A New Solution for N-Queens Problem using Blind Approaches: DFS and BFS Algorithms

**Farhad Soleimanian
Gharehchopogh**
Computer Engineering
Department, Science and
Research Branch, Islamic Azad
University,
West Azerbaijan, Iran

Bahareh Seyyedi
Computer Engineering
Department, Science and
Research Branch, Islamic Azad
University,
West Azerbaijan, Iran

Golriz Feyzipour
Computer Engineering
Department, Science and
Research Branch, Islamic Azad
University,
West Azerbaijan, Iran

ABSTRACT

The $N \times N$ queen's puzzle is the problem of placing N chess queen on an $N \times N$ chess board so that no two queens attack each other. This approach is a classical problem in the artificial intelligence area. A solution requires that no two queens share the same row, column or diagonal. These problems for computer scientists present practical solution to many useful applications and have become an important issue. In this paper we proposed new resolution for solving n -Queens used combination of depth first search (DFS) and breathe first search (BFS) techniques. The proposed algorithm act based on placing queens on chess board directly. This is possible by regular pattern on the basis of the work law of minister.

The results show that performance and run time in this approach better then back tracking methods and hill climbing modes.

Keywords

N-queen's, Pawn, Depth First Search, Breathe First Search.

1. INTRODUCTION

The n -queens problem introduced in 1850 by Carl Gauss and has been studied for many decades by scientists. The n -queens problem [7] is an effort to find a placement of n queens on an n by n chess board so that no two queens attack each other [1]. Today, this problem has opened as a research field among of practical and engineering applications such as traffic control, VLSI testing, parallel optical computing, communications and etc. Classical methods to solve n -queens problem contain search heuristic methods, local search and conflict minimization methods [1, 2]. However, it has been considered new artificial intelligence (AI) techniques and strategies to solving n -queens problem versus traditional methods to solve this problem in recent years.

In the literature, there are many researches in this domain. In classical approach, backtracking algorithms are used for solving n -queens problem that make all possible resolutions [1, 4] widely. The backtracking algorithms generate the solution vector one component at a time and then test it. According to the criterion function to define whether the vector being formed still has a chance of success [1]. However, backtracking search is not able to solve the large size n -queens problem [9].

Solutions to the n -queens problem can be performed as an n -tuples $(q_1, q_2, q_3, \dots, q_n)$. Each queen must be on a different row and column so that no two are in the same row, column or diagonal. Finding resolution for n -queens problem composed of three types. First type is an effort to acquire new resolution.

Next type is finding a family of resolutions and last type is finding all resolutions. Empirical observations indicate that n is increasing lead to the number of solutions increases exponentially [2].

The paper deals with n -queens problem and present new resolution for it. The paper structured as follows: in the next section, we review the previous works in the literature. Section 3, is proposed new resolution for solving n -queens problem based on AI techniques. Section 4, describes the results of comparisons and evaluation. In the section 5, we conclude the results of computing in this paper.

2. REVIEW LITERATURE

In the body of literature, various researches are done to overcome n -queens problem. For example, Sosic and Gu [3] proposed an novel local search algorithm with conflict minimization to solve the n -queens problem. This algorithm is significantly faster than any backtracking search algorithm and is capable of solving the n -queens problem in linear time. Backtracking algorithms offers a very limited class of resolutions for great size boards. Because, it is difficult for a backtracking search to detect resolutions that are separate in the solution space considerably [3].

Reference [4] presents probabilistic algorithm utilizes gradient-based heuristic search. This algorithm is able to generate a solution for extremely large values of n . In this algorithm, a random permutation generator is used to place the queens on the board and reduce the number of collisions by swapping a pair of queens. The results indicate that the number of permutations necessary to build a resolution is usually very small [4]. Also, reference [5] defines perspective of n -queens problem and presents the classification of applied algorithms for the n -queens problem into three categorizations. Manzuik [6] implemented a binary Hopfield neural network (HNN) in the asynchronous mode as well as in the synchronous to solving n -queens problem. The HNN is a simple artificial neural (ANN) network. It is able to save specified patterns in a manner similar to the brain in that the complete pattern for a given problem can be recovered if the network is indicated with only regional information [1]. The results show that convergence rate is up to 100% and the experimental predicate of the average computational complexity is polynomial [6].

Various models of ANN such as HNN are able to adapt and learn from input information in optimization problems such the n -queens problem and many more. Also, Shagrir [10] proposed a new algorithm by means of AI techniques based on HNN. In this HNN, every unit (neurons) is permissive to prevent itself. The results indicate that the HNN never

continually oscillates but relaxes into a resolution in polynomial time and it solves the n-queens problem regardless of the dimension n or the initialized values. In addition, in reference [8], has been presented a new algorithm by global parallel genetic algorithm to solve n-queens problem. Also, reference [9] presents a new probabilistic algorithm that it is based on gradient based heuristic. It is capable to find a resolution for millions of queens and runs in polynomial time with a local search.

3. PROPOSED METHOD

The goal of this paper is implementation DFS, BFS recursive function for n-queen problem. Then, we compare two algorithms base on run time and used memory. First we give some information about DFS and BFS algorithms:

3.1 Depth First Search (DFS)

Suppose that undirected graph $G(V, E)$. We want to find available as a special node V . Assuming that start the search from node V , first we met node V then such node W that already have not been met and the second adjacent to node V is chosen and continue the depth-search with node W . finally our search lead to find a node like U that there isn't any node that not met in the list of adjacent (proximity) in this step we need, return and chose another node if there are selection and met it, and the same goes the process up again at the start of the node V is reached and the algorithm ends when received the node V and not met a neighbor. This algorithm is shown in the Table 1.

Table 1: DFS Algorithm

```
Alg DFS(v){
Visit (v);
For (each vertex w adjacent from v)do
If(visit[w]==false)then DFS(w);}
```

For DFS, we used stack that its quasi codes are shown in the Table 2.

Table2: Quasi Code for DFS Algorithm

```
1)Create an array for store nodes(UN)
2)Crate an array for store results(DFS_results)
3)Create first node for start
4)Add this node to array UN
5)Create child's of this nodes
6)Delete this node from array and check for being answer
7)If result is true
    7-1)Add this node to results array
    7-2)Add this node's Childs to array(UN)
8)Else
    8-1)Delete this node from array
9)Repeat steps 4-8 for start node's first child
```

Fig 1 shown the output.

Fig 1: Output of the Program

3.2 Breadth First Search (BFS)

This approach is strategy for searching in a graph when limited to essentially two operations:

- visit and inspect a node of graph
- gain access to visit the node that neighbor the currently visited node

The BFS begins at a root node (V) and inspect all the neighboring nodes. Then for each of this neighbor in turn, it inspects their neighbor nodes which were unvisited and so on (Table 3).

Table3: BFS Algorithm

```
Alg BFS(v){
U=v
Visit(v);
Repeat {
For(all vertex w adjacent from u) do {
If(visit[w]==false)then {
AddQ(W);
Visit(w);
}
}
If (Queue is empty ) then return;
Delete Q(u);
Until(false);}
}
```

For implementation this method we can to use queueas shown in Table 4.

Table4: Quasi Code for BFS Algorithm

1) Create an array for store nodes(UN)
2) Create an array for store results(BFS_results)
3) Create first node for start
4) Add this node to array UN
5) Create child's of this nodes
6) Delete this node from array and check for being answer
7) If result is true
7-1) Add this node to results array
7-2) Add this node's child's to array(UN)
8) Else
8-1) Delete this node from array
9) Repeat steps 4-8 for start node's child's

Table 5 is shown the quasi code for checking corrected node.

Table5.Quasi code for checking corrected node

Check_For_Being_Answer
1)repeat steps 2-3 for number of queens
2)if new queen has canted or Columnar threat with past queens
2-1)Return false as result for main function(DFS or BFS search)
3)else
3-1)Return true as result for main function(DFS or BFS search)

4. RESULT AND DISCUSSION

In BFS algorithm we just keep a tree (the breath first search tree), a list of nodes to be added to the tree, and marking on the vertices to tell whether they are in the tree of list. Depth first search is another way of algorithm which is closely related to preorder traversal of a tree. Recall that preorder traversal simply visits each node before its children. It is most easy to program as a recursive routine. Maximum needed time and space for this algorithms respectively $S(n) \in O(n)$, $T(n,e) \in O(n+e)$ and if we use adjacency matrix $S(n) \in O(n^2)$, $T(n,e) \in O(n.n)$. (n: number of nodes and e number of graph edge).

In Table 6 we used DFS and BFS algorithms for 8-queens in 10 different times:

Table6.DFS and BFS algorithms for 8-queens (number of runs are 10)

AlgorithmBFS	AlgorithmDFS	
Time	Time	Period execute
11.4036	0.4524	1
12.0432	0.468	2
11.5908	0.4212	3
12.5736	0.4212	4
11.8404	0.4212	5
12.0276	0.4212	6
12.2772	0.468	7
12.0588	0.4992	8
12.3552	0.4992	9
11.8404	0.4368	10
120.0108 / 10 = 12.00108	4.49280 / 10 = 0.44928	Average time

16777216	57	maximum length Un
3696597	1485549	Number of extracted node

In Table 7, we compare to BFS and DFS time for deferent queens:

Table7.BFS and DFS time comparison for different queens

BFS		DFS		Number of queens
Number of extracted node	Time	Number of extracted node	time	
200	0	155	0	4
1140	0	451	0	5
22092	0.03	15316	0.01	6
182609	0.870	52914	0.03	7
3696597	16.140	1485549	0.452	8
63328349~	276.502~	14226688	3.884	9
1637426900~	7149.275~	286609046	82.245	10
35663958826~	155714.706~	6276661887	1849.959	11
1098170729697~	4794793.902~	160422261853~	47282.235~	12
1537228672809129301~	6711793045511.638~	165547703225598540~	48792887702.802~	16

According to up results, we can see the average run time to reach the first answer in DFS algorithm 0.44928 second while this time for BFS algorithm 12.00108 second. Therefore, the DFS algorithm is quicker than BFS algorithm. Number of extended node in DFS algorithm is less than BFS algorithm therefore the needed memory for BFS is more than DFS.

Fig 2:View of DFS algorithm's performance in solving N-queens problem

Fig 3:View of BFS algorithm's performance in solving N-queens problem

5. CONCLUSION AND FUTURE WORK

In this paper, we proposed new blind algorithm for solving n-queens using combination of DFS and BFS methods. The proposed algorithm act based on placing queens on chess board directly. The results show that performance and run time in this approach better then back tracking methods and hill climbing modes. As a result, the DFS algorithm is quicker than BFS algorithm and Number of extended node in DFS algorithm is less than BFS algorithm as well as the required memory for BFS is larger than DFS. Using this method, time and cost of solving n-queens problem is minimized in comparison of old methods. As future work, we can also find more efficient algorithms to solving n-queen problem based on other AI techniques.

6. REFERENCES

- [1] M. G. Kaosar, M. Shorfuzzaman, S. Ahmed, "A Novel Approach to Solving N-Queens Problem". In Proceedings of the 8th World Multi conference on Systemic, Cybernetics and Informatics (SCI 2004), Orlando, Florida, USA, pp.1-5, June 2004.
- [2] R. Sosi and J. Gu. "Efficient local search with conflict minimization: A case study of the n -queens problem".IEEE Transactions on Knowledge and Data Engineering, Vol. 6, No. 5, pp. 61-68, 1994.
- [3] R. Sosic and J. Gu. "Efficient local search with conflict minimization: A case study of the n -queens problem". IEEE Transactions on Knowledge and DataEngineering, Vol. 6, No. 5, pp. 61-68, 1994.
- [4] H.S. Stone and J.M. Stone. "Efficient search techniques - An empirical study of the n -queens problem". IBM Journal of Research and Development, 31: 464-474, 1987.
- [5] C. Erbas, S. Sarkeshik, and M. M. Tanik. "Different perspectives of the n -queens problem". In Proceedings ofthe ACM 1992 Computer Science Conference, 1992.
- [6] J. Mandziuk. "Solving the n -queens problem with a binary Hopfield-type network synchronous and asynchronous model". Biological Cybernetics, Vol. 72, No. 5, pp. 439-446, 1995.
- [7] C. Letavec, J. Ruggiero, The n-Queens Problem, INFORMS Transactions on Education, Vol. 2, No. 3, pp. 101-103.
- [8] M. Bozikovic, M. Golub and L. Budin. "Solving n -queens problem using global parallel genetic algorithm".EUROCON, Vol 2, pp 22-24, 2003.
- [9] R Sosic and J. Gu. "A polynomial time algorithm for the n -queens problem". SIGART, 1(3): 7-11, 1990.
- [10] O. Shagrir. "A neural net with self-inhibiting units for the n -queens problem". International Journal of Neural Systems, Vol. 3, No. 3, pp. 249-252, 1992.