A Star Algorithm

Sai Varsha Konakalla

Indiana State University

skonakalla@sycamores.indstate.edu

December 12, 2014

Abstract

The A* algorithm combines features of uniform-cost search and pure heuristic search to effectively compute optimal solutions. Noted for its performance and accurancy, it enjoys widespread use.

1 Introduction

What is A*? A* is one of the many search algorithms that takes an input, evaluates a number of possible paths and returns a solution.

In computer science, A* (pronunced as A Star) is a computer algorithm that is widely used in pathfinding and graph traversal, the process of plotting an effi- ciently traversable path between points, called nodes.

The A* algorithm combines features of uniform-cost search and pure heuristic search to effectively compute optimal solutions. Noted for its performance and accurancy, it enjoys widespread use. Peter Hart, Nils Nilsson and Bertram Raphael first described the algorithm in 1968. It is an extension of Edsger Dijkstr as's 1959 algorithm. A* achieves better performance (with respect to time) by using heuristics.

A Heuristic Approach The defining characteristics of the A* algorithm are the building of a "closed list" to record areas evaluated, a "fringe list" to record areas adjacent to those already evaluated, and the calculation of distances travelled from the "start point" with estimated distances to the "goal point".

The fringe list, often called the "open list" , is a list of all locations immediately adjacent to areas that have already been explored and evalua ted (the closed list).

The closed list is a record of all locations which have been explored and evaluated by the algorithm.

2 Little History on A*

In 1964 Nils Nilsson invented a heuristic based approach to increase the speed of Dijkstra's algorithm. This algorithm was called A1.

In 1967 Bertram Raphael made dramatic improvements upon this algorithm, but failed to show optimality. He called this algorithm A2.

Then in 1968 Peter E.Hart introduced an argument that proved A2 was optimal when using a consistent heuristic with only minor changes. His proof of the algorithm also included a section that showed that the new A2 algorithm was the best algorithm possible given the conditions.

He thus named the algorithm in kleene star syntax to be the algorithm that starts with A and includes all possible version number or A* .

3 Concept

As A* traverses the graph, it follows a path of the lowest known cost, Keeping a sorted priority queue of alternate path segments along the way. If, at any p oint, a segment of the path being traversed has a higher cost than another encountered path segment, it abandons the higher-cost path segment and traverses the lower-cost path segment instead. This process continues until the goal is reached.

4 Working of A*

A* uses a best-first search and finds a least-cost path from a given initial node to one goal node (out of one or more possible goals). As A* traverses the graph, it follows a path of the lowest known heuristic cost, keeping a sorted priority queue of alternate path segments along the way. Similar to greedy best-first search but is more accurate because A* takes into account the nodes that have already been traversed.

A* figures the least-cost path to the node which makes it a best first search algorithm. Uses the formula f(x) = g(x) + h(x) where

g(x) is the total distance from the initial position to the current position.

h(x) is the heuristic function that is used to approximate distance from the current location to the goal state. This function is distinct because it is a mere estimation rather than an exact value. The more accurate the heuristic the better the faster the goal state is reach and with much more accuracy.

f(x) = g(x) + h(x) this is the current approximation of the shortest path to the goal.

5 Example Using A* Search

Here we are using Disneyland Paris to provide an example to traverse nodes from an initial state to a goal state.

- The main entrance is the initial node.
- The Magic Kingdom is the goal state.
- There are two paths that can be taken and are marked by nodes.
- Each node will have the f(x), g(x), and h(x).
- Then it will show at each node and indicate which is the next node that it will traverse based on least path cost.

varsha.png

Say you are at the entrance of Disneyland Paris and you are trying to get to the Magic Kingdom. There are two distinct paths that overlap in the center. There are two options in this case:

- The purple one with the higher f(x).
- The green one with the lower f(x).

As mentioned before A* will choose the one with the lowest f(x). A* found the node with the smallest f(x) value. When the goal node is popped off the pr iority queue then the search stops.

6 Pseudocode for A* Search

```
function A*(start,goal)

     closed = empty set

   q = makequeue(path(start))

  while q is not empty do

    p = remove first(q)

    x = lastnode(p)

  if x in closed then

  end if
  if x = goal then

    return p

  end if
  add x to closed

  for y \leftarrow  successor(x) do

    enqueue(q, p, y)

   end for
   return F ailure


  end while
end function
```

7 Uses for A*

- Shortest path - Usually we are interested in finding the shortest or most efficient path between two nodes, such as the shortest path between two tiles on a map. A boar d game

may need to know if a piece can reach some tile and how many moves it would require to get there.

- Flood fill - If we ask a path finding algorithm to search for a path to an unreachable destination we won't get a result but we still get useful information. The set of nodes the algorithm explored trying to find a path gives us all the nodes that are reachable from our starting location. If our graph represents a map we can use this to identify if two land masses are connected or find all the locations which are part of a lake.
- Decision making - Our graph does not need to represent a set of physical locations. Instead suppose each node represents some form of technology in our game's tech tree. We can use a path finding algorithm as part of our AI to determine the cheapest series of upgrades requires to reach a specific technology level.

8 Drawbacks

The main drawback of A* algorithm and indeed of any best-first search is its memory requirement. Since at least the entire open list must be sa ved, A* algorithm is severely space-limited in practice, and is no more practical than best-first search algorithm on current machines. For example, while it can be run successfully on the eight puzzle, it exhausts available memory in a matter of minutes on the fifteen puzzle.

9 Improvements

A* is a breadth first algorithm and as such consumes huge memory to keep the data of current proceeding nodes. The search can be more efficient if the machine searches not just for the path from the source to the target, but also in parallel for the path from the target to the source (the answer is found when these two searches meet at some point).

10 Practical Applications of A*

A* is the most popular choice for path finding, because it's fairly flexible and can be used in a wide range of contexts such as games (8-puzzle and a path finder).

- Variations of A*
- Bidirectional search
- Iterative deepening
- Beam search
- Dynamic weighting
- Bandwidth search
- Dynamic A* and Lifelong Planning A*

References

[1] http://www.policyalmanac.org/games/aStarTutorial.htm

[2] http://www.codeproject.com/Articles/9880/Very-simple-A-algorithm-implementation

[3] http://en.wikipedia.org/wiki/A*_search_algorithm

[4] http://cssubjects.skoze.com/2013/02/24/shortest-path-finding-using-a-star-search-algorithm/