are used to solve these problems in a realistic time frame.

This paper compares heuristic algorithm simulated annealing, tabu search and genetic algorithm in case of the n-queen problem\(^2\) by their efficiencies and achievements. Furthermore, for each algorithm the upper bound complexity is determined as well as complexity of the fitness function. For algorithm simulated annealing and tabu search a heuristic function is created and a custom C program written. All three algorithms are run until the first solution is found; in a series of 10 runs for a given number of queens. To test algorithms’ achievements, problems with up to 100000 queens are solved.

2. N-Queen Problem

It is known that the maximum number of queens that can be placed on an \(n \times n\) chessboard, so that no two attack one another, is \(n\). The eight queens problem is a classical combinatorial problem of putting eight queens on an 8 \(\times\) 8 chessboard so that none of them is able to capture any other. This problem can be generalized as putting \(n\) non-attacking queens on an \(n \times n\) chessboard. The number of different ways the \(n\) queens can be placed on an \(n \times n\) chessboard, in that way, for the first eight \(n\) are 1, 0, 0, 2, 10, 4, 40, 92. That number is known for the first 25 \(n\) (Table 5) [8].

\[ Q (2,4,1,3) \quad Q (3,1,4,2) \]

Figure 1. N-tuple notation examples

\(^1\) A problem that requires a “yes” or “no” answer, in the computational complexity theory.

\(^2\) This famous combinatorial problem is the benchmark of the Constraint Satisfaction Problem (CSP), which consists of three components: variables, values and constraints. The goal of solving a CSP is to find an assignment of values to variables so that all constraints are satisfied [3].
A very poor, brute-force, algorithm for solving the $n$-queen problem, which places a single queen in each row, leads to $n^n$ placements. Since each queen must be in a different row and column, we will use solution representation as $n$-tuples $(q_1, q_2, ..., q_n)$ that are permutation of $n$-tuple $(1, 2, ..., n)$. Using this representation, guaranteeing no rook attacks, the complexity of this problem becomes $O(n!)$. Figure 1 illustrates 4-tuples for the 4-queens problem (all (two) solutions in the 4-queens problem are shown).

Since $n$-tuple representation eliminates row and column conflicts, the wrong solutions have only diagonal attacks between queens. Accordingly, the fitness function\(^3\) should count diagonal attacks. The $2n$-1 “left” and $2n$-1 “right” diagonals have to be checked (Figure 2), but there cannot be a conflict on the first and last diagonal (such diagonals consist of only one field) – so that algorithm should check the $2n$-3 “left” and $2n$-3 “right” diagonals. For a correct solution, the fitness function will return zero. A queen that occupies $i$-th column and $q_i$-th row is located on the $i+q_i$-1 left and $n-i+q_i$ right diagonal. $i$-th and $j$-th queens share a diagonal if:

$$i - q_i = j - q_j$$  

(1)

or

$$i + q_i = j + q_j$$  

(2)

Equation (1) represents “left diagonal” and vice-versa.

This approach leads to $O(n)$ complexity of the fitness function. But in case of simulated annealing algorithm (Figure 7), it was possible to reduce complexity to $O(1)$. Since every new solution differs only in two positions from the previous one, the new value of the fitness function can be calculated by observing the 8 diagonals that eventually change the number of queens.

3 A function $f: A \rightarrow R$ from some set $A$ to real numbers. A feasible solution that minimizes (or maximizes) the fitness function is called an optimal solution.

### 3. Heuristic Algorithms

#### 3.1. Simulated Annealing

Simulated annealing is a heuristic technique of escaping from a locally optimum, which is based on an analogy with a method of cooling metal (known as “annealing”). In each iteration, the new candidate for solution $Y$ (which the heuristic function returns) becomes the new best solution $X_{best}$ (as well as the new current solution $X$), if it is better than the current best solution. However, sometimes it is allowed to replace the current solution with the new candidate even if the new candidate for the solution is not better than the current one (a mechanism of escaping from the locally optimal solution); which depends on “temperature” $T$. The higher the temperature, the higher the probability of replacing the current solution with a worse one. The value of $T$ is decreasing during the time of run, according to cooling ratio $\alpha$.

### Algorithm Simulated Annealing ($c_{max}, \alpha, T_0$)

```python
set T to $T_0$
select initial solution X
$P_x = \text{Fitness}(X)$
$X_{best} = X$
$P_{best} = P_x$

while ($c <= c_{max}$)
    $Y = \text{Heuristic}(X)$
    if (Fitness($Y$) > $P_x$) than
        $X = Y$
        $P_x = \text{Fitness}(Y)$
        if ($P_x > P_{best}$) than
            $X_{best} = X$
            $P_{best} = P_x$
    else
        $r = \text{Random}(1)$
        if ($r < e^{(\text{Fitness}(Y)-\text{Fitness}(X))/T}$) then
            $X = Y$
            $P_x = \text{Fitness}(Y)$

    $T = \alpha \cdot T_0$
return($X_{best}$)
```

### Figure 3. Pseudocode of the simulated annealing algorithm

Algorithm is usually the most sensitive to the cooling ratio $\alpha$. Figure 3 shows the pseudocode of generic simulated annealing algorithm.
whereas Figure 7 presents the pseudocode of that algorithm in case of the \( n \)-queen problem.

3.2. Tabu Search

The basic idea in tabu search is to replace the current solution \( X \) with another one \( (Y) \) with the maximum (minimum) fitness function value in the whole neighbourhood of \( X \) (which is marked as \( N(X) \)). This approach usually involves an exhaustive search of the neighbourhood of the current solution. If in one iteration \( Y \) is the best element in \( N(X) \), it might happen that in the next iteration the best element in the neighbourhood of \( Y \) can be just \( X \) – which would cause the algorithm to enter a useless loop. To avoid this problem (and similar problems such as \( X \rightarrow Y \rightarrow Z \rightarrow \ldots \rightarrow X \) a “tabu list” is used. A tabu list remembers the last \( L \) solutions, which are excluded from \( N(X) \). A tabu list usually does not memorise solutions, but functions that generated them (the function Change(\( ) \) in the pseudocode in Figure 4).

<table>
<thead>
<tr>
<th>Algorithm TabuSearch ( (c_{\text{max}}, L) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>set initial solution ( X ) ( \hat{X}<em>{\text{best}} \equiv X ) ( P</em>{\text{best}} \equiv \text{Fitness}(X) )</td>
</tr>
<tr>
<td>while ( c \leq c_{\text{max}} )</td>
</tr>
<tr>
<td>( N \equiv N(X) \setminus \text{TabuList}[d]; d \equiv c-L, \ldots, c-1 )</td>
</tr>
<tr>
<td>find ( Y \in N ) such that fitness(( Y )) is maximum</td>
</tr>
<tr>
<td>( \text{TabuList}[c] \equiv \text{Change}(Y, X) )</td>
</tr>
<tr>
<td>( X \equiv Y )</td>
</tr>
<tr>
<td>if ( \text{Fitness}(X) &gt; P_{\text{best}} ) then</td>
</tr>
<tr>
<td>( \hat{X}_{\text{best}} \equiv X )</td>
</tr>
<tr>
<td>( P_{\text{best}} \equiv \text{Fitness}(X) )</td>
</tr>
<tr>
<td>return(( X_{\text{best}} ))</td>
</tr>
</tbody>
</table>

**Figure 4. Pseudocode of tabu search algorithm**

3.3. Genetic Algorithms

Genetic algorithms are search and optimization heuristic techniques based on the natural evolution process. The space solution is represented as the population, which consists of individuals that are evaluated using the fitness function representing the problem being optimized. The basic structure of a genetic algorithm is shown in Figure 5.

In each iteration (generation) of algorithm, a certain number of best-ranking individuals (chromosomes) is selected in the manner to create new better individuals (children). The children are created by some type of recombination (crossover) and they replace the worst-ranked part of the population. After the children are obtained, a mutation operator is allowed to occur and the next generation of the population is created. The process is iterated until the evolution condition terminates.

<table>
<thead>
<tr>
<th>Genetic Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>generate initial population</td>
</tr>
<tr>
<td>evaluate the fitness of each individual in the population</td>
</tr>
<tr>
<td>repeat</td>
</tr>
<tr>
<td>select best-ranking individuals to reproduce</td>
</tr>
<tr>
<td>create new generation through crossover and mutation</td>
</tr>
<tr>
<td>evaluate the individual fitnesses</td>
</tr>
<tr>
<td>until (terminating condition)</td>
</tr>
<tr>
<td>return best chromosome</td>
</tr>
</tbody>
</table>

**Figure 5. Structure of genetic algorithm**

4. Experiments with Simulated Annealing

The heuristic element, which appears in all three algorithms, changes two randomly chosen positions (Figure 6). In case of simulated annealing this mechanism consists of the whole heuristic function. This means that in simulated annealing algorithm the fitness function is calculated once in each iteration.

<table>
<thead>
<tr>
<th>NQueenHeuristic ( (X, n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>position1 \equiv \text{Random}(n)</td>
</tr>
<tr>
<td>position2 \equiv \text{Random}(n)</td>
</tr>
<tr>
<td>( Y \equiv X )</td>
</tr>
<tr>
<td>( Y[\text{position1}] \equiv X[\text{position2}] )</td>
</tr>
<tr>
<td>( Y[\text{position2}] \equiv X[\text{position1}] )</td>
</tr>
<tr>
<td>return(( Y ))</td>
</tr>
</tbody>
</table>

**Figure 6. Heuristic element used in compared algorithms**

In the phase of input parameters optimization two initial solutions are compared, one with randomly chosen \( n \)-tuple and the other one with the configuration which \( n \)-tuple \( (1,2,\ldots,n) \) presents. Test results have shown that solutions with queens in the bottom-left to up-right
diagonal usually have more optimal solutions than the other initial solution.

**Algorithm** SA\_n\_queen (α, T₀, n)

set T to T₀
set initial solution X to (1,2,…,n)

Px = Fitness(X)

while (Px > 0)

Y = NQueenHeuristic(X)

if (Fitness(Y) < Px)

X = Y

Px = Fitness(Y)

else

r = Random(1)

if r < exp((Fitness(X)−Fitness(Y))/T) then

X = Y

Px = Fitness(Y)

T = α T₀

return(X)

**Figure 7. Pseudocode of simulated annealing algorithm in the case of n-queen problem; upper bound complexity is O(n²)**

<table>
<thead>
<tr>
<th>n</th>
<th>numiter (in 10 runs)</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>66 803 492.8 7.700</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>202 2006 947.8 9.478</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1403 3777 2159.9 2.400</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>1815 5426 2848.6 1.139</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>3288 9874 6091.3 1.083</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>4482 11769 7872.7 0.787</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>9185 38681 21708.2 0.543</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>21687 31917 24636.2 0.274</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>30103 66846 48435.7 0.303</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>36205 83222 56629.7 0.227</td>
<td></td>
</tr>
<tr>
<td>750</td>
<td>49352 108271 88953.0 0.158</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>95191 223175 126401.7 0.126</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>217367 424044 314373.0 0.079</td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td>304300 729974 464336.7 0.052</td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>664156 1154717 855202.3 0.034</td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td>1520348 2533632 1978524.8 0.020</td>
<td></td>
</tr>
</tbody>
</table>

The simulated annealing program, with α=0.99, T=1000 and “diagonal” initial solution was started 10 times for each number of queens (n); results are shown in Table 1, where numiter is an abbreviation for the number of iterations. According to test results, simulated annealing with this conceptually simple heuristic function presents itself as a very efficient algorithm for this NP-hard problem that runs in polynomial time - the upper bound complexity is O(n²).

**5. Experiments with Tabu Search**

When using tabu search algorithm, the neighbourhood of the current solution X is a set of all n-tuples that are different from X in one exchange of queen places. In each iteration, algorithm finds the best solution (configuration with minimum conflicts between queens) in the neighbourhood. The tabu list remembers the last L pairs of exchanging positions, in order to avoid searching the same neighbourhood repeatedly. The upper bound complexity of this heuristic function is O[(n−1)n/2].

The initial solution with random positions of queens leads to a better performance of algorithm than the initial solution with queens on the bottom-left to up-right diagonal. Experiments with these input parameters are done and results are demonstrated in Table 2.

**Table 1. Simulated annealing results in the n-queen problem**

<table>
<thead>
<tr>
<th>n</th>
<th>numiter (in 10 runs)</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2 17 6.5</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>4 30 10.5</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>7 15 10.7</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>12 28 18.5</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>19 49 29.2</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>26 77 41.8</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>89 187 120.6</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>164 283 209.2</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>349 469 394.6</td>
<td></td>
</tr>
</tbody>
</table>

**6. Experiments with Genetic Algorithm**

In case of genetic algorithm, the mutation operator consists of changing two randomly chosen positions. Algorithm with 3-tournament selection is used. The crossover operator is developed in a way that parents' redundant positions are transferred to a child, and other positions are chosen randomly [2]. Algorithm uses a population of 100 chromosomes, and the probability of mutation was 0.02. The condition
necessary for the evolution to end is to find the solution (which is done when the fitness function returns zero). Experiments are done, and results are shown in Table 3.

**Table 3. Genetic algorithm results in n-queen problem**

<table>
<thead>
<tr>
<th>n</th>
<th>number (in 10 runs)</th>
<th>min</th>
<th>max</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
<td>1</td>
<td>10</td>
<td>4.0</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>16</td>
<td>113</td>
<td>49.1</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>212</td>
<td>1546</td>
<td>917.9</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>491</td>
<td>8364</td>
<td>17592.3</td>
</tr>
<tr>
<td>75</td>
<td></td>
<td>1114</td>
<td>1611</td>
<td>5711.7</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>3395</td>
<td>14581</td>
<td>8877.7</td>
</tr>
<tr>
<td>200</td>
<td></td>
<td>13345</td>
<td>48013</td>
<td>22879.6</td>
</tr>
<tr>
<td>300</td>
<td></td>
<td>20168</td>
<td>38084</td>
<td>27748.2</td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>36471</td>
<td>167404</td>
<td>89406.4</td>
</tr>
</tbody>
</table>

**7. Conclusion**

This paper showed that the n-queen problem can be successfully solved using heuristic algorithms even in case of extremely large dimensions of the problem. Heuristic algorithm simulated annealing, tabu search and genetic algorithm are compared by their efficiencies and achievements. It is demonstrated that a conceptually very simple heuristic function (as in case when the neighbourhood consists of n-tuples which are different from the current solution in the two queens position) can solve this NP-hard problem.

**Table 4. Comparison of used algorithms in the n-queen problem**

<table>
<thead>
<tr>
<th>n</th>
<th>Average number of fitness function computation (in 10 runs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SA</td>
</tr>
<tr>
<td>8</td>
<td>492.8</td>
</tr>
<tr>
<td>10</td>
<td>947.8</td>
</tr>
<tr>
<td>30</td>
<td>2159.9</td>
</tr>
<tr>
<td>50</td>
<td>2848.6</td>
</tr>
<tr>
<td>75</td>
<td>6091.3</td>
</tr>
<tr>
<td>100</td>
<td>7872.7</td>
</tr>
<tr>
<td>200</td>
<td>21708.2</td>
</tr>
<tr>
<td>300</td>
<td>24636.2</td>
</tr>
<tr>
<td>500</td>
<td>56629.7</td>
</tr>
</tbody>
</table>

All compared algorithms run in polynomial time; the complexity of simulated annealing is reduced from $O(n!)$ to $O(n^{2})$ and other two algorithms to $O(n^{3}).$ The complexity of the fitness function in case of simulated annealing is $O(1)$, whereas in two other algorithms it is $O(n)$.

![Figure 8. Dependence of the number of fitness function computation on the number of queens](image)

In case of simulated annealing, the number of fitness function calculations is equal to the number of iterations, whereas in case of other two algorithms the heuristic function is more complicated and the fitness function is calculated more than once in each step of algorithm. Since fitness function calculation takes the most time and this function is the same for each heuristic algorithm, algorithms are compared by the number of fitness function calculation (Table 4, Figure 8).

![Figure 9. Comparison of improvement value of fitness function during iteration](image)
Experiments showed that heuristic algorithms are able to find a different solution for a given number of queens (only in case of genetic algorithm, n ≤ 10, several equal solutions are observed). Furthermore, genetic and simulated annealing algorithm, in contrast with tabu search, usually comes close to the solution very fast and, after that, loses a lot of time for slight improvement (Figure 9). Simulated annealing is the only algorithm that is able to solve instances with large dimensions (500000 queens) of the problem in a realistic time frame which is achieved due to the reduction of the fitness function complexity to O(1).

8. Appendix A: Size of Space Solution and the Number of Solutions of N-Queen Problems

Table 5. Size of space solution and the number of different solutions

<table>
<thead>
<tr>
<th>n</th>
<th>Size of space solution (n!)</th>
<th>Number of solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>120</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>720</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>5040</td>
<td>40</td>
</tr>
<tr>
<td>8</td>
<td>40320</td>
<td>92</td>
</tr>
<tr>
<td>9</td>
<td>362880</td>
<td>352</td>
</tr>
<tr>
<td>10</td>
<td>3628800</td>
<td>724</td>
</tr>
<tr>
<td>11</td>
<td>39916800</td>
<td>2680</td>
</tr>
<tr>
<td>12</td>
<td>479901600</td>
<td>14200</td>
</tr>
<tr>
<td>13</td>
<td>62287208000</td>
<td>73712</td>
</tr>
<tr>
<td>14</td>
<td>871799212000</td>
<td>365596</td>
</tr>
<tr>
<td>15</td>
<td>1.30E+12</td>
<td>2279184</td>
</tr>
<tr>
<td>16</td>
<td>2.09E+13</td>
<td>14772512</td>
</tr>
<tr>
<td>17</td>
<td>3.55E+14</td>
<td>95815104</td>
</tr>
<tr>
<td>18</td>
<td>6.40E+15</td>
<td>666090624</td>
</tr>
<tr>
<td>19</td>
<td>1.21E+17</td>
<td>4968057848</td>
</tr>
<tr>
<td>20</td>
<td>2.43E+18</td>
<td>3.90E+10</td>
</tr>
<tr>
<td>21</td>
<td>5.10E+19</td>
<td>3.14E+11</td>
</tr>
<tr>
<td>22</td>
<td>1.12E+21</td>
<td>2.69E+12</td>
</tr>
<tr>
<td>23</td>
<td>2.58E+22</td>
<td>2.42E+13</td>
</tr>
<tr>
<td>24</td>
<td>6.20E+23</td>
<td>2.27E+14</td>
</tr>
<tr>
<td>25</td>
<td>1.55E+25</td>
<td>2.20E+15</td>
</tr>
</tbody>
</table>

9. Appendix B: 500-Queen Solution

10. References


