IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 4, Ver. I (July – Aug. 2015), PP 124-127
www.iosrjournals.org

Optimized AO* Algorithm for and-Or Graph Search

Aby Abahai T.[1]
[1](*Computer Science and Engineering, M. A. College of Engineering Kothamangalam,, India)*

Abstract : *The objective of this paper is to optimize AO* algorithm with a depth limited search. AO* is one of the major AND-OR graph search algorithms. According to AO* algorithm it is not exploring all the solution paths once it has got the solution. Here the modified method is providing better heuristic value for the solution if the search space is unstable. It is also considering interacting sub problems in the in the search space considering the loop structures.*
Keywords - *AND-OR graph, best first search, depth limited search, heuristic.*

I. Introduction

In an AND-OR graph AO* algorithm [1] is an efficient method to explore a solution path. AO* algorithm works mainly based on two phases. First phase will find a heuristic value for nodes and arcs in a particular level. The changes in the values of nodes will be propagated back in the next phase.

There are situations like a non-promising node eventually become a promising one. AO* algorithm [1] will not give the best solution in such cases. Here a new method is introduced to consider such nodes by performing a depth limited search before fixing the heuristic value. Also the proposed method is handling interacting sub problems which is not handled by AO* algorithm [1].

II. General AO* Alogorithm

In order to find solution in an AND-OR graph AO* algorithm [1] works well similar to best first search [2] with an ability to handle the AND arc appropriately. The algorithm finds an optimal path from initial node by propagating the results like solution and change in heuristic value [3] to the ancestors as in algorithm 1[1].

Algorithm 1: Basic AO*

1. Initialize the graph to the starting node.
2. Loop until the starting node is labeled SOLVED or until its cost goes above FUTILITY:
a) Traverse the graph, starting at the initial node and following the current best path and accumulate the set of nodes that are on that path and have not yet been expanded or labelled as solved.
b) Pick one of these unexpanded nodes and expand it. If there are no successors, assign FUTILITY as the value of this node. Otherwise, add its successors to the graph and for each of them compute f '. If f ' of any node is 0, mark that node as SOLVED.
c) Change the f ' estimate of the newly expanded node to reflect the new information provided by its successors. Propagate this change backward through the graph till the initial node. If any node contains a successor arc whose descendants are all solved, label the node itself as SOLVED.

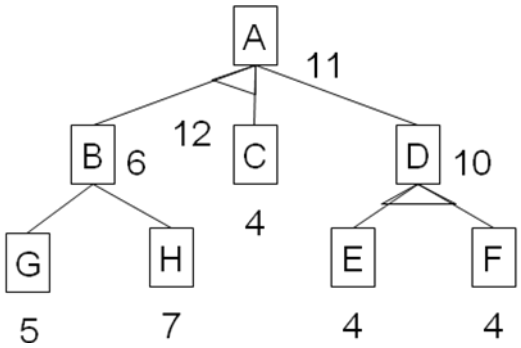

Fig. 1: Search According to AO*

Fig. 1 shows a solution according to AO* algorithm. Here path through node D is giving solution with a value 11 since arc containing B and C is not optimal.

III. Optimised Method

When performing AO* algorithm [1] some non promising nodes will be underestimated (Fig. 2). Once solution is found AO* is not looking into that. Here in Fig. 2 arc containing nodes B and C is becoming a promising on further exploration. So node A is getting a better score of 10.

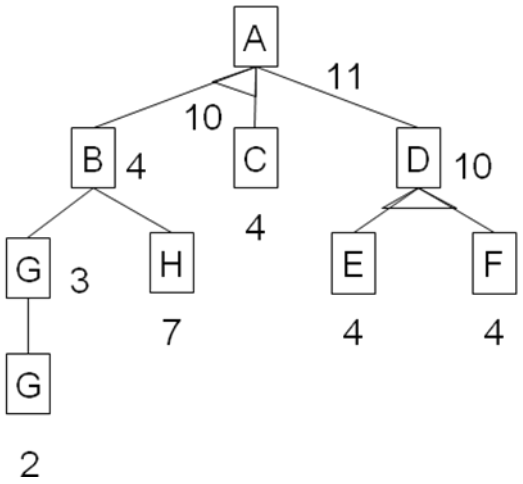

Fig. 2: Search Tree with Better Solution

The algorithm can be optimised by measuring the goodness of a node with a look ahead. In figure 2 node G is further estimated and found that I is giving a solution with better heuristic value. If there a look ahead on node B this would have been explored. Also node C may or may not give better value if explored further. This look ahead is implemented efficiently using a depth limited search on nodes examined by step 2(b) in the basic AO* algorithm.

3.1 Depth Limited Search

Basically Depth Limited Search(DLS) [4] is only a Depth First Search(DFS) with a depth limit. In a recursive method for DFS each recursive call is performed only if depth of the search tree is less than depth limit. But there is some change in the DLS algorithm for AND-OR graphs since arcs are there. The non promising nodes can be explored efficiently by the DLS method [4]. So this modification will propagate back a stable result to ancestors. In the figure 1 search with a depth limit 3 will give a solution with score 10.

3.2 Interacting Sub Problems

Interacting sub problem is one of the limitation to the algorithm. Here in Fig. 3 nodes B and C are interacting. But C finds that D optimal compared to B for solving it. This is an overhead for finding the solution since both B and D has to be solved separately.

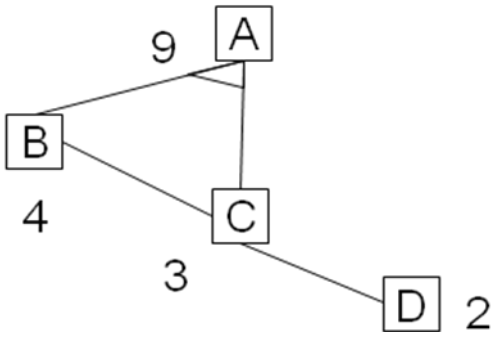

Fig. 3: Solution for Interacting Sub Problems in AO*

Considering the interaction the cost on A will get reduced to 7 (Fig. 4) instead of 9 without considering the node D. Here B has to be done only once under the arc. So B and C altogether taking a cost of 5 and the cost of the arc is 2 which will bring the total cost to 7.

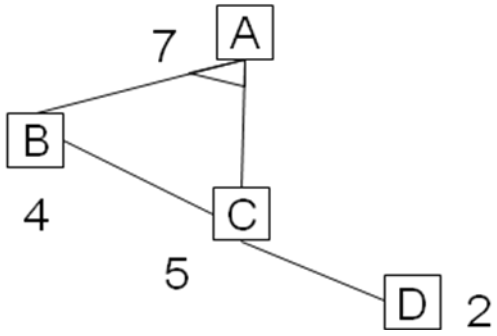

Fig. 4: Solution for Interacting Sub Problems in OAO*

3.3 Optimised AO* (OAO*) Algorithm

The optimization of AO* algorithm can done based the two aspects as discussed above. One is considering the non promising nodes and the other is considering interacting sub problems.

The algorithm 2 is explaining the depth limited search on nodes which will consider the non promising nodes also. An additional data structure in the form of Explored_List containing explored nodes with the information about the parent, list of unexpanded successors obtained through depth limited search, depth and evaluated score is required for the algorithm.

Algorithm 2: OAO* Algorithm
Data structure : Explored_List for explored nodes

1. Initialize the graph with the starting node as CURRENT node.
2. Loop with CURRENT until the starting node is labelled SOLVED or until its cost goes above FUTILITY:

a. If depth limit has reached return node's or arc's heuristic value.
b. Explore successor nodes and arcs of CURRENT. If there are no successors, assign FUTILITY as the value of this node and exit.
c. Otherwise loop for each SUCCESSOR node or arc of CURRENT.

i. find unexpanded successors of the SUCCESSOR of CURRENT by referring Explored_List if CURRENT is in Explored_List.
ii. make recursive calls for the algorithm with unexpanded successors SUCCESSOR as the starting node.
iii. if SUCCESSOR is an arc check for loops based on the parental information for finding interacting sub problems obtained from Explored_List. Evaluate the score for the loop and assign to SUCCESSOR if it is better than that of previous step.
iv. if SUCCESSOR is giving better value change the f estimate of the CURRENT node to reflect the new information provided by its successors.
v. if SUCCESSOR is giving better value call the it the BEST_SUCC.
vi. add or update CURRENT and the successor in Explored_List with relevant information.

a. Propagate this change backward through the graph till the initial node. If any node contains a successor arc whose descendants are all solved, label the node itself as SOLVED.
b. Make the BEST_SUCC as the CURRENT node.

The OAO* algorithm is finding a solution according to best first search where each node or arc performing a depth limited search. The Explored_List is keeping an updated list of explored nodes so that DLS overhead is reduced. Here the search is started only from the unexpanded nodes because the other explored nodes are already evaluated which reduces the search overhead. The interacting sub problems are identified by checking loops [5] making use of parental information in the Explored_List. If loops are there the evaluation is done as in Fig. 4. This is giving more realistic solution to problems.

IV. Analysis

The Optimised AO* algorithm is always giving a better result with new concepts like depth limited search and evaluation of loop for interacting sub problems. This is verified with evaluation of graphs shown in Fig. 2 and Fig. 4.The OAO* algorithm is evaluating lesser number of nodes compared to AO* with the a look up in Explored_List. So the explored nodes are not evaluated further which improves time complexity even though slight compromise with space complexity is required. Here the algorithm offers a better consistent trace for the solution by the look ahead based on a depth limited search

V. Conclusion

The OAO* algorithm has taken care of major drawbacks basic AO* algorithm. The proposed algorithm gives a stable heuristic search by avoiding underestimation of non promising nodes and overestimation of promising nodes. The algorithm has handled the problem of interacting sub problems to give better heuristic search. Also the algorithm is giving more consistent search path compared to AO* algorithm.

References

[1]. Elaine Rich, Kevin knight, Shivashankar b. Nair , Artificial Intelligence (New Delhi,The Tata McGraw-Hill Companies,2009).
[2]. Ethan Burns, Sofia Lemons, Wheeler Ruml, Rong Zhou, Best-First Heuristic Search for Multicore Machines, Journal of Artificial Intelligence Research, 39, 2010, 689–743.
[3]. Eric A. Hansen, Rong Zhou, Anytime Heuristic Search, Journal of Artificial Intelligence Research, 28,2007, 267-297.
[4]. Richard E. korf, Depth limited search for real time problem solving, The journal of real time systems,2,1990,7-24.
[5]. Eric A. Hansen, Shlomo Zilberstein, LAO*: A heuristic search algorithm that finds solutions with loops, Artificial Intelligence, 129, 2001, 35–62.