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. $O(2^n)$, $O(n!)$,..) 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* x *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 x 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* x *n* chessboard. The number of different ways the *n* queens can be placed on an *n* x *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].

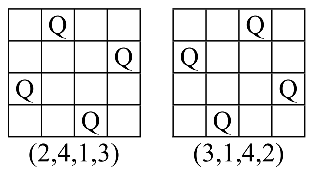

Figure 1. *N*-tuple notation examples

[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].

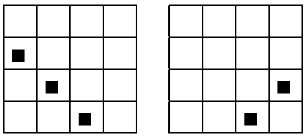

Figure 2. Third "left" and second "right" diagonal

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 2*n*-1 "left" and 2*n*-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 2*n*-3 "left" and 2*n*-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:

$$i - q_i = j - q_j \tag{1}$$

or

$$i + q_i = j + q_j \tag{2}$$

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 $f : A \rightarrow \mathbf{R}$ 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 *α*.

```
Algorithm Simulated Annealing (c_max, α, T_0)

set T to T_0
select initial solution X
P_x = Fitness(X)
X_best = X
P_best = P_x

while (c <= c_max)

    Y = Heuristic(X)
    if (Fitness(Y) > P_x) than
        X = Y
        P_x = Fitness(Y)
        if (P_x > P_best) than
              X_best = X
              P_best = P_x
    else
        r = Random(1)
        if  r < e^((Fitness(Y)-Fitness(X))/T) then
              X = Y
              P_x = Fitness(Y)

    T =  α T_0

return(X_best)
```

Figure 3. Pseudocode of the simulated annealing algorithm

Algorithm is usually the most sensitive to the cooling ratio *α*. 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).

```
Algorithm TabuSearch (c_max, L)

set initial solution X
X_best = X
P_best = Fitness(X)

while (c <=  c_max)
    N = N(X) \ TabuList[d]; d = c-L,…,c-1
    find Y ∈ N such that fitness(Y) is maximum
    TabuList[c] = Change(Y, X)
    X = Y
    if (Fitness(X) > P_best) then
        X_best = X
        P_best = Fitness(X)

return(X_best)
```

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.

```
Genetic Algorithm

generate initial population
evaluate the fitness of each individual in the
population

repeat
    select best-ranking individuals to
    reproduce
    create new generation through crossover
    and mutation
    evaluate the individual fitnesses
until (terminating condition)

return best chromosome
```

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.

```
NQueenHeuristic (X, n)

position1 = Random(n)
position2 = Random(n)
Y = X
Y[position1] = X[position2]
Y[position2] = X[position1]
return(Y)
```

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, T_0, n)

set T to T_0
set initial solution X to (1,2,…,n)
P_x = Fitness(X)

while (P_x > 0)
    Y = NQueenHeuristic(X)
    if (Fitness(Y) < P_x)
        X = Y
        P_x = Fitness(Y)
    else
        r = Random(1)
        if  r < e^((Fitness(X)-Fitness(Y))/T)  then
                X = Y
                P_x = Fitness(Y)

    T =  α T_0

return(X)
```

Figure 7. Pseudocode of simulated annealing algorithm in the case of *n*-queen problem; upper bound complexity is *O*(n^2)

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

n	numiter (in 10 runs)			average / n^2
	min	max	average	
8	66	803	492.8	7.700
10	202	2006	947.8	9.478
30	1403	3777	2159.9	2.400
50	1815	5426	2848.6	1.139
75	3288	9874	6091.3	1.083
100	4482	11769	7872.7	0.787
200	9185	38681	21708.2	0.543
300	21687	31917	24636.2	0.274
400	30103	66846	48435.7	0.303
500	36205	83222	56629.7	0.227
750	49352	108271	88953.0	0.158
1000	95191	223175	126401.7	0.126
2000	217367	424044	314373.0	0.079
3000	304300	729974	464336.7	0.052
5000	664156	1154717	855202.3	0.034
10000	1520348	2533632	1978524.8	0.020
	numiter (until the first solution)			
100000	28272365			0.003
200000	79976796			0.002
500000	198997853			0.001

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 $O(n^2)$.

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 $O\left(\frac{(n-1)n}{2}\right)$.

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 2. Tabu search results in the n-queen problem

n	numiter (in 10 runs)		
	min	max	average
8	2	17	6.5
10	4	30	10.5
30	7	15	10.7
50	12	28	18.5
75	19	49	29.2
100	26	77	41.8
200	89	187	120.6
300	164	283	209.2
500	349	469	394.6

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

n	numiter (in 10 runs)		
	min	max	average
8	1	10	4.0
10	16	113	49.1
30	212	1546	917.9
50	491	83641	17592.3
75	1114	16118	5711.7
100	3395	14581	8877.7
200	13345	48013	22879.6
300	20168	38084	27748.2
500	36471	167404	89406.4

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

n	Average number of fitness function computation (in 10 runs)		
	SA	TS	GA
8	492.8	182.0	400.0
10	947.8	472.5	4910.0
30	2159.9	4654.5	91790.0
50	2848.6	22662.5	1759230.0
75	6091.3	81030.0	571170.0
100	7872.7	206910.0	887770.0
200	21708.2	2399940.0	2287960.0
300	24636.2	9382620.0	2774820.0
500	56629.7	49226350.0	8940640.0

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 $O(n)$.

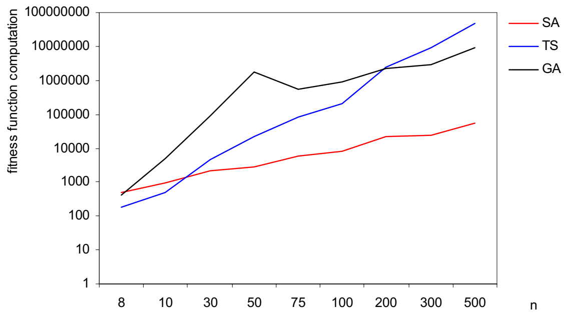

Figure 8. Dependence of the number of fitness function computation on the number of queens

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).

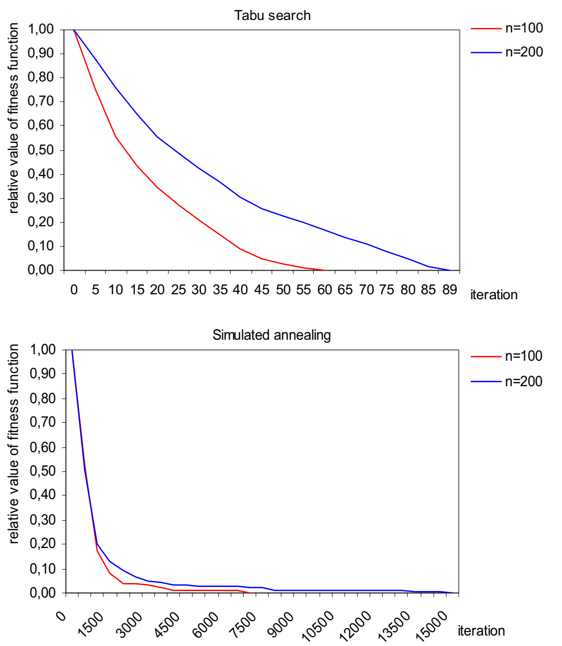

Figure 9. Comparison of improvement value of fitness function during iteration

Experiments showed that heuristic algorithms are able to find a different solution for a given number of queens (only in case of genetic algorithm, $n \leq 10$, 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[4] 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

n	Size of space solution (*n!*)	Number of solutions
1	1	1
2	2	0
3	6	0
4	24	2
5	120	10
6	720	4
7	5040	40
8	40320	92
9	362880	352
10	3628800	724
11	39916800	2680
12	479001600	14200
13	6227020800	73712
14	87178291200	365596
15	1,30E+12	2279184
16	2,09E+13	14772512
17	3,55E+14	95815104
18	6,40E+15	666090624
19	1,21E+17	4968057848
20	2,43E+18	3,90E+10
21	5,10E+19	3,14E+11
22	1,12E+21	2,69E+12
23	2,58E+22	2,42E+13
24	6,20E+23	2,27E+14
25	1,55E+25	2,20E+15

[4] Also, large dimension is possible to achieve with genetic algorithm if parallel genetic algorithm is used [2].

9. Appendix B: 500-Queen Solution

309 243 255 445 3 469 218 284 457 464 129 357 412 405 330 220 84 36 242 103 178 168 259 333 219 241 373 79 250 88 415 254 223 306 383 325 292 482 107 138 136 70 181 249 385 102 324 169 98 268 82 411 44 355 229 171 86 321 317 364 135 313 307 26 361 481 20 227 247 334 164 261 85 11 187 369 374 406 471 393 475 377 452 146 234 282 286 271 50 199 344 375 299 104 74 303 132 225 53 142 339 350 395 55 121 288 488 435 93 191 214 403 239 57 176 216 280 289 463 189 366 116 342 230 89 61 465 408 461 30 112 367 336 212 474 296 473 130 161 123 443 49 253 5 59 311 125 448 391 120 400 97 478 101 235 113 29 37 165 394 48 66 87 15 293 65 301 109 222 141 210 2 345 459 491 16 354 421 444 240 231 92 17 122 467 356 266 19 290 456 388 195 119 431 58 152 470 137 347 368 500 35 285 154 134 462 25 246 332 224 499 90 429 331 359 149 238 291 274 207 414 32 99 438 494 446 489 13 304 417 162 387 118 251 279 153 143 399 108 208 260 63 453 315 38 365 480 114 22 46 64 363 155 83 42 205 305 439 75 226 404 275 442 67 322 139 183 460 449 401 495 56 170 43 197 287 427 117 124 346 248 12 484 386 349 295 422 402 441 202 281 68 51 450 188 209 252 278 396 340 159 54 479 420 409 47 193 433 18 33 458 312 497 300 6 407 211 283 320 308 472 8 455 77 351 206 184 323 167 419 140 454 425 7 39 41 466 451 338 71 486 80 151 430 390 21 95 1 203 397 327 217 437 180 126 148 9 392 389 244 23 358 297 72 215 131 410 96 233 447 14 158 493 270 294 10 245 182 27 228 341 94 424 314 492 52 376 185 204 436 423 172 263 62 370 175 329 257 78 496 111 196 105 179 156 326 477 380 434 145 384 166 316 468 110 360 45 60 100 483 310 258 381 426 343 81 76 265 174 262 198 157 4 133 150 372 69 273 432 160 213 40 318 413 267 498 362 236 31 115 256 186 416 352 34 192 201 353 147 371 485 28 490 221 277 319 487 144 476 173 237 328 272 379 337 177 378 302 200 106 276 428 190 91 73 24 269 335 128 398 127 440 232 382 418 264 194 298 348 163

10. References

[1] Božiković, Marko, Globalni paralelni genetski algoritam, diplomski rad, http://www.zemris.fer.hr/~golub/ga/ga.html (22.05.2006.), Faculty of Electrical Engineering and Computing, Zagreb, 2000.

[2] Božiković, Golub, Budin, Solving n-Queen problem using global parallel genetic algorithm, Eurocon, Ljubljana, Slovenija, 2003.

[3] Han, J., Liu, J., Cai, Q., From Alife Agents to a Kingdom of N Queens, http://arxiv.org/abs/cs/0205016, (10.02.2007.)

[4] Kreher, D.L., Stinson, D.R., Combinatorial algorithms, CRC Press, New York, 1999.

[5] Leung, Y-T. Joseph, Handbook of scheduling, CRC Press, New York, 2004.

[6] Martinjak, I., Golub, M., Comparison of Heuristic Algorithms in Functions Optimization and Knapsack Problem, IIS, Varaždin, 2006.

[7] Sedgewick, Robert, Algorithms, Addison-Wesley Publishing Company, Inc., 1988.

[8] Sloane, Neil J. A., Number of ways of placing *n* nonattacking queens on *n* x *n* board, The On-Line Encyclopedia of Integer Sequences id:A000170, http://www.research.att.com/~njas/sequences/A000170, (30.01.2007.)