Quest Journals
Journal of Software Engineering and Simulation
Volume 8 ~ Issue 9 (2022) pp: 16-22
ISSN(Online) :2321-3795 ISSN (Print):2321-3809
www.questjournals.org

Research Paper

Advance Search Algorithm: A configurable algorithm using a combination of local search and systematic search method

Priti Adinath Shinde
(M. tech student, BITS, Pilani WILP Division)
(Vambori, Maharastra, India)

ABSTRACT: *In this paper, local search and systematic search methods, their advantages and disadvantages are discussed, and a new advanced search algorithm, "The Third Eye" is introduced. The Third Eye uses the benefits of local search and systematic search techniques and tries to avoid the disadvantages it carries. It's a parametrized algorithm where 2 of its parameters help configure the search process based on the user's system. This motivation is taken from Artificial Neural Network basics. An ANN is based on a collection of connected units or nodes called artificial neurons [2]. These nodes are configurable based on the user's system configuration. Users with higher system configurations tend to choose larger nodes to get the optimal solution. The same concept is used here while parameterizing the algorithm, so based on the user's configuration user will try to reach the near-optimal solution.*

KEYWORDS: *The Third Eye, Exact Search/Systematic Search, Local Search, Near-optimal solution*

R*eceived 04 Sep., 2022;* ***R****evised 17 Sep., 2022;* ***A****ccepted 19 Sep., 2022 © The author(s) 2022.*
P*ublished with open access at* ***www.questjurnals.org***

I. INTRODUCTION

Search algorithms are one of the most important areas of Artificial Intelligence. In Artificial Intelligence, Search techniques are universal problem-solving methods [3].

Based on the search problems, we can classify the exact search algorithms into uninformed (Blind search) search and informed search (Heuristic search) algorithms [3]. In this paper, we are going to use an uninformed search algorithm.

Uninformed search is a class of general-purpose search algorithms that operates in brute force-way [3]. Uninformed search algorithms do not have additional information about the state or search space other than how to traverse the tree, also called blind search [3].

Now, another essential algorithm that is used in The Third Eye algorithm is the local search algorithm. In this particular algorithm, we are just randomly going to generate the state. Then we can use the objective/evaluator function to check the result of the randomly generated state. This objective or evaluator function checks the state's correctness generated by local search.

Note: It's not mandatory to use only random search as a local search algorithm; users can also try to use other local search algorithms that are available and suitable to their problem statement.

While local search has several disadvantages, its advantages can't be ignored. It is a good algorithm for many real-time problems, but at the same time, local search algorithms are purely randomized algorithms. If you're lucky, you get the global optimal solution, but there is no guarantee to get the global optimal solution all the time. There could be a possibility that we are somewhere near to the global optimal solution, but just because we keep checking on the next best move, we miss the global optimal. You can imagine climbing a mountain; wherever you are, you will reach the highest point of that particular path.

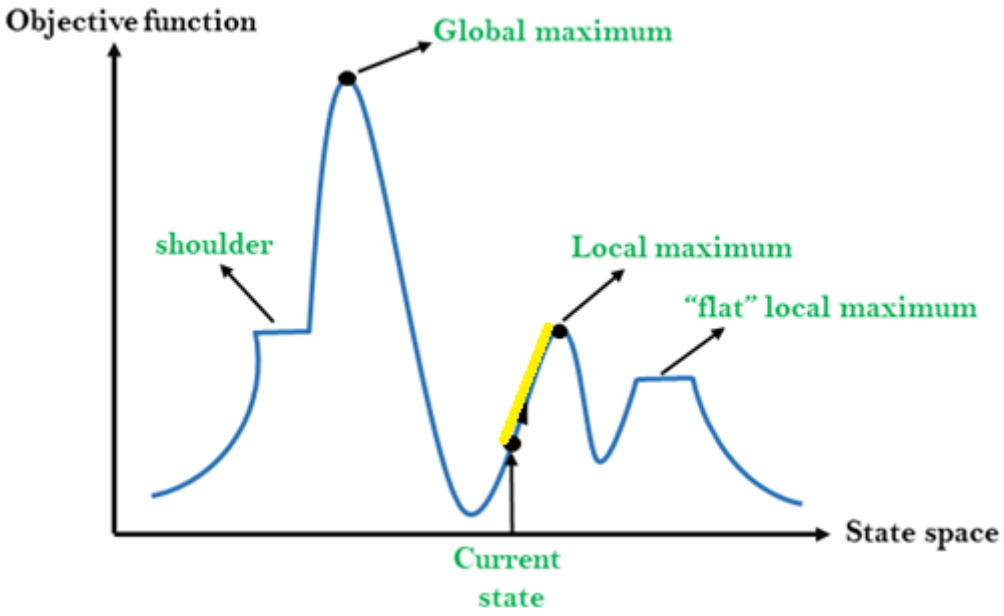


Image Source: https://www.javatpoint.com/hill-climbing-algorithm-in-ai

In this case, we were close to the global optimal solution, but we missed it because we greedily followed steps that kept giving us higher and higher values for the objective function of the problem.
The intuition behind the TheThirdEye algorithm is to address this problem statement.

II. THE THIRD EYE ALGORITHM

Instead of relying on complete randomness or avoiding systematic search due to memory and time complexity, the third eye algorithm keeps an eye on both the results. It produces the final result out of these two algorithm results. The third eye name comes from doing the third observation as a final result after using two significant observations, one observation of local search result and another one of systematic search result.

This paper also proposes a combination of 2 ideas but in such a way that the algorithm is parameterized based on the user's system so we can move towards global optimum without facing time and space complexity issues.
The Third Eye addresses the problem mentioned in the introduction section.

Note: For a better understanding of the algorithm, I have used the N Queens problem explained with search algorithms in the book 'Artificial Intelligence A modern approach' by Stuart J. Russel and Peter Norvig. I will use the same problem to present the third eye algorithm.

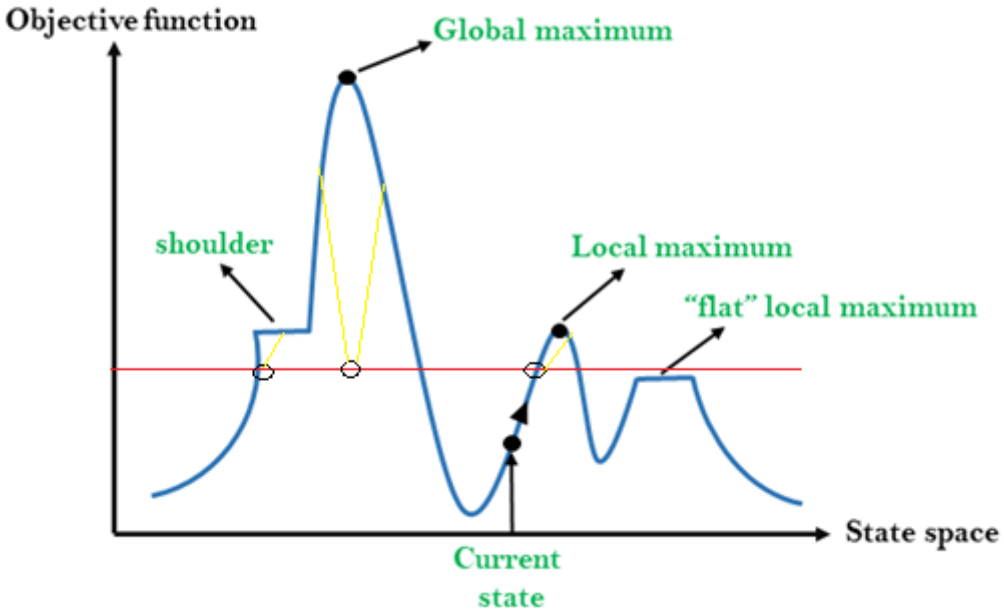
The simple idea here is to jump at mid of the mountain using a local search but then from that point, start building a tree that will explore all the further possibilities and gives the best result.

Following are the benefits of jumping at mid of the mountain using a local search:

1. Lesser state space to search as here we are considering only to solve half of the problem, so easier to find the global optimal solution for the local search problem. For example, In the eight queens' problem, the user can choose to use a local search for placing four queens on board. So, state space for four queens will be lesser as compared to 8 queens, and it will be easy to find the nearest optimal for four queens' problem.
2. If we are not at the global optimal solution for half of the problem, we can still further expand it as a tree and get the best possible optimal result, making us reach the nearest optimal solution.

It will be up to the user to choose algorithm parameters based on their system configurations. The higher the configuration, the more significant percentage of the problem the user can choose to solve using a systematic search approach. For example, In the eight queens' problem, If the user's systems configuration is relatively reasonable but not so good to solve the eight queens' problem fully, then, in that case, the user can choose to solve the three queens or four queens problem using local search and remaining queens using systematic search.

In the below diagram, the red line represents the objective function threshold we will provide as input. The black dots are the solution given by the local search for some percentage of the problem (this value is explained in more detail in the later step) with the given threshold of the objective function. In the below example, I have shown the 3 sample states with the same value as the objective function threshold, but while doing a local search, we might even get a higher objective function value than the given threshold. So, here we made a jump on the mountain using local search, but then the yellow lines represent the further tree that is expanded from the chosen state using exact search algorithms.



This way, we are less likely to miss the global maxima if our chosen random state has a value same or equal to the threshold of the problem and that state further expands to global maxima via any path.

There are two approaches to the TheThirdEye algorithm:

1) Randomization- Systematic search approach
2) Systematic search- Randomization approach

Randomization- Systematic search Approach:

Algorithm:

Function TheThirdEye(Random_Search_Threshold, n_top_instances, Random_Search_Iteration, Optimizer_function) **returns** a solution state

Inputs:

Random_Search_Threshold: Percentage of the problem solved using random search.
The remaining percentage of the problems will, by default, become a systematic search problem.
Random_Search_Iteration: Number of iterations we want to perform to do Random Hill climbing.
Optimizer_function: Use the optimizer function for Random Hill Climbing to check the optimizer value and if it's n_top_instances, then save the state.
n_top_instances: Choose the number of top instances you want to build and perform a systematic tree search.

```
Best_Solutions = []

For Random_Search_Iteration:
   objective_function_value = Random_Hill_Climbing(Random_Search_Threashold)

   If objective_function_value >= Random_Search_Threshold:
       Add objective_function_value _state in Best_Solutions
```

```
For n_top_instances:
    tree_solution  = Generate_Remaining_Tree_State(n_top_instance)

    If tree_solution > Best_Solutions[n_top_instance]:
       Return tree_solution
    Else:
       Return Best_Solutions[n_top_instance]
```

There are two major steps taken in this algorithm.
Step 1:
Choose the percentage of problems solved using the local search problem. For example, in the n queens' problem, if we solve the 4-queen problem, we may decide that out of placing four queens on board, we can place only two queens and solve it using local search.
In this scenario, the state place will be lesser than the four queens' problem. We will use Random State Generator as a local search algorithm. It will randomly generate states where two queens are placed on the board and evaluate them using an objective function.
In this 1st step, two parameters are configurable:
1) The percentage of problems to be solved using a local search.
2) Number of top best instances to keep as a result of local search.

To relate this idea to any use cases, you can think of a problem statement and its systematic tree search solution.

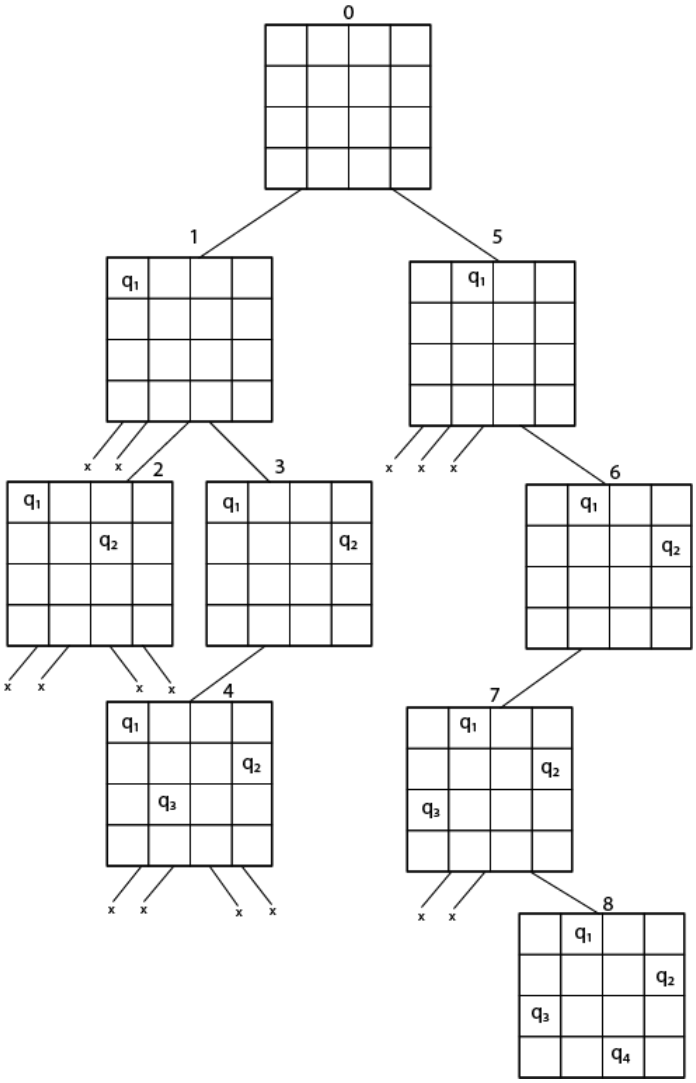
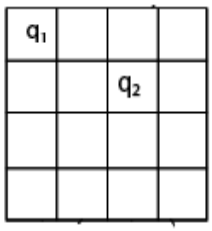


Image Source: https://www.javatpoint.com/n-queens-problems

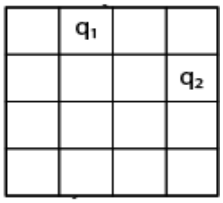
In the above tree diagram, one may think that percentage of the problems solved will be the layer of the tree. So, in the 4-queen problem, we choose this parameter value as 2, so we have to perform a local search till the 2nd layer of the systematic tree search.

Step 2:
In this step, we can use the top best solutions kept for taking it forward.
For example, we kept the three best solutions from the earlier step, now, we will pick the first one, and then we will expand the tree from that solution. So, we have a local search solution for four queens' problems till two queens, now, for 3rd queen, we will place the third queen on board in all possible combinations, and the same we will do for 4th queen. We will place the 4th queen also in all possible combinations, and then we will return the state with the highest objective value.
We will keep expanding the best solution one by one until we get global optimal results, or after completion of tree expansion of all best solutions, we can return the best out of all the solutions.
Based on the system configuration and time requirement, the user can choose the value of n_top_instances.
Following is the illustration of the example mentioned earlier:
If we choose to solve the four queens problem using The Third Eye algorithm, then we may select solving two queens using local search, and the following are the three best states we get after performing a local search on the two queens.

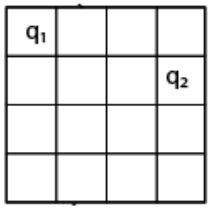
1)



2)

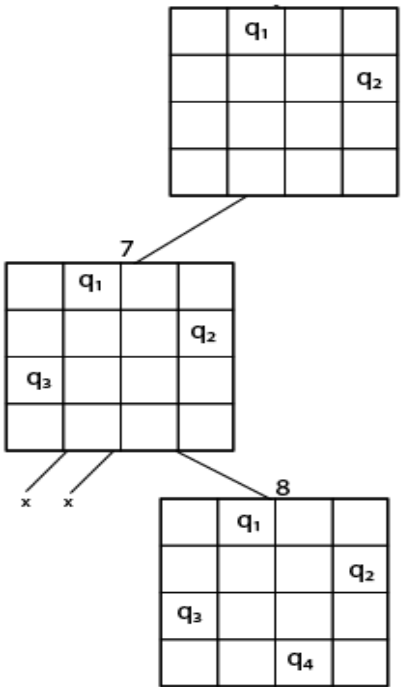


3)



Now, we can pick these solutions one by one and will generate a further tree.
Let's, pick the 3rd solution and generate its further tree with all possible combinations now.

It's one of the paths that would look like below, giving the global optimal solution.



After completing the paths' expansion, we can return the resulting state with the highest value for its objective function.

Systematic search-Randomization Approach:
This approach can perform exactly the reverse operations of what we did in the Randomization- Systematic search Approach.
Steps:

1) Do a systematic search for the percentage of level we want.
2) Do a local search on the remaining part.

Algorithm:

Function TheThirdEye(Random_Search_Threshold, n_top_instances, Random_Search_Iteration, Optimizer_function) **returns** a solution state

Inputs:

Random_Search_Threshold: Percentage of the problem solved using random search.
The remaining percentage of the problems will, by default, become a systematic search problem.
Random_Search_Iteration: Number of iterations we want to perform to do Random Hill climbing.
Optimizer_function: Use the optimizer function for Random Hill Climbing to check the optimizer value and if it's n_top_instances, then save the state.
n_top_instances: Choose the number of top instances you want to build and perform a systematic tree search.

```
Best_Solutions = []

Loop do:
   objective_function_value = Generate_Tree_State (Random_Search_Threashold)

   If objective_function_value >= Random_Search_Threshold:
       Add objective_function_value _State in Best_Solutions

For n_top_instances:
     local_solution  = Remaining_State_Random_Hill_Climbing(n_top_instance)
```

```
If local_search_solution > Best_Solutions[n_top_instance]:
   Return local_search_solution
Else:
   Return Best_Solutions[n_top_instance]
```

III. CONCLUSION

The third eye is a step toward a more accurate solution using lesser time as this algorithm does not entirely depend on randomization. This parameterized algorithm allows users to go for a percentage of randomization and the percentage of systematic search approach.

REFERENCES

[1]. Dumitrescu, Irina & Stützle, Thomas. (2003). Combinations of local search and exact algorithms. 211-223.
[2]. Artificial neural network - Wikipedia. https://en.wikipedia.org/wiki/Artificial_neural_network
[3]. Search Algorithms in AI - Javatpoint. https://www.javatpoint.com/search-algorithms-in-ai
[4]. Uninformed Search Algorithms - Javatpoint. https://www.javatpoint.com/ai-uninformed-search-algorithms