PATH FINDING - Dijkstra's and A* Algorithm's

Harika Reddy

December 13, 2013

1 Dijkstra's - Abstract

Dijkstra's Algorithm is one of the most famous algorithms in computer science. Back before computers were a thing, around 1956, Edsger Dijkstra came up with a way to find the shortest path within a graph whose edges were all non-negetive values. To this day, almost 50 years later, his algorithm is still being use d in things such as "link-state routing". It has been extended by others to create more advanced path finding algorithms such as A*.

As far as optimal solutions (paths) are concerned, Algorithms designed for the shortest path problems should be capable of handling three cases.

- An optimal solution exists.
- No optimal solution exists because there are no feasible solutions.
- No optimal solution exists because the length of feasible paths from city 1 to city n is unbounded from below.

keywords : Dijkstra's Algorithm, Shortest Path, Link-State Routing, Path Finding Algorithms.

2 Dijkstra's - A Greedy Approach

Approach of the algorithm is iterative and also maintains shortest path with each intermediate nodes.

Greedy algorithms use problem solving methods based on actions to see if there's a better long term strategy. Dijktra's algorithm uses the greed y approach to solve the single source shortest problem. It repeatedly selects from the unselected vertices, vertex v nearest to source s and declares the dis tance to be the actual shortest distance from s to v. The edges of v are then checked to see if their destination can be reached by v followed by the relevan t outgoing edges.

Here is the greedy idea of Dijkstra's algorithm:

- Maintain a set S of vertices whose shortest-path from s are known ($s \in S$ initially).
- At each step add vertex v from the set V-S to the set S. Choose v that has minimal distance from s (be greedy).
- Update the distance estimates of vertices adjacent to v.

IRONY: Ants dont need dijkstra's algorithm to find the shortest path to the picnic. They always find their way, selecting the fastest route, overco ming obstacles, maximizing return effort. near perfect utilization of resources with maximum benefit; completing herculean tasks with unparalleled cooperatio n and teamwork.

3 A* - Abstract

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* (as "A star") is a computer algorithm that is widely used in pathfinding and graph traversal, the process of plotting an efficiently traversable path between points, calle d 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 a's 1959 algorithm. A* achieves better performance (with respect to time) by using heuristics.

Keywords : Search Algorithms, Path Finding, Graph Traversal, Heuristics, Optimal Solutions.

4 A* - 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.

5 History on Dijkstra’s

Edsger Wybe Dijkstra
1930 –2002

"Computer Science is no more about computers than astronomy is about telescopes."

Edsger Dijkstra is Dutch.

He is one of the big names in computer science. He is known for his handwriting and quotes such as:

- Simplicity is prerequisite for reliability.
- The question of whether machines can think is about as relevant as the question of whether submarines can swim.

Dijkstra’s algorithm was created in 1959 by Dutch Computer Scientist Edsger Dijkstra. While employed at the Mathematical Center in Amsterdam, Dijkstr a was asked to demonstrate the powers of ARMAC, a sophisticated computer system developed by the mathematical Center. Part of his presentation involved illus trating the best way to travel between two points and in doing so, the shortest path algorithm was created. It was later on renamed as Dijkstra’s Algorithm in recognition of its creator.

In particular, we are reminded that this famous algorithm was strongly inspired by Bellman's principle of optimality and that both conceptually and tec hnically it constitutes a dynamic programming successive approximation procedure par excellence.

click here to watch Dijkstra's Algorithm:
`http://www.youtube.com/watch?v=8Ls1RqHCOPw`

6 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* .

7 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 t he lower-cost path segment instead. This process continues until the goal is reached.

8 Implemenatation with example

Consider the following example:

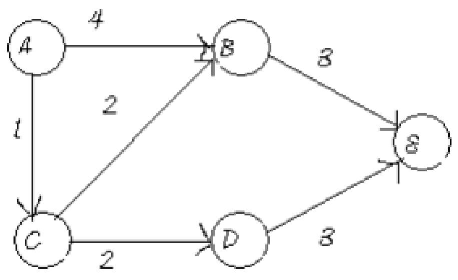

Figure: Weighted-directed graph

The above weighted graph has 5 vertices from A-E. The value between the two vertices is known as the edge cost between two vertices. For example the edge cost between A and C is 1. Using the above graph the Dijkstra's algorithm is used to determine the shortest path from the source A to the remaning vertices in the graph.

The example is solved as follows:

Initial Step

sDist[A] = 0; *The value to the source itself*

sDist[B] , sDist[c] , sDist[D] , sDist[E] equals Infinity ; *The nodes not processed yet*

STEP 1 :

sDist[B] = 4;

sDist[C] = 2;

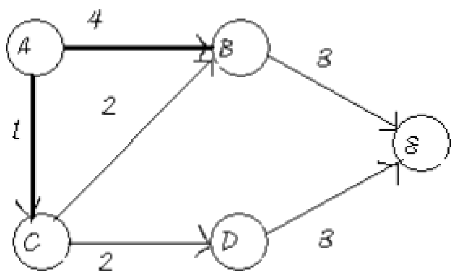

Figure: shortest path to vertices B, C from A

STEP 2 :

sDist[B] = 3;

sDist[D] = 2;

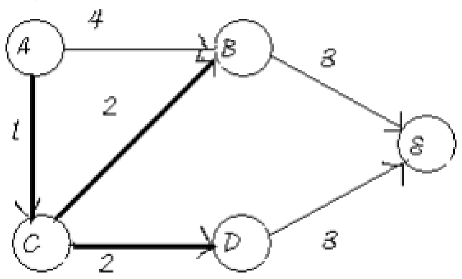

Figure: Shortest path from B, D using C as intermediate vertex

STEP 3 :

sDist[E] = 6;

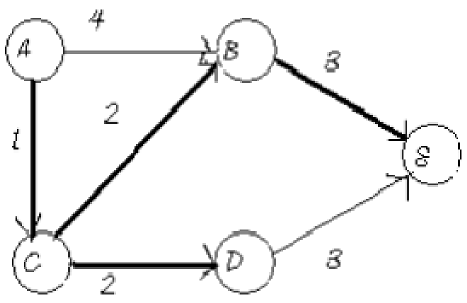

Figure: Shortest path to E using B as intermediate vertex

STEP 4 :

Adj[E] = 0; means there is no outgoing edges from E, And no more vertices, algorithm terminated. Hence the path which follows the algorithm is :

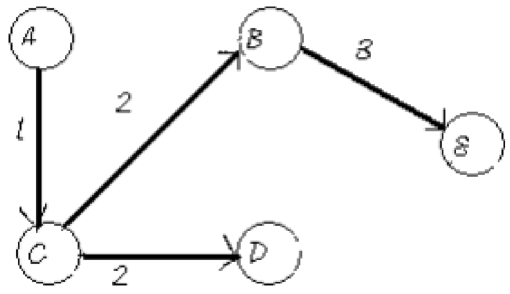

Figure: the path obtained using Dijkstra's Algorithm

9 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.

10 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.

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.

11 Pseudocode for Dijkstra's

```
procedure DIJKSTRA'S(G)                    ▷ Weighted graph,with all weights positive
    $dist[s] \leftarrow 0$                           ▷ distance to source vertex is Zero
    for $all\ v \in V - \{s\}$ do
        $dist[v] \leftarrow \infty$                  ▷ set all other distances to infinity
    end for
    $S \leftarrow \emptyset$                         ▷ S,the set of visited vertices is intially empty
    $Q \leftarrow V$                                 ▷ Q,the queue intially contains all vertices
    while $Q \neq \emptyset$ do                      ▷ while the queue is not empty
        $u \leftarrow mindistance(Q, dist)$          ▷ select ele of Q with the min. distance
        $S \leftarrow S \cup \{u\}$                  ▷ add u to list of visited vertices
        for $all\ v \in neighbors[u]$ do
            if $dist[v] > dist[u] + w(u, v)$ then    ▷ if new shortest path found
                $d(v) \leftarrow d(u) + w(u, v)$     ▷ set new value of shortest path
            end if
        end for                                      ▷ if desired,add traceback code
    end while
return $dist$
end procedure
```

12 Pseudocode for A* Search

```
function A*(start,goal)
    $closed =$ empty set
    $q = makequeue(path(start))$
    while q is not empty do
```

```
        $p = removefirst(q)$

        $x = lastnode(p)$

        **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** $Failure$

    **end while**
**end function**
```

13 Efficiency of Dijkstra

The complexity/effciency can be expressed in terms of Big-O notation. Big-O gives another way of talking about the way inputs affects the algorithm's ru nning time. It gives an upper bound of the running time.

In Dijkstra's algorithm, the effciency varies depending on $|V| = n$ DeleteMins and $|E|$ updates for priority queues that were used.

If a **Fibonacci heap** was used then the complexity is $\bigcirc(|E| + |V| \log |V|)$, which is the best bound. The DeleteMins operation takes $\bigcirc(\log |V|)$.

If a **Standard binary heap** is used then the complexity is $\bigcirc(|E| \log |E|)$, $|E| \log |E|$ term comes from $|E|$ updates for the stand ard heap.

If the set used is a prriority queue then the complexity is $\bigcirc(|E| + |V|^2)$. $\bigcirc(|V|^2)$ term comes from $|V|$ scans of the unordered set New Frontier to find the vertex with least sDist value.

14 Pitfalls

There's a problem with this algorithm - it can only see the neighbors of the immediate node. The issue that can arise is if you choose a short node that is forked. Since the algorithm is not backtracking, it can potentially degrade into a infinite loop, especially since it will eventually run out of suitable
neighbors to inspect all while knowing that not all nodes have been visited.

The major problem of the algorithm is the fact that it does a blind search there by consuming a lot of time waste of necessary resources.

Another problem is that it cannot handle negetive edges. This leads to acyclic graphs and most often cannot obtain the right shortest path.

15 Related Algorithms - Dijkstra

A* algorithm is a graph/tree search algorithm that finds a path from a given initial node to a given goal node it employs a "heuristic estimate " h(x) that gives an estimate of the best route that goes through that node. It visits the nodes in order of this heuristic estimate. It follows the approach of best first search.

The **Bellman-Ford algorithm** computes single-source shortest paths in a weighted digraph. It uses the same concept as that of Dijkstra's algorith m but can handle negetive edges as well. It has a better running time than that of Dijkstra's algorithm.

Prims's algorithm finds a minimum spanning tree for a connected weighted graph. It implies that it finds a subset of edges that form a tree whe re the total weight of all the edges in the tree is minimized. it is sometimes called the DJP algorithm or jarnik algorithm.

16 Improvements

Use a d-way heap(Johnson, 1970s)

- easy to implement.
- reduces cost to $Ed\log_d(V)$.
- Indistinguishable from linear for huge sparse grpahs found in practice.

Use a Fibonacci heap(Sleator-Tarjan, 1980s)

- very dfficult to implement.
- reduces worst-case costs(in theory) to $E + V \log V$
- not quite linear
- practical utility questionable.

17 Applications

Traffic information systems use Dijkstra's algorithm in order to track the source and destinations from a given particular dource and destination.

OSPF-Open Shortest Path First, used in internet routing. It uses a link-state in the individual areas that make up the hierarchy. The computation is bas ed on Dijkstra's algorithm which is used to calculate the shortest path tree inside each area of the network.

18 Limitations

Once thing we haven't looked at is the problem of finding shortest paths that must go through certain points. This is a hard problem and is reducible to the Travelling Salesman problem–what this means in practice is that it can take a very long time to solve the problem even for very small inputs.

19 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.

20 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.

21 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).

21.1 Improved reversely a star path search algorithm based on the comparison in valuation of shared neighbor nodes

A new A star path finding algorithm(named SNA star) was proposed to solve the problem of path finding and optimization by comparing the valuati on of shared neighbor nodes, which is based on the reverse improved A star path search algorithm. The characteristics of this algorithm in the paper take the use of the valuation of shared neighbor nodes to verify the search direction, and reversely search from the target to starting, which used the way of local evaluation instead of the way of global evaluation. Taking two simulation experiments as examples to show that the algorithm is effective both on speed and a pplicability in complex environment. The validity of the algorithm was demonstrated from the comparison and results of the simulation experiment. The evaluat ion function of this algorithm in the multi-obstacles environment was optimized and evaluation speed was increased at the same time.

22 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*

23 For Dijkstra's

References

[1] `http://en.wikipedia.org/wiki/Dijkstra's_algorithm`

[2] `http://courses.csail.mit.edu/6.006/spring11/lectures/lec16.pdf`

[3] `http://www.ifp.illinois.edu/~angelia/ge330fall09_dijkstra_l18.pdf`

[4] `http://www.youtube.com/watch?v=8Ls1RqHCOPw`

24 For A*

References

[1] `http://www.policyalmanac.org/games/aStarTutorial.htm`(Description for A* algorithm)

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

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

[4] `http://www.princeton.edu/~achaney/tmve/wiki100k/docs/A*_search_algorithm.html`

[5] `https://decibel.ni.com/content/docs/DOC-8983`

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

[7] http://gamedev.tutsplus.com/tutorials/implementation/speed-up-a-star-pathfinding-with-the-jump-point-search-algorithm

[8] https://code.google.com/p/jianwikis/wiki/AStarAlgorithmForPathPlanning

[9] https://blog.itu.dk/bads-f2010/files/2010/03/41-a-star.pdf

25 for Documentation

References

[1] http://cs.indstate.edu/CS695/latexprimer.pdf

[2] http://omega.albany.edu:8008/Symbols.html