Solving N-Queen problem using Gravitational Search Algorithm

[1]Homayun Motameni, [2]Seyyed Hossein bozorgi, [2]mohammad ali shaban nezhad, [3]Golnaz Berenjian, [4]Behnam Barzegar*

[1]Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran
[2]Young Researchers Club, Qaemshahr Branch, Islamic Azad University, Qaemshahr, Iran
[3]Department of Computer Engineering, Tabari Institute of Higher Education, Babol, Iran
[4]Department of Computer Engineering, Nowshahr Branch, Islamic Azad University, Nowshahr, Iran

***Corresponding author**: Behnam Barzegar
behnam.barzegar@yahoo.com or barzegar@iauns.ac.ir

Abstract : The N-queens problem aims at placing N queens on an NxN chessboard, in such a way that no queen could capture any of the others. This problem is considered as one of the hard problem to be solved. Many researches have been interested to solve it with different Heuristic methods. In this paper, a new heuristic method, called the Gravitational Search algorithm or GSA will be shown in solving of the N-queens problem. The offered algorithm is named Gravitational Search algorithm N-queens or GSA-NQ and so as to confirm it, the algorithm should be implemented and compared with GA. The experimental results with proposed method gives the best results compare genetics algorithm.

[Homayun Motameni, Seyyed Hossein bozorgi, mohammad ali shaban nezhad, Golnaz Berenjian, Behnam Barzegar**. Solving N-Queen problem using Gravitational Search Algorithm.** Life Sci J 2013;10(1):37-44](ISSN:1097-8135). http://www.lifesciencesite.com. 6

Key words: N-queen problem, Gravitational search algorithm, combinatorial optimization.

1. INTRODUCTION

Combinatorial optimization problems form a set of problems which need a considerable effort and time to be solved. Their difficulty lies in the fact that there is no formula for solving them exactly. Every possibility has to be examined in order to find the best solution and the number of possibilities increases exponentially as the size of the problem increases. Researchers and scientists offered various Heuristic algorithms for optimization by modeling from physical and biological processes in nature, which often operate collectively. Heuristic algorithms against classic algorithms operate randomly and search along with the space. The other difference between them is that Heuristic algorithms don't use space gradient information. These kinds of methods just use fitness function for guiding the search, but because of having intelligence as type of collective intelligence, are able to find solution. Examples of these algorithms includes inherited algorithms that have inspired by Genetics and evolution science (1975), simulated annealing by modeling from thermodynamics observations (1983), immunity algorithm by simulating human defensive system (1986), searching ants population by simulating ants behavior in finding food (1991), and optimization particles swarm by following birds social behaviors (1995).

Recently, an ultra-Heuristic technique, called gravitational search algorithm, has been introduced by modeling from gravity rule and movement rules, in order to solve optimizing problems. Way of solving group of combined optimization problem, called NP-hard problems effectively, means that finding best solutions among a large but limited series of possible solutions. Today, approximate solutions of NP-hard problems and among them, ultra- Heuristic solutions for solving this kind of problems have been extremely interested. These methods try to search the searching space of an optimization problem regulated by using two concepts of exploring and exploiting.

As much as the power of an algorithm be high in suitable controlling these two parameters, its ability in finding reasonable solutions would be increased.
We also by offering a solution in gravitational search algorithm, controlled these two parameters by researcher factors and finally by algorithm itself, and proved its practicability with this solution. Also, solving N-queen problem with Genetics algorithm and noted gravitational search algorithm and comparing their performances, prove the optimization of search with suggested solution.

The rest of the paper is organized as fallow:

In first section we defined N-queen problem basically. In section two we explained gravitational search algorithm (GSA) and its rules. In section three we evaluated suggested solution and implemented N-queen problem with gravitational algorithm. In section four we compared and evaluated performance of N-queen problem with performance of Genetics algorithm and SA and Hilling. In section five we concluded the paper.

2. RELATED WORK

Ahmed Tariq, has been solved N-queen problem by using a combined method, known as DNA accounting algorithm and Taboo searching and producing the mode by chance [2]. Miguel also has been solved N-queen problem by using P-systems, and presented way of formulating in P-systems and CNF and extended it for larger N in N-queen problem, too [4]. Douglas has been solved N+K-queen problem by reflecting on the plate and related matrix, and turning movement in different directions and degrees [3]. Marko also has been solved N-queen problem with parallel Genetics algorithm and solved it for different functionalizing and different parts and different phases [5].

3. N- QUEEN PROBLEM

The classic combinatorial problem is to place eight queens on a chessboard so that no two attack. This problem can be generalized as placing n non attacking queens on an n×n chessboard. Since each queen must be on a different row and column, we can assume that queen i is placed in i-th column. All solutions to the n-queens problem can therefore be represented as n-tuples $(q_1, q_2, \ldots, q_n)$ that are permutations of an n-tuple (1, 2, 3, …, n). Position of a number in the tuple represents queen's column position, while its value represents queen's row position (counting from the bottom) Using this representation, the solution space where two of the constraints (row and column conflicts) are already satisfied should be searched in order to eliminate the diagonal conflicts. Complexity of this problem is O(n!). Figure 1 illustrates two 4-tuples for the 4-queen problem.

Q			
		Q	
			Q
	Q		

(4, 1, 3, 2)

	Q		
			Q
Q			
		Q	

(2, 4, 1, 3)

Figure 1. n-tuple notation examples

The problem with determining a good fitness function for n-Queen problem is the same as for any combinatory problem: the solution is either right or wrong. Thus, a fitness function must be able to determine how close a wrong solution is to a correct one. Since n-tuple representation eliminates row and column conflicts, wrong solutions have queens attacking each other diagonally.

A fitness function can be designed to count diagonal conflicts: more conflicts there are, worse the solution. For a correct solution, the function will return zero.

For a simple method of finding conflicts [7], consider an n-tuple: $(q_1,\ldots, q_i,\ldots, q_j, \ldots, q_n)$. i-th and j-th queen share a diagonal if:

$$i - q_i = j - q_j \quad (1)$$

or

$$i + q_i = j + q_j \quad (2)$$

which reduces to:

$$\| q_i - q_j \| = \| i - j \| \quad (3)$$

This simple approach results in fitness function with complexity of $O(n^2)$. It is possible to reduce complexity to O(n) by observing diagonals on the board. There are 2n-1 "left" (top-down, left to right) and 2n-1 "right" (bottom-up, right to left) diagonals (see Figures 2 and 3)

■			
	■		
		■	

Figure 2. Third "left" diagonal

			■
		■	

Figure 3. Second "right" diagonal

A queen that occupies i-th column and q_i-th row is located on $i+q_i-1$ left and $n-i+q_i$ right diagonal. A fitness function first allocates counters for all diagonals. Then, for each queen, counters for one left and one right diagonal that queen occupies are increased by one. After evaluation, if a counter has a value greater than 1, there are conflicts on the corresponding diagonal. Fitness value is obtained by adding counter values decreased by 1 (except for counters with value 0) Figure 4 shows a pseudo code for such a function. Note that each counter value is normalized with respect to length of corresponding diagonals.

```
set left and right diagonal counters to 0
for  i= 1 to n
left_diagonal[i+qi] ++
right_diagonal[n-i+qi] ++
end
sum = 0
for i = 1 to (2n-1)
counter = 0
if (left_diagonal[i]  >  1)
counter += left_diagonal[i] - 1
if (right_diagonal[i]  >  1)
counter += right_diagonal[i] - 1
sum += counter / (n-abs(i-n))
end
```

Figure 4. Fitness function for n-queen problem

4. GRAVITATIONAL SEARCH ALGOROTHM

In GSA, optimization is done by using gravitational rules and movement rules in an artificial discrete-time system.

System area is same as problem definition area. According to gravitational rule, act and state of other masses are recognized through gravitational forces. So, this force could be used as a tool for transferring information. We can also use suggested solution for solving any optimization problem which within it any answers of problem is definable as a state in space, and its degree of similarity with other answers of problem is mentioned as a distance.

Value of masses in each problem is also mentioned in regards to purpose function. In first step, system space is determined. Area includes a multi-dimensional coordinated system in problem definition space. Each point in space is one of the answers of problem and search factors are also series of masses.

Each mass has three properties:
a) mass state, b) gravitational mass, c) Inertia mass.

Abovementioned masses are resulted from active gravitational mass and Inertia mass concepts in physics. In physics, active gravitational mass is criteria of degree of gravitational force around a body, and Inertia mass is criteria of body resistance against movement. These two properties could be not equal, and their amounts are determined base on suitability of each mass. Mass state is a point in space which is one of problem answers. After forming system, its rules are determined.

We suppose that only there are only gravity rule and movement rule. Their general forms are similar to nature rules and have defined as below:

Gravity Rule): Any mass in an artificial system attracts all other masses toward itself. The value of this force is proportional with gravitational mass of related mass and distance between two masses.

Movement Rule): Recent speed of each mass is equal to sum of the coefficient of last speed of that mass and its variable speed. Also, acceleration or variable speed is equal to delivered force on mass, divide on amount mass.

In following, we explain principals of this algorithm: Suppose that there is a system with S masses and within it, state of mass i-th is defined as relation (4), where x denotes position of mass i-th in dimension d and n denotes number of dimensions in the search space.

$$X_i = (x_i^1,\dots,x_i^d,\dots,x_i^D) \qquad (4)$$

worst(t) and best(t) are for minimization problems and are calculated with relations (5) and (6).

(For maximization problems is just enough to consider the inverse of these two relations).

$$\mathrm{Best}(t) = \max_{j\in\{1,\dots,m\}} \mathrm{fit}_j(t) \qquad (5)$$

$$\mathrm{worst}(t) = \min_{j\in\{1,\dots,m\}} \mathrm{fit}_j(t) \qquad (6)$$

We can account fitness of recent population with relation (7), and obtain mass of factor i-th in time t (i.e. with relation (8)), where M and fit are denote mass and fitness of factor i-th in time t, respectively.

$$q_i(t) = \frac{fit_i - worst(t)}{best(t) - worst(t)} \qquad (7)$$

$$M(t) = \frac{q_i}{\sum_{j=1}^{s}(t)} \qquad (8)$$

In this system, force F is delivered on mass i-th from mass j-th in time t in the direction of dimension d, which value of this force is obtained base on relation (9), And in relation (9), G(t) is gravity constant in time t which is regulated in the beginning of operating algorithm, and is decreased by the time.

$$F_{ij}^d(t) = \frac{G(t)\times M_j(t)}{R_{ij}(t)+\varepsilon}\,\left(X_j^d(t) - X_i^d(t)\right) \qquad (9)$$

R is ECLIDIAN distance between factor i-th and factor j-th that is defined as relations (10),” ” is also a small value for avoiding denominator from becoming zero.

(10)

$$\mathrm{ij} = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2 + \cdots + (n_2 - n_1)^2}$$

The force delivered on mass i-th in direction d at time t is equal to resultant of total force from k superior mass in population (k is better factor than recent factor). Kbest denotes series of k superior masses in population. K value is not constant and is defined as a time-dependant value, such that all masses at the beginning influence on each other and deliver force, but by passing time, number of effective members in population is decreased linearly. And for accounting sum of delivered forces on mass i-th in dimension d we could write (11). In this relation, rand is a random number with normal distribution in the interval [0,1].

$$\mathrm{F}_{\mathrm{i}}^{\mathrm{d}}(t) = \sum_{j\epsilon kbest, j\neq i} rand\, j \times (t) \frac{M_j(t) \times M_i(t)}{R_{ij}(t)+\varepsilon} (X_j^d(t) - X_i^d(t)) \qquad (11)$$

According to Newton's second movement rule; each mass takes acceleration in the direction of dimension d, which is proportional with delivered force on that mass, and has mentioned in relation (12).

$$\mathrm{a}_{\mathrm{i}}^{\mathrm{d}}(\mathrm{t}) = \frac{\mathrm{F}_{\mathrm{i}}^{\mathrm{d}}(\mathrm{t})}{\mathrm{M}_{\mathrm{i}}(\mathrm{t})} \rightarrow \qquad (12)$$

$$\mathrm{a}_{\mathrm{i}}^{\mathrm{d}}(\mathrm{t}) = \sum_{\mathrm{j\epsilon kbest, j\neq i}} \mathrm{rand\ j} \times \mathrm{G(t)} \frac{\mathrm{M_j(t)} \times \mathrm{M_i(t)}}{\mathrm{R_{ij}(t)} + \varepsilon} (\mathrm{X}_{\mathrm{j}}^{\mathrm{d}}(\mathrm{t}) - \mathrm{X}_{\mathrm{i}}^{\mathrm{d}}(\mathrm{t}))$$

And speed of each mass is equal to sum of coefficient of mass recent speed and acceleration, and is explained as relation (13). In this relation, rand is a random number with normal distribution in the interval [0,1], and its random property is resultant of keeping search in random mood.

$$\mathrm{V}_{\mathrm{i}}^{\mathrm{d}}(t+1) = rand_i \times V_i^{d}(t) + a_i^{d}(t) \qquad (13)$$

Now, mass should moves. It is obvious that more speed of the mass, cause more movement in that dimension. New state of factor i-th is mentioned by relation (14).

$$\mathrm{X}_{\mathrm{i}}^{\mathrm{d}}(t+1) = \mathrm{X}_{\mathrm{i}}^{\mathrm{d}}(t)_i + V_i^{d}(t+1) \qquad (14)$$

At the beginning of forming system, each mass (factor) is randomly positioned in one point of space that is an answer of problem. In each moment, masses are evaluated and then changing in the position of each mass is calculated after solving relations 11 to 14. System parameters are updated in each stage (G, M).

Stop condition could be determined after passing specified time. In Figure 5, semi-code of this algorithm has been presented [1,6]:

```
1) determining system area and initial valuing;
2) initial positioning the masses;
3) evaluating masses;
4) updating parameters G, best, worst and M ;
5) calculating delivered force on each mass;
6) accounting acceleration and speed of each mass;
7) updating position of masses;
8) if stop condition doesn't meet, go to phase3.
```

Figure 5. semi-code of GSA

5. Proposed Algorithm

As we know, state of a N*N chessboard with queen chessmen on it, could be mentioned as an one-dimensional array with N cells. So each cell of this array is attributed to one column of plate, and number of each house of array denotes number of that line which queen is there.

For example, second house of below array shows that queen has been placed in second column and third line (Figure 6).

Q			
	Q		Q
		Q	

4	3	1	2

Figure 6. N-Queen problem

In gravitational algorithm, each factor in searching space includes one-dimensional array which keeps recent state of queen on related plate. So with five masses, we in fact applied five researcher factors for finding target state. Each factor independently has a mass, which is determined in regard to number of queen crashes. In order to show that heavier mass has better state, we should subtract number of queen crashes from a constant number (we suppose that this constant number is number of maximum crashes of N queen).

Result answer of this subtraction is q_i , conforming with formula (7). Now if base on formula (8), we divide fitness of one factor on sum of factors fitness, mass factor is attained. Accounting delivered force, acceleration, speed and position of each mass are depended on dimension of each mass, and they are independent of each other.

Consider a two-dimensional space. If there are two masses in one column during applying calculations on dimension X, calculations should be stopped, since second mass doesn't deliver force on first mass in direction of dimension X.

For example, in Figure 7, you see that two masses (A and B) are placed in one column, so they don't deliver force in direction of dimension X on each other. And similarly, two masses C and D are placed in one line, and so don't deliver force on each other in direction of dimension Y.

But pair masses (B, C) , (B, D) , (C, D) , (A, D) are delivered force on each other in both directions of dimensions X and Y, and so calculations are applied on them completely (Figure 7).

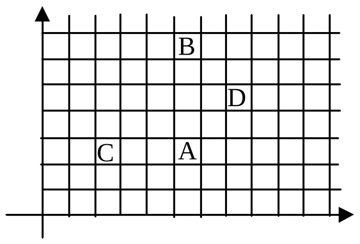

Figure 7. masses A,B,C,D

Therefore, in first condition, we investigate un-parallelism of two masses in interested dimension. Then in order to account sum of delivered forces on related mass, we need to determine forces delivered from those masses which are placed in Kbest series.

Kbest array is filled with initial value of (-1). According to gravitational algorithm, at the first moment of operating algorithm, all masses deliver force on each other. After assessing first condition, we add number of masses to Kbest series (Figure 8).

```
for (byte j = 0; j < j_num - 1; j++)
              K_best[j] = -1;

for (byte i = 0; i <= mass_num - 1; i++)
 {
   if (Loc_arr[0, i] >= n)
      Loc_arr[0, i] = n - 1;
    if (Loc_arr[1, i] >= n)
       Loc_arr[1, i] = n - 1;
    if (Loc_arr[0, k] != Loc_arr[0, i])
      {
       for (byte j = 0; j < mass_num - 1; j++)
         if (K_best[j] == -1)
           {
        K_best[j] = arr[Loc_arr[0, i], Loc_arr[1, i]];
             break;
            }
       }
 }
```

Figure 8. calculate of K_best

It is obvious that according to gravitational algorithm, in next moment, we should add the condition of "being masses heavier" to the first condition, i.e. in addition to condition of un-parallelism of masses, those masses which are heavier than recent masses, should be added to Kbest series.

Now, we could write calculations as follow (Figure 9):

```
while ((K_best[l] >= 0) && (number <= mass_num - 2))
 {
  R = Math.Sqrt((Math.Pow((Loc_arr[0, k_best_T] –
  Loc_arr[0, k]), 2) + Math.Pow((Loc_arr[1, k_best_T] –
  Loc_arr[1, k]), 2)));
  F_arr[0, k] = F_arr[0, k] + ((rand_obj.Next(100) / 100.0) * G *
  (Math.Abs((hiu_mass[k_best_T] - hiu_mass[k])) / (R + E)) *
  Math.Abs(Loc_arr[0, k_best_T] - Loc_arr[0, k]));
 }
 A_mass = F_arr[0, k] / hiu_mass[k];
 V_arr[0, k] =((rand_obj.Next(100)/100.0)* V_arr[0, k]) + A_mass;
 x_temp=Convert.ToInt32(Loc_arr[0,k]+Math.Round(V_arr[0, k]));
```

Figure 9. calculate of Force

New positions of mass, has been specified. And it is obvious that researcher factor should have new state and finally new mass in new position of search space. But how these changes in state and mass should be created?

In suggested solution, we divide state array on N dimensions of search space, i.e. for each dimension, we assign some houses to state array.

For example, we would have a state array with six houses and a three-dimensional search space, where we assign two houses for dimension X and two houses for dimension Y and two houses for dimension Z (Figure 10).

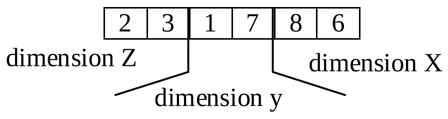

Figure 10. dimension x, y, z

Attention that order of assigned dimensions to the houses are arbitrary, but with change in position of factor in search space, movement is determined in direction of related dimension, and only corresponding cells with that dimension may be change in state array, and other values remain constant. So factors could move in direction of their corresponding dimensions.

Way of changing values is important, and is explained as follow:

When a factor starts to move in one direction, we divide each corresponding value with related dimension on distance, then by recalling neighborhood function, we determine that with which substituting in state array, number of queen crashes is decreased, or in other words, better mass is specified. Finally, if by substituting no new values, better state is achieved, mass moves to the last position of state array in related dimension.

It is obvious that if distance be zero, division is not done, and it means that two masses in search space have attracted each other, and placed in same position.

Now, after determining new position and state of each mass, new masses should be determined. So by recalling heuristic function, masses are evaluated.

```
h = 0;
for (j = 0; j <= 7; j++)
 {
  for (int l = j + 1; l <= 7; l++)
   {
    if (sta_arr[j] == sta_arr[l])
        { h++; break; }
   }
   for (int l = 1; l <= 7; l++)
    {
      if ((j + l) <= 7 & sta_arr[j] + l <= qu_num)
         if ((sta_arr[j] + l) == sta_arr[j + l])
             { h++; break; }
      if ((j + l) <= 7 & sta_arr[j] - l > 0)
         if ((sta_arr[j] - l) == sta_arr[j + l])
             { h++; break; }
    }
 }
```

```
if (h == 0)
    {
      for (j = 0; j <= 7; j++)
        {
          k = sta_arr[j];
          i = (j * 8) + k;
          switch_page(i);
        }
       break;
     }
```

Figure 11.

If a mass with no crashes is found, search has been finished. And switch_page function is called in order to puts queen icons in related houses. If still there is a factor with no crashes, calculation function is called once again, to accounts related calculations (force, acceleration, …) for new states and positions. These series of recalling would be continued continuously, until an answer (a factor with no crashes) is founded, or distance between masses became zero.

6. RESULTS AND PERFORMANCE DISCUSSION

In suggested solution, if dimension of search space was not equal to number of state array houses, two situations are occurred: a) if dimensions of search space be more than number of state array houses, we recommend that assign some dimensions to one house of state array , arbitrary. b) If dimensions of search space be less than number of state array houses, we suggest that assign some houses of state array to one dimension, arbitrary.

This fact that moving one factor just changes a part of state array, guarantees that we have a controlled search, and distances between masses are regulated base on gravitational algorithm. Also, this fact that changing part should improve factor mass, has been well-operated base on proximity principal, and guarantees that search is finding best states, and sudden changes don't make situation worse and don't waste time and space (Figure 12).

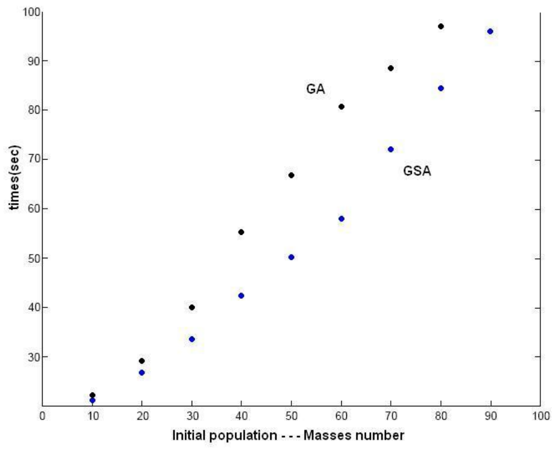

Figure 12. Genetics algorithm (initial population) and Gravitational search algorithm (masses number)

7. CONCLUSION

Random algorithms are widely used for optimization problems. These algorithms are often created by inspired from nature or known physical processes or creature behaviors. Since solving this algorithm with hilling algorithms and heat simulating has been widely comprehensive and has published in different articles and books, we, also solved our 8-ministers problem with two abovementioned algorithms. In comprising their performances with suggested algorithm, reasonable speed and implying optimizing works in suggested gravitational searching algorithm is obvious.

Gravitational searching algorithm is one of those algorithms which have been organized base on gravity rules in nature. We have used this new algorithm for offering suggested solution, and solving N-queen problem. Our suggested solution guarantees controlling situations and regulating them by corresponding dimensions of search space with state array houses, and improving search. We hope that by using reasonable changes in moving factor in search space and doing several compares, which are presented in Figure 3, could have taken small steps toward solving problems and optimizing search algorithms.

REFERENCES

[1] Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.(2009). GSA: A Gravitational Search Algorithm. Information Sciences,vol. 179, pp. 2232–2248

[2] Ahmed Tariq Sadiq, Dr. Hasanen S. Abdullah, Mohmed Natiq Fadhel. Tabu DNA Computing Algorithm to Solve N-Queens Problem.

[3] R. Douglas *Chatham (2009). Reflections on the N + k Queens Problem. College Mathematics Journal,pp.204-210*.

[4] Miguel A. Gutierrez-Naranjo, Miguel A. Martinez-del-Amor, Ignacio Perez-Hurtado, Mario J. Perez-Jimenez. Solving the N-Queens Puzzle with P Systems.

[5] Marko Božikovic, Marin Golub, Leo Budin (2003). Solving n-Queen problem using global parallel genetic algorithm. EUROCON 2003 Ljubljana,Slovenia.

[6] Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.(2010). BGSA:binary gravitational search algorithm. Natural Computing, vol. 9, pp. 727-745.

[7] Ellis Horowitz and Sartaj Sahni (1978). Fundamentals of computer algorithms. Computer Science Press Inc., Rockville, MD.

8/15/2012