International Journal of Artificial Intelligence,
ISSN 0974-0635; Int. J. Artif. Intell.
Autumn (October) 2011, Volume 7, Number A11

Ant n-Queen Solver

Salabat Khan, Mohsin Bilal, Muhammad Sharif, Rauf Baig

National University of Computer and Emerging Sciences,
Islamabad, Pakistan
{salabat.khan, mohsin.bilal, m.sharif, rauf.baig}@nu.edu.pk

ABSTRACT

Swarm intelligence and evolutionary techniques are heavily used by the researchers to solve combinatorial and NP hard problems. The n-Queen problem is a combinatorial problem which become intractable for large values of 'n' and thus placed in NP (Non-Deterministic Polynomial) class problem. In this paper, a solution is proposed for n-Queen problem based on ACO (Ant Colony Optimization). The n-Queen problem is basically a generalized form of 8-Queen problem. In 8-Queen problem, the goal is to place eight queens such that no queen can kill the other using standard chess queen moves. The environment for the ants is a directed graph which we call search space is constructed for efficiently searching the valid placement of n-queens such that they do not harm each other. We also develop an intelligent heuristic function that helps in finding the solution very quickly and effectively. The paper contains the detail discussion of problem background, problem complexity, Ant Colony Optimization (Swarm Intelligence), proposed technique design and architecture and a fair amount of experimental results.

Keywords: n-Queen Problem, 8-Queen Problem, Heuristic Techniques, Ant Colony optimization (ACO), Swarm Intelligence (SI).

1998 ACM Computing Classification System: I.2.1, I.2.8, I.2.11

1 Introduction

Ant Colony Optimization (ACO) is a meta-heuristic proposed in the early 90's which comes under the umbrella of swarm intelligence. It has been shown to perform well on many combinatorial optimization problems e.g. travelling salesperson problem (TSP), quadratic assignment, vehicle routing, connection oriented and connectionless network routing, sequential ordering, graph coloring, shortest common super sequence, single machine tardiness, multiple knapsack, etc. The problems in computer science can be grouped in different classes on the basis of time required to find a correct solution. The class 'P' contains all those decision problems for which polynomial time algorithms exist. The problems which can be solved in polynomial time on a non-deterministic computer are placed in 'NP' class. The non-deterministic computer is a theoretical computer that does not exist. This theoretical computer contains infinite amount of resources to spawn as many processes as there are possible solutions to a problem. However, note that solution to a problem in 'NP' can be verified in polynomial time (I. Martinjak et al., 2007). In order to tackle the problems in 'NP', heuristic techniques are usually used.

The n-Queen problem is also an intractable problem i.e. for large values of 'n' the problem cannot be solved in polynomial time and thus placed in 'NP' class. There are total $\binom{n^2}{n}$ number of possible

arrangements of 'n' queens on the chessboard (Crawford, 1992). For 8-queen problem, there are thus 4,426,165,368 possible arrangements of queens on board out of which only 92 are correct solutions.

The solution to the n-Queen problem proposed in this paper is based on ACO. The main idea is to use artificial ants to search for possible solutions of the problem at hand. The ants use information collected in past to direct their search and this information is available and modified through the environment. A fitness function is used to evaluate the quality of the solution constructed by an ant.

The remainder of this paper is organized as follows. In the next section, we review related research. In Section 3, we present the basics and the background of ant colony optimization meta-heuristic. In Section 4, the architecture of the proposed solution will be discussed. Subsequently, in Section 5, we present some simulation results to show the promising ability of our technique. Finally, Section 6 will conclude this work.

2 Related Work

The n-Queen problem is an interesting problem and a fair amount of research has been done to solve this problem. Rok Sosic et al. proposed two search techniques QS2 (Quick Search) and QS3 based on non-conflicting placement using optical processing elements for the solution of n-Queen problem (Sosic et al., 1991). An insight to the complexity of the search space in n-Queen problem using backtracking is discussed in (Bozinovski et al., 2004). K. D. Crawford proposed Genetic Algorithm (GA) in two different ways to solve the n-Queen problem (Crawford, 1992). Ivica et al. provided a comparison of different heuristic techniques in (I. Martinjak et al., 2007) to solve the n-Queen problem. The techniques include simulated annealing, tabu search and genetic algorithm. The n-Queen problem is not just a toy problem rather it can also be mapped to the maximum coverage problem. The maximum coverage problem is that, from any horizontal, vertical or diagonal direction, all '*n*' objects can be accessed without conflict (Sosic et al., 1991). A solution to the n-Queen problem is also a solution to the specialized case of maximum coverage problem as well. R. Eastridge et al. used genetic algorithm for the solution of n-Queen problem and also described the usefulness of this problem in the field of Artificial and Computational Intelligence (Eastridge et al., 2008).

To the best of our knowledge, there has been no application of the ACO to the n-Queen problem, so far In order to apply ACO, we first propose a search space. We then discuss a few modifications in the calculation of some parameters for ACO. We have added a few constraints in basic ACO as it cannot be applied directly to the n-Queen problem. A detail discussion on all these is provided in the subsequent sections.

3 Basics of ACO

Swarm intelligence is a branch of evolutionary computation, inspired by the natural world, which has been applied, with success, to several hard problems in artificial intelligence. ACO meta-heuristic has been developed by studying the collective foraging behavior of ants Individual ants perform relatively simple tasks but the entire colony of ants can collectively accomplish sophisticated work. For solving a problem through ACO, we need to describe the environment of the problem, a method to determine the fitness of the solution and a heuristic measure for the solution's component.

ACO is a multi-agent system; an ant behavior depicts the behavior of an agent in the system. The first ant algorithm was developed by M. Dorigo (Dorigo, 1992). Improvements in algorithm are made in (Gambardella et al., 1995 & 1996). Ants in nature modify their environment by constantly depositing a chemical substance called pheromone. The pheromone is used as an indirect communication among ants and guides them to find shortest path from their nest to a food place. If there are multiple paths to a food place, ants choose one with high concentration of pheromone. In Figure 1, one can logically find that how a shortest path will be concentrated with the high pheromone values as most of the ants on this path will come back quickly.

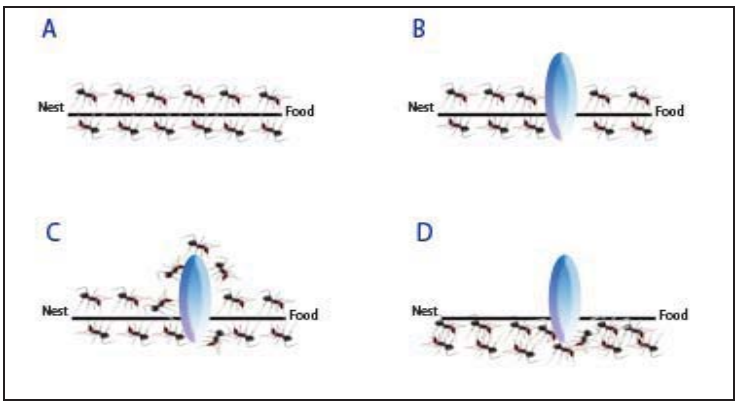

Fig. 1. Ants finding shortest path. a*)* no obstacle. *b)* obstacle placed. c*)* ants searching for shortest path. d*)* ants found shortest path.

3.1 Simple ACO Algorithm

Suppose, we have a connected graph *G = (V, E)* where |*V*| denotes the total number of nodes or vertices and |*E*| total number of connecting edges in graph. The simple ant colony optimization meta-heuristic can be used to find the shortest path between a given source node 'V_s' and a given destination node 'V_d' in the graph '*G*'. The path length is either given by the number of nodes on the path or summation of cost values on edges constituting the path. Each edge of the graph connecting the nodes 'V_i' and 'V_j' has a variable (artificial pheromone), which is modified by the ants when they visit the nodes (Gunes et al., 2002).

From a node, when an ant decides which node to move next, it uses two parameters to calculate the probability of moving to a particular node; first, distance to that node and second, amount of pheromone on the connecting edge. Let $d_{i,j}$ be the distance between the nodes *'i'* and *'j'*, the probability that the ant chooses *'j'* as the next node after it has arrived at node *'i'* where *'j'* is in the set '*S*' of nodes that have not been visited (Gunes et al., 2002).

$$p_{i,j} = \frac{[\tau_{i,j}]^{\alpha}[\eta_{i,j}]^{\beta}}{\sum_{k \in S}[\tau_{i,k}]^{\alpha}[\eta_{i,k}]^{\beta}}. \qquad (3.1)$$

Where $\tau_{i,j}$ is the pheromone value on edge *e(i, j) and* $\eta_{i,j}$ is a heuristic value calculated as $1/d_{i,j}$. The parameters *α* and *β* are influencing factors of pheromone value and heuristic value respectively. The pheromone at edges is usually initialized with small random values at start.

The complete route/ tour of an ant from a source node to a destination node is called a solution to the problem at hand. The evaluation of a solution is done using a fitness function. Some best ants (having good solutions) or all ants modify the pheromone values on the edges added to their tour. One possible modification of the pheromone may be done as:

$$\tau_{i,j} = \tau_{i,j} + \frac{Q}{L}. \tag{3.2}$$

Where *'Q'* is some constant and *'L'* is the length of the tour, small the value of *'L'* high the pheromone value added to the previous pheromone value on an edge.

With time, concentration of pheromone decreases due to diffusion affects; a natural phenomenon known as evaporation. This also ensures that old pheromone should not have a too strong influence on the future. So, with evaporation, chances to get stuck at local minima are minimized in ACO. This evaporation can be performed as:

$$\tau_{i,j} = \tau_{i,j} \cdot \rho \ \{\text{Where } \rho \text{ will be between } 0 \text{ and } 1\}. \tag{3.3}$$

4 N-Queen ACO

N-Queen problem is to place '*n*' queens on chessboard such that no queen can kill the other using standard chess queen moves. There are three such situations in which a queen can kill the other; first, if two queens are in the same row, second, if two queens are in the same column and third if one queen is in the diagonal of the other queen.

4.1 Solution/ Search Space

In order to solve the n-Queen problem, basic ACO is modified and some constraints are added. Moreover, the equations described above are also changed a little bit. The cores of this solution include "formation of search space" & "calculation of heuristic value". In Figure 2, formation of search space is described which makes the search efficient and fast.

The vertices in the search space are organized as a grid of n^2 *rows* * *n* columns with two extra vertices start and stop. Every vertex in a column is connected to all the vertices in the next column through directed edges except vertices in n^{th} column. The start vertex is connected with all the vertices in first column and all the vertices in the n^{th} column are connected with the stop vertex. The vertices label shows the chess cell position e.g. label 1 is used as a mapping to cell # [0, 0], label 2 is used as a mapping to cell # [0, 1] and n^2 is used as a mapping to cell # [n-1, n-1]. For 8-queen problem or 8*8 chess board positions, *n* will be 8, above gird will have 64*8+2 vertices, 64*64*7+2*64 edges and n^2 i.e. 64 will map to cell # [7,7] at chess board.

In order to simplify the things, we will consider *n* equal to '8' i.e. we will now talk about 8-queen problem, as from the search space, it is evident that solution to 8-queen problem is easily extendible to n-queen problem.

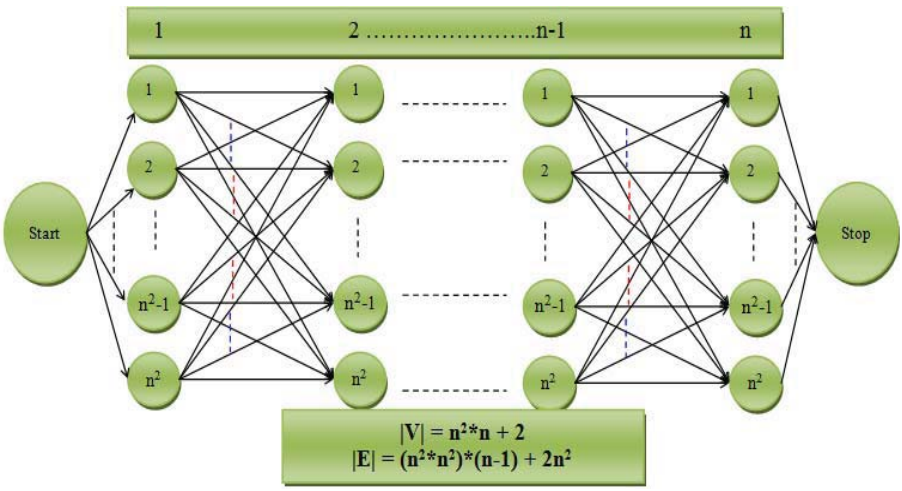

Fig. 2. Search Space.

4.2 Constraints on Ant Movement

The tour of an ant begins from start vertex and ends at the stop vertex. An ant can only move forward (from left to right). It first picks a node in the first column, then the next node from the second column, and so on. Once a node is selected from a column, the ant can pick the next node only from the column next to the current column. Tour ends on the last column. An ant during a tour visits only '*n*' nodes (8 for 8-queen). The labels of nodes in the tour cannot be the same. In other words, no two nodes in a tour will be in the same row of the search space. A selected node in a tour represents a cell of chessboard where a queen is to be placed. The prohibition of selecting more than one node with the same label assures that only one queen will be placed on a cell on the chessboard.

4.3 Parameter Initialization and Transition Equation

Initially, all the edges are assigned with small and equal pheromone values. A user defined number of ants (swarm size) are created and moved in the search space one by one Pheromone value is modified on the basis of fitness value of a solution found by an ant.

Fitness Value: The fitness represents the number of positions at chessboard that satisfy the game constraint, i.e. queens if placed on these positions will not kill any other queen at chessboard. In case of 8-queen problem the possible range of fitness values is 0-8.

```
Runs = 0;
Fitness_Achieved = 0;
Valid_Solution_Found = 0;
/*TotalRuns = 50 used during experiments*/
WHILE(Runs < TotalRuns)
   Initialize pheromone values on all the edges
   BestAnt = {};                                   /*Storing the best solution, currently empty*/
   Solution_Found = FALSE;
   Iterations = 0;
   WHILE(Iterations < Total_Iterations_Allowed && Solution_Found == FALSE)
      t = 1                                                  /*Count for ants*/
      REPEAT
         REPEAT
            Calculate heuristic values for the vertices in column, next to the current column
            Select a vertex based on equation 3.1
            Add vertex to the partial tour of Ant_t
         UNTIL (Ant_t   Tour_is_Completed)

         IF(  Fitness(Ant_t) > Fitness(BestAnt)  )
            BestAnt = Ant_t
          /* Max_Possible_Fitness = Number of queens e.g. for 8-queen, it will be equal to 8*/
            IF(  Fitness(BestAnt) == Max_Possible_Fitness )
               Valid_Solution_Found++;
               Solution_Found = TRUE;                    /*Strat Next Run*/
            END IF
         END IF
      UNTIL (t ≥ No_of_Ants || Solution_Found == TRUE)      /*No_of_Ants = Swarm Size*/
      Update Pheromone for 't' number of ants
      Perform evaporation for 't' number of ants
   END WHILE
   Fitness_Achieved = Fitness_Achieved + Fitness(BestAnt);
   Runs++;
END WHILE
Average_Fitness_Achieved = Fitness_Achieved / TotalRuns;
Average_Perfect_Covergence = Valid_Solution_Found / TotalRuns;
```

Fig. 3. n-Queen ACO Algorithm.

Heuristic Value and Evaporation: The probability equation is same as described above but the calculation of heuristic value is changed. Now, in Equation (3.1), heuristic value is not based on the distance between two vertices, rather it is the inverse of the number of contradictions plus one, if node '*j*' is selected as next node. Contradictions represent the positions at chessboard where if queens are placed will kill each other. Note that node '*j*' is a chess position at which a queen will be placed.

Furthermore, value of α and β is initialized with a random number generated in the range 0-2. The evaporation is done on constant rate in our solution, decaying the pheromone value on all the edges is done after the completion of a single iteration. A single iteration is completed when all the ants complete their tour. The evaporation is done by subtracting the current pheromone value of an edge from a small constant. The complete n-Queen ACO algorithm is described in Figure 3.

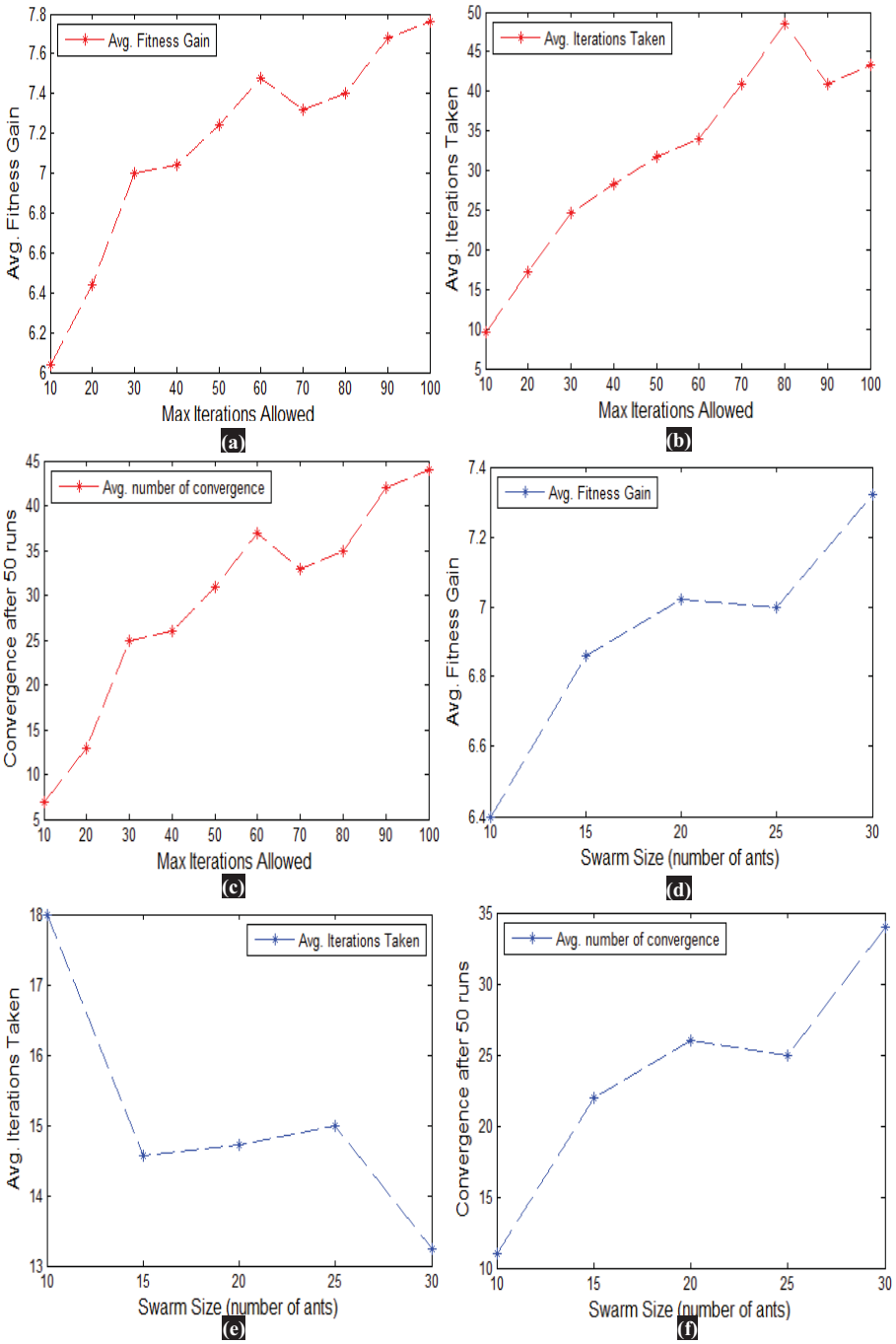

Fig. 4. Simulation Results – I.

5 Simulation Results

Ant n-Queen Solver has following user defined parameters:

Number of ants , total Runs, iterations allowed, the powers of pheromone and heuristic values (α, β) in the probabilistic selection (Equation 3.1) & the pheromone evaporation rate (used in Equation 3.3)

Ant n-Queen Solver has following output:

Average Fitness Gain, different best positions found during a run, iterations taken per run, average Iterations Taken & average Convergence Rate

The ACO is a probability based algorithm, so, the results it would generate will be different if run multiple times on the same instance of a problem. So, in order to generate simulations results, we run it 50 times for 8-queen problem and then take the average of the generated output. The proposed technique is implemented in Microsoft Visual Studio 2008 C# and run on Intel P-IV processor with 1 GB of Ram. In Figure 4 (a), average fitness is drawn against maximum iterations allowed. Note that, if we keep increasing the maximum iterations allowed, we also get improving average fitness. In Figure 4 (b), relationship between average iterations taken and max iteration is shown. We used two termination criterions; one application is terminated if we get maximum fitness i.e. 8 in case of 8-queen problem, second, if max allowed iterations are over. It's possible that we get the correct solution before the maximum iterations are over. So, the average number of iterations taken shows the early convergence of the algorithm in certain tests out of 50 total tests. Some parameters are kept constant as swarm size (number of ants) = 10, alpha and beta = 1 for the Figures 4 (a), (b) and (c).

In Figure 4 (c), perfect convergence is shown against maximum iterations allowed. Perfect convergence mean that the correct solution is found (fitness = 8). The behavior is self descriptive that as long as we keep increasing the number of maximum iterations allowed, the perfect convergence is also increase.

For Figure 4 (d), (e) and (f), parameters which are kept constant are as maximum iterations allowed = 20, alpha and beta = 1. The relationship between swarm size and average fitness gain is depicted in Figure 4 (d), as expected if we keep increasing the swarm size the average fitness gain will also become better. In Figure 4 (e), as shown prefect convergence also increases if swarm size is increased. It’s also natural that if swarm size is increased, the number of perfect convergence will increase accordingly as shown in the Figure 4 (f).

For Figure 5 (g) and (h), parameters which are kept constant are as maximum iterations allowed = 20, swarm size = 10 and beta = 1. The average fitness gain and perfect convergence are found best on alpha value 1.5 as shown in Figure 5 (g) and (h), respectively. Both of the curves are declining after the alpha value 1.5, so, it may be considered as a saturation point if the value of beta is 1.

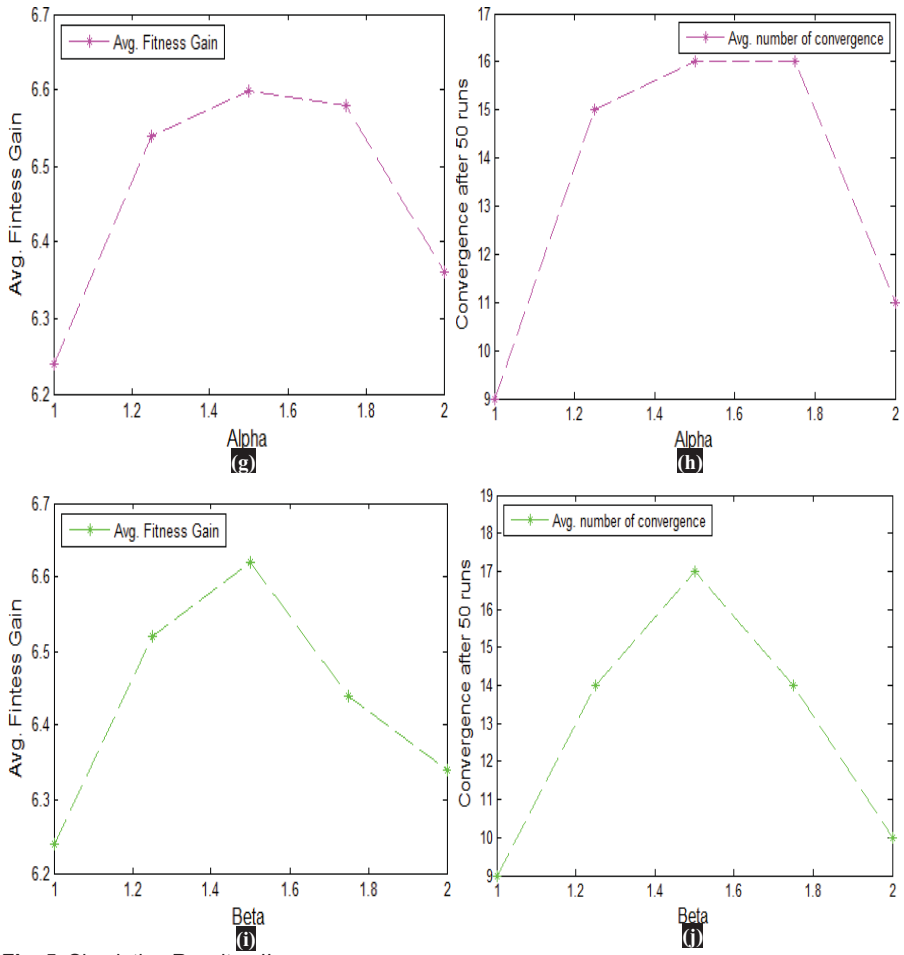

Fig. 5. Simulation Results - II.

For Figure 5 (i) and (j), parameters which are kept constant are as maximum iterations allowed = 20, swarm size = 10 and alpha = 1. The average fitness gain and perfect convergence are found best on beta value 1.5 as shown in Figure 5 (i) and (j), respectively. Both of the curves are declining after the beta value 1.5, so, it may be considered as a saturation point if the value of alpha is 1.

6 Conclusion

In this paper, a solution to the n-Queen problem is proposed based on Ant Colony Optimization. A DAG (Directed Acyclic Graph) is modeled as a search space to assist the ants search, efficiently. The fitness function is very simple to evaluate the candidate solution in terms of time complexity. The solution is working efficiently for 8-queen problem and it can easily be extended to large values of '*n*' because of the simplistic model of the search space. We found our solution working efficiently for 8-queen if parameters are set as swarm size = 15, alpha = 1, beta = 1.5 or alpha = 1.5 and beta = 1.

We conclude that the ACO can provide better solution in reasonable amount of time for combinatorial optimization problems and as future work we will explore its applicability to some other such problems.

Acknowledgement

The authors would like to acknowledge Higher Education Commission (HEC) of Pakistan for their continuous support to pursue our Ph.D. research under the indigenous Ph.D. Program. It would have been impossible to complete this effort without their continuous support. The authors are also thankful to the reviewers for their valuable comments which improve the quality of this research work.

References

A. Bozinovski and S. Bozinovski, 2004, *N-queens pattern generation: an insight into space complexity of a backtracking algorithm*, Proceedings of the 2004 International Symposium on Information and Communication Technologies, Las Vegas, NV, USA, 281-286.

K. D. Crawford, 1992, *Solving the N-queens problem using genetic algorithms,* Proceedings of ACM/SIGAPP Symposium on Applied Computing, Kansas City, KS, USA, 1039-1047.

M. Dorigo, 1992, *Optimization, Learning and Natural Algorithms*, PhD thesis, Politecnico di Milano, Italy.

R. Eastridge and C. Schmidt, 2008, *Solving n-queens with a genetic algorithm and its usefulness in a computational intelligence course*, Journal of Computing Sciences in Colleges, 23 (4), 223-230.

P. Engelbrecht, 2007, *Computational Intelligence: An Introduction*, Second Edition, John Wiley & Sons, England.

L. M. Gambardella and M. Dorigo, 1995, *Ant-Q: A reinforcement learning approach to the TSP*, Proceedings of Twelfth International Conference on Machine Learning, Tahoe City, CA, A. Prieditis and S. Russell (Eds.), Morgan Kaufmann, 252-260.

L. M. Gambardella and M. Dorigo, 1996, *Solving symmetric and asymmetric TSPs by ant colonies*, Proceedings of IEEE International Conference on Evolutionary Computation, Nagoya, Japan, 622–627.

M. Gunes, U. Sorges and I. Bouazizi, 2002, *ARA - The ant-colony based routing algorithm for MANETs*, International Workshop on Ad Hoc Networking (IWAHN 2002), Vancouver, BC, Canada, 79-85.

I. Martinjak and M. Golub, 2007, *Comparison of heuristic algorithms for the N-queen problem*, Proceedings of the ITI 2007 29th International Conference on Information Technology Interfaces, Cavtat, Croatia, 759-764.

R. Sosic and J. Gu, 1991, *Fast search algorithms for the N-queens problem*, IEEE Transactions on Systems, Man, and Cybernetics, 21 (6), 1572-1574.