Anytime A* Algorithm – An Extension to A* Algorithm

Disha Sharma, Sanjay Kumar Dubey

Abstract— A* algorithm is a powerful tool that can be used to solve a large number of different problems. However, its computation cost may be unaffordable for some applications. To ease this problem and to adopt different application contexts, various extensions of A* have been anticipated. The main extension to the A* algorithm is Anytime Algorithm. This approach find improved solution as well as improve a bound on the sub optimality of the current solution.

Anytime A* Algorithm is algorithms those can optimize the memory and time resources and are considered best for A* algorithm. When the time available to solve a search problem is limited or uncertain, this creates an anytime heuristic search algorithm that allows a flexible substitution between search time and solution quality. They can generate a fast, non-optimal solution and then refine it to the optimal if more time is provided. Some applications like Real Time Strategy (RTS) games have applied these algorithms to find a fast solution that will help the algorithm to prune some paths during the subsequent computation that will find the optimal solution.

Index Terms— A*, AI Algorithms, Anytime Algorithms, Artificial Intelligence, Best First Search, Heuritics Search, Optimality

1 Introduction

A* is probably one of the most well known Artificial Intelligence Algorithms[1, 20]. Its objective is to find the shortest path in a graph from a node x-start to a node x-goal[13]. It combines the features of uniform-cost search and pure heuristic search to efficiently compute optimal solutions. As a best-first heuristic search, it employs a function f that guides the selection of the next node that will be expanded[10,18]. The order in which nodes are expanded is determined by the node evaluation function f(n) = g(n) + h(n), where g(n) is the cost of the best path currently known from the start node to node n, and h(n) is a heuristic estimated cost to reach from this current node to the goal node[15]. This estimation of the cost is determined using the problem specific information about the environment that the A* algorithm has. The behavior of A* depends in large part on the heuristic h(n) that guides the search. If h(n) is admissible, that is, if it never overestimates h*(n), and if nodes are expanded in order of f(n), then the first goal node selected for expansion is guaranteed to be optimal.

A heuristic is said to be consistent if $h(n) \le c(n,n') + h(n')$ for all n and n' where c(n,n') is the cost of an edge from node n to node n'[4]. If h(n) is consistent and nodes are expanded in order of f(n), the g-cost of a node is guaranteed to be optimal when the node is selected for expansion, and a node is never expanded more than once. If h(n) is not consistent, it is possible for A* to find a better path to a node after the node is expanded.

To prove that A* algorithm is optimal when using admissible heuristic, let us suppose sG as a suboptimal goal node and g(sG) as the exact cost of reaching from the root node to the current suboptimal goal node sG[14,19]. Also consider h(sG) as zero because for every goal node the estimated cost is always zero and let C* is the cost of the optimal path from the root node to the real goal node then

$$f(sG) = g(sG) + h(sG) = g(sG) > C^* \qquad (i)$$

The cost of the optimal path from root node to the suboptimal goal node sG is larger then the cost of the optimal path from root node to the real node[14,20]. Thus the suboptimal goal node sG will never be taken in as a goal.

Also suppose that node n as a node on the optimal path and if heuristic function h(n) is admissible, then we know

$$f(n) = g(n) + h(n) \le C^* \qquad (ii)$$

So, we can say that from (i) and (ii)

$$f(n) \le C^* < f(sG)$$

A* Algorithm will never selected sG as a goal node because its f-cost is greater than the cost of the real goal node rather A* algorithm will select node n.

2 Literature Review

Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute(now SRI International) first described the algorithm in 1968[19]. It is an extension of Edsger Dijkstra's 1959 algorithm. In 1968 Nils Nilsson suggested a heuristic approach for Shakey the Robot to navigate through a room containing obstacles. This path-finding algorithm, called A1, was a faster version of the then best known formal approach, Dijkstra's algorithm, for finding shortest paths in graphs. Bertram Raphael suggested some significant improvements upon this algorithm, calling the revised version A2. Then Peter E. Hart introduced an argument that established A2, with only minor changes, to be the best possible algorithm for finding shortest paths[19]. Hart, Nilsson and Raphael then jointly developed a proof that the revised A2 algorithm was *optimal* for finding shortest paths under certain well-defined conditions. They thus named the new algorithm in Kleene star syntax to be the algorithm that starts with A and includes all possible version numbers or A*.

The A* algorithm, stripped of all the code, is fairly simple. There are two sets, OPEN and CLOSED. The OPEN set contains those nodes that are candidates for examining. Initially, the OPEN set contains just one element: the starting position. The CLOSED set contains those nodes that have al-

ready been examined. Initially, the CLOSED set is empty. Graphically, the OPEN set is the "frontier" and the CLOSED set is the "interior" of the visited areas. Each node also keeps a pointer to its parent node so that we can determine how it was found.

There is a main loop that repeatedly pulls out the best node n in OPEN (the node with the lowest f value) and examines it. If n is the goal, then we're done. Otherwise, node n is removed from OPEN and added to CLOSED. Then, its neighbors n' are examined. A neighbor that is in CLOSED has already been seen, so we don't need to look at it (*). A neighbor that is in OPEN is scheduled to be looked at, so we don't need to look at it now (*). Otherwise, we add it to OPEN, with its parent set to n. The path cost to n', g(n'), will be set to g(n) + movement cost(n, n').

2.1 Existing Study

The A* algorithm is a flexible cost based algorithm, which can help satisfy all our requirements[5]. It is a best first search algorithm that finds the least costly path from an initial configuration to a final configuration. It uses an exact + estimate cost heuristic function. The cost function must be admissible i.e. the estimated cost must be less than the actual cost, this produce computationally optimal results.

Apart from using the evaluation function f(n), A* algorithm also uses two lists, Open list and Close list for the systematic search of the search space[8]. Close list stores all those nodes that are already been selected by the A* algorithm to be checked as the goal node and the Open list stores all the successors of those nodes in the Close list. These nodes in the Open list are sorted in an increasing order of their f-costs. The node with the least f-cost is selected next by the A* algorithm. So evaluation function f(n) is the main driving force that guides the A* search in the search space. Implementation of A* algorithm in the form of pseudo code is shown below:

```
public void A*(Node Root, Node Goal)
{
 if(Root==Goal)
 {
     goalSucc=true;
     return;
 }
 else
 {
     closeList.InsertNode(Root);
     Node[ ] successors = ExpandNode(Root);
     foreach( Node s in successor )
     {
         if(s==closeList[Item])
         {
         is_in_close = true;
         if(s.fcost <= closeList[Item].fcost)
             then update the item with the new
fcost
         }
         elseif(s == openList[Item])
         {
         is_in_open = true;
         if(s.fcost <= openList[Item].fcost)
             the update the item with the new fcost
         }
        elseif(is_in_close == false && is_in_open ==
false)
              openList.InsertNode(s);
       }
        openList.Sort();
        Node bestNode = open-
List.RemoveFirstNode();
      While( goalSucc == false )
      A*(bestNode, Goal);
      }
      }
```

The most essential part of the A* algorithm is a good heuristic estimate function[12]. This can improve the efficiency and performance of the algorithm and depends on the specific state space being explored. Inspite of being extensively used, A* may present problems in some situations.

i. Firstly, depending on the characteristics of the problem and heuristics used, its cost can be prohibitive.
ii. Secondly, there are some contexts such as dynamic environments or real-time searches may require adaptations in the original algorithm[9].

According to Pearl and Korf the main shortcoming of A* and any best-first search, is its memory requirement. Because the number of nodes expanded can be exponential in the length of the optimal path. A* is space limited[20]. For large search space, A* will run out of memory.

2.2 Proposed Study

There are several extensions to the A* algorithm in past few years out of which one of the extension is Anytime A* Algorithm[2].

The main structure of our implemented A* algorithm remains the same while converting it into anytime, because the basic working of the A* algorithm remains the same. The most important property of anytime algorithms is that, they can be stopped and then can be restarted at any time. So in order to make our implemented A* algorithm anytime we simply need to add a time constraint or a time limit to our algorithm. When the A* algorithm hits this limit it should be stopped and then restarted if desirable with new time limit. To avoid changes in our implemented A* algorithm, we implement a sub- module class called as control manager class (*ctr_manager class*), which takes care of the time limit and the stopping and restarting of the A* algorithm. Implementation of this class in the form of pseudo code is :

```
public class ctr_manager
{
     bool is_First = true;
     Node new_Solution = null;
```

```
public void run_A*( )
{
    thread.sleep( sleep_Time );
    goalSucc = true;
}
public void thread_AA*( int sleep_Time, Node rN, Node gN)
{
    thread_a = new thread( );
    if(is_First)
    {
        new_Solution = rN;
        thread.Start(run_A*);
        A*( rN, gN);
    }
    elseif( rN == new_Solution)
    {
        // keep all the states intact because algorithm needs more time
        // to find a better solution
        thread.Start(run_A*);
        A*( rN, gN);
    }
    elseif( rN != new_Solution)
    {
        // algorithm finds an improve solution, clears all the previous states
        // to allow algorithm a fresh start
        closeList.RemoveAll( );
        openList.RemoveAll( );
        thread.Start(run_A*);
        A*( new_Solution, gN);
    }
  }
}
```

3 Analysis

The complexity analysis of A* and related algorithms depends primarily on the quality of the heuristic function and has been the subject of a large body of research[3]. The time complexity of A* depends on the heuristic. In the worst case, the number of nodes expanded is exponential in the length of the solution (the shortest path), but it is polynomial when the search space is a tree, there is a single goal state, and the heuristic function *h* meets the following condition[16]:

$$| h(x) – h^*(x) | = O(\log h^*(x))$$

where h^* is the optimal heuristic, the exact cost to get from x to the goal. In other words, the error of *h* will not grow faster than the logarithm of the "perfect heuristic" h^* that returns the true distance from *x* to the goal[5,6].

A* algorithm can be easily declare as Anytime A* algorithm only when it must satisfy all properties of Anytime algorithms[11]. There are few such properties on the basis of which we can judge are as follows :

i. Measurable quality: This property means that the quality of the solution that an anytime algorithm returns, should be measureable and represent able in some way. In our implemented anytime A* algorithm we measure this quality of the solution in terms of its straight line distance from the goal node.
ii. Mono-tonicity: This property means that the quality of the solution that an anytime algorithm returns, should increases with increase in computational time and quality of the input.
iii. Consistency: According to this property the quality of the solution of an anytime algorithm is connected with computational time it have and the quality of the input.
iv. Interrupt-ability: This property means that for an algorithm to be declared as anytime we should be able to stop it at any time and it should provides us with some solution. Our implemented anytime A* algorithm is stoppable at any time and it will return a solution whenever it stops.
v. Preempt-ability: According to this property an anytime algorithm can be stopped and can also be restarted again with minimal overhead.

If the above defined all properties are satisfied then the anytime A* algorithm shows a high performance.

4 Conclusion

A* (a-star) is a well known best - first search algorithm that has been applied to the solution of different problems. We have presented a simple approach for converting a heuristic search algorithm such as A* into an anytime algorithm that offers a tradeoff between search time and solution quality. The approach uses a control manager class which takes care of the time limit and the stopping and restarting of the A* algorithm to find an initial, possibly suboptimal solution, and then continues to search for improved solutions until meeting to a provably optimal solution. It also bounds the sub-optimality of the currently available solution.

The simplicity of the approach makes it very easy to use. It is also widely applicable. Not only can it be used with other search algorithms that explore nodes in best-first order. Thus, we conclude that anytime heuristic search provides an attractive approach to challenging search problems, especially when the time available to find a solution is limited or uncertain.

References

[1] Koenig, S.:Dynamic fringe-saving A* (june 2010) retrieved 7, 2010 from http://idm-lab.org/bib/abstracts,koen09e.html
[2] S.Zilberstein, (1996), "Using Anytime Algorithms in Intelligent Systems", AI Magazine, Volume 17, No. 3, page 73-83

[3] Korf, R. E., Reid, M.: Complexity analysis of admissible heuristic search. In: Proceedings of the National Conference on Artificial Intellligence- AAAI (1998)

[4] Koenig, S., Likhachev, M, Lice, Y., Furcy, D.: Incremental heuristic search in AI, AI Magazine, Volume 25, No. 2, 2004, page 99-112

[5] Pearl, Judea (1984), Heuristics: Intelligent Search Strategies for computer Problem solving, Addison-Wesley, ISBN 0-201-5564-5

[6] Russell, S.J: Norvig, (2003), Artificial Intelligence: A Modern Approach, N,J: Prentice Hall, page 97-104, ISBN 0-13-790395-2

[7] A. Gyorgy and L. Kocsis(2011) "Efficient Multi-Start Strategies for Local Search Algorithms", Journal of Artificial Intelligence Research, Voulme 41, page 407-444

[8] N. Fu, H.c. Lau, P. Varakantham and F. Xiao (2012) "Robust Local Search for Solving RCPSP/max with Durational Uncertainity", Journal of Artificial Intelligence Research, Vol;ume 43, page 43-86

[9] V. Bulitko, Y. Bjornsson and R. Lawrence (2010) " Case-Based Subgoaling in Real Time Heuristic Search for Video Game Pathfinding", Journal of Artificial Intelligence Research, Volume 39, pages 269-300

[10] T.Nurkkala, "Informed Search- Lecture 4", Department of Computer Science, university of Minnesota

[11] W. Kywe, D. Fujiwara and K. Murakarni, " Scheduling of image Orocessing Using Anytime Algorithm for Real Time System", IEEE – International Conference on Pattern Recognition, 2006

[12] G. Pesant, C. Quimper and A. Zanarini (2012), " Counting Based Search: Branching Heuristics for Constraint Satisfaction Problem", Journal of Artificial Intelligence Research, Volume 43, page 173-210)

[13] Bolc, L., and J. Cytowski, Search Methods for Artificial Intelligence, Academic Press, London, 1992

[14] Dechter, R., and J. Pearl, Generalized best-first search strategies and the optimality of A*, Journal of the Association for Computing Machinery, Volume 32, No. 3, July 1985, page 5050-536

[15] P. F. Felzenszwalb and D. McAllester (2007)," The Generalized A* Architecture", Voulme 29, page 153-190

[16] Korf, R. E., Space-Efficient Search Algorithms, Computing Surveys, Volume 27, No. 3, Sept., 1995, page 337-339

[17] Bonent, B., & Geffner, H.(2001), Planning as heuristic search, Journal of Artifical Intelligence, Volume 129, No. 1, page 5-33

[18] Hansen, E., and Zilberstaein, S. (2001), LAO*: A heuristic search algorithm that finds solution with loops, Journal of Artificial Intelligence, Volume 129, No. 2, page 35-62

[19] Hart, P.E.: Nilsson, N. J.; Raphad, B.(1968). "A Formal Basis for the Heuristic determination of Minimum Cost Paths", IEEE Transactions on Systems Science and Cybernatics SSC4 4, No. 2: page 100-107, doi: 10.1109/TSSC.1968.30013C

[20] SreeKanth Reddy Kallem, " Artificial Intelligence Algorithms", IOSR Journal of Computer Engineering(IOSRJCE), Voulme 6, Issue 3, Sep-Oct. 2012, page 1-8

- *Disha Sharma is currently pursuing masters degree program in Computer Science and Engineering in Amity School of Engineering & Technology, Amity University, Noida, India. E-mail: disha.24sharma@gmail.com*
- *Sanjay Kumar Dubey, Assistant Professor, Amity School of Engineering & Technology, Amity University, Noida, India.. E-mail: skdubey1@amity.edu*