Position Paper: Dijkstra's Algorithm versus Uniform Cost Search or a Case Against Dijkstra's Algorithm

Ariel Felner
Information Systems Engineering
Ben-Gurion University
Be'er-Sheva, Israel 85104
felner@bgu.ac.il

Abstract

Dijkstra's single-source shortest-path algorithm (DA) is one of the well-known, fundamental algorithms in computer science and related fields. DA is commonly taught in undergraduate courses. *Uniform-cost search* (UCS) is a simple version of the *best-first search* scheme which is logically equivalent to DA. In this paper I compare the two algorithms and show their similarities and differences. I claim that UCS is superior to DA in almost all aspects. It is easier to understand and implement. Its time and memory needs are also smaller. The reason that DA is taught in universities and classes around the world is probably only historical. I encourage people to stop using and teaching DA, and focus on UCS only.

Introduction

Edsger Wybe Dijkstra (1930-2002), a winner of the 1972 Turing award is perhaps best remembered by his *single-source shortest path* algorithm. This algorithm is known to anyone who studied computer science or relevant fields as "Dijkstra's algorithm", or even as "Dijkstra". We denote it hereafter by DA. The main idea is to place all vertices of a graph in a priority queue which is keyed by the distance from the source vertex s (labeled $dist[s]$). Then, vertices that are closest to s are removed and the $dist$ of their neighbors is updated. This process is repeated until the shortest path from s is found. The original paper was published in 1959 (Dijkstra 1959) and it is still cited 50 years later in modern papers. According to *Google scholar* it has been cited 7817 times (June 23, 2011). The *Uniform cost search* algorithm (denoted hereafter as UCS) is a special case of the general class of *best-first search* algorithms. An *open-list* (denoted OPEN) of nodes is initiated. Then, at each cycle a node with lowest cost in OPEN, according to some cost function, is selected for expansion until the goal node is chosen for expansion. In UCS the cost of a node is the shortest distance found so far from the source node ($g(n)$).

The two algorithms have many similarities and are logically equivalent. The most important similarity is that they expand exactly the same nodes and in exactly the same order. However, there are many differences between these algorithms as described in text books and taught in classes. The main difference is the identity of the nodes in the priority queue. In DA, *all* nodes are initially inserted into the queue. In UCS, nodes are inserted to the queue lazily during the search. In this paper I would like to shed light on this and on other differences between the algorithms with an important message - *In almost all aspects UCS is superior*. In my opinion, UCS has significant advantages over DA in its pedagogical structure, in its pseudo code, in its time and memory needs and in its behavior in practice. Based on Dijkstra's own words (Dijkstra 1959) I also conjecture that he formulated his algorithm as UCS but at some point of time people reverted to the way DA is commonly presented. It is therefore my opinion that scientists, university instructors, teachers and practitioners should cease to use DA and teach, implement and experiment with UCS. The name *Dijkstra's algorithm* can/should still be used as he was perhaps the first to write about this logical behavior. However, in practice, we must only use UCS.

Dijkstra's algorithm (DA)

In this section I give the simplest possible description of Dijkstra's algorithm (DA) as provided in the common textbook *"Introduction to algorithms"* (Cormen et al. 2001) and other classical books (Sedgewick and Wayne 2011; Aho, Hopcroft, and Ullman 1987) and as is usually taught around the world. Similarly, many scientific papers about DA, use this framework (Sniedovich 2006; Cherkassky, Goldberg, and Radzik 1996). Given a source vertex s in a weighted directed graph $G = (V, E)$ where all edges are nonnegative, DA finds the path with lowest cost (shortest path) between s and every other vertex in G.

DA, presented in Algorithm 1, maintains the vertices of the graph V in two disjoint and complementary sets of vertices $S \subseteq V$ and $Q \subseteq V$. S, initialized by $\emptyset$ (line 2), includes all vertices whose shortest path from s has been determined. Q is a priority queue initialized by *all* $x \in V$ (Line 4). Q is keyed by $dist[x]$, which is the length of the currently shortest known path from s to $x \in Q$. $dist[]$ is initialized (Lines 1-2) such that for $x \in V \setminus s$ $dist[x] = \infty$ while $dist[s] = 0$. S is initialized by $\emptyset$.

At every cycle, DA extracts the vertex $u \in Q$ with the minimal $dist[]$ in Q. Then, for each neighbor v of u it sets $dist[v] = min(dist[v], dist[u] + w(u, v))$ (this is called the *relax* operation). u is then added to S. DA maintains an

Algorithm 1: Dijkstra's algorithm

```
   Input: Graph $G = (V, E)$
1  $(\forall x \neq s) dist[x] = +\infty$   //Initialize dist[]
2  $dist[s] = 0$
3  $S = \emptyset$
4  $Q = V$   // Keyed by $dist[]$.
5  while $Q \neq \emptyset$ do
6      $u = extract\_min(Q)$
7      $S = S \cup \{u\}$
8      foreach vertex $v \in Adj(u)$ do
9          $dist[v] = min(dist[v], dist[u] + w(u, v))$
10             //"Relax" operation.
```

invariant that whenever u is chosen from Q then it must be that $dist[u] = \delta(u)$, where $\delta(x)$ is the shortest path from s to x. The proof is by induction. Initially this is true for s. Then, every time u is chosen, since it was the node with the minimal $dist[]$ in Q it follows that $dist[u]$ must be equal to $\delta(u)$, otherwise, we reach a contradiction. We note here that the exact formal proof as presented by (Cormen et al. 2001) is very long and rather complicated.

It is important to note that DA in its basic form as usually appears in text books is designed to find the shortest paths to *all* vertices of the graph (known as the *single-source shortest path* problem). However, with a small modification, it can be used to only find a shortest path to any given goal (a *source-target shortest-path* problem) or to any set of goals. Once S includes the goal(s), the algorithm halts.

Uniform Cost Search (UCS)

Best-first search, shown in Algorithm 2 is a class of algorithms which includes *uniform cost search* (UCS) as a special case. The main data structure is the *open-list* (OPEN). OPEN is a priority queue initialized with the source vertex s. Then, at each cycle, a node u with the lowest cost is extracted from OPEN. This is referred to the *expansion* of u. Each of its neighbors v is now *generated* and its cost is determined. A duplicate check (see below) is performed on v and if v passed the duplicate check it is inserted to OPEN. u itself is inserted into the closed-list (CLOSED). Nodes in OPEN are usually referred to as the *frontier* of the search.

Versions of best-first search differ by the cost function used to rank nodes in OPEN. *Breadth-first search* is a special case of best-first search where the cost of a node n is $d(n)$ - the depth of the node in the search tree. UCS is a straightforward generalization of breadth-first search where the cost function is $g(n)$, the sum of the weights of the edges from the source node to node n along the shortest currently known path. In UCS, when a node u is *expanded* and its neighbor v is *generated*, its cost is set to $g(v) = g(u) + w(u, v)$.

Algorithm 2: Best-first search algorithm

```
   Input: Source vertex $s$
1  $OPEN.insert(s)$
2  while $OPEN \neq \emptyset$ do
3      $u = OPEN.extract\_min()$
4      foreach vertex $v \in Adj(u)$ do
5          $g(v) = g(u) + w(u, v)$
6          $v' = check\_for\_duplicates(v)$
7          $OPEN.insert(v')$
8          $CLOSED.insert(u)$
```

It is important to note that in DA, *all* nodes were initially inserted to Q, while in UCS, as a best-first search, nodes are inserted to the queue only when they are generated by the search. This is a great advantage of UCS as detailed below.

Duplicate check for UCS

Duplicate checks might be slightly different for different versions of best-first search. For UCS it is done as follows. When a new node v is generated we check whether it is already in OPEN. If it is, we keep a single copy of v with the smallest g-value among the two copies (labeled v' in Algorithm 2). If v is in CLOSED, its new copy is discarded.[1]

Correctness of UCS

It is easy to see that when all edges are nonnegative, then when a node n is chosen for expansion, its g-value equals the shortest path to n. This is due to a list of easy to verify *UCS invariants* which can be briefly sketched as follows:
Invariant 1: OPEN (the frontier nodes) is a perimeter of nodes around s.
Invariant 2: Since all edges are non-negative, $g(n)$ is monotonically non decreasing along any path.
Invariant 3: When a node n is chosen for expansion, $g(n)$ is the length of a path which is *smaller than or equal to* the g-value of all other nodes in OPEN.
Consequence: If there is a shorter path to n, there must be an ancestor a of n in OPEN (Invariant 1) with $g(a)$ smaller than $g(n)$ (Invariant 2). This is impossible (Invariant 3). Note that Invariant 2 is not true if edges might be negative.

In general a node n in UCS (and in best-first search in general) is going through the following three stages:

Unknown: n was not yet generated.

Opened: n was generated and it is in OPEN.

Closed: n was expanded and added to CLOSED.

As in DA, UCS can be tuned to halt either when a goal node is chosen for expansion (or when the set of goals were expanded), or when *all* nodes were closed. That is, both the *single-source shortest path* problem and the *source-target shortest-path* problem can be solved by UCS.

Similarities of DA and UCS

In this section, we show that the two algorithms are similar. In particular, we will show that DA is a simulation of UCS. In a deeper inspection of DA, one can recognize both OPEN and CLOSED. It is easy to see that the set of nodes

[1]In some versions of best-first search, if the new copy of v has a better cost than the copy in CLOSED, v is removed from CLOSED and is reinserted to OPEN with the new cost. This process is called *node reopening* and only occurs if the cost function is non-monotonic (Felner et al. 2010).

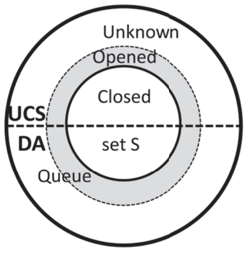

Figure 1: Stages of DA Vs. UCS

S in DA is equivalent to CLOSED in UCS. Once a node is added to S or inserted into CLOSED, the shortest path to it has been found. Recognizing OPEN in DA is more tricky. In UCS, the set of OPEN nodes is exactly the set of nodes in the priority-queue. In DA, however, *all* nodes were initially entered to the priority-queue. A simple well-known observation about best-first search is that OPEN is a *frontier* or a *perimeter* of nodes around CLOSED. Thus, OPEN can be recognized in DA as all the nodes in Q which are neighbors of S. When a node in a best-first search is expanded and moved to CLOSED, its neighbors are generated and added and to OPEN. Their g-value is that of their parent plus the edge connecting them. In DA, the OPEN nodes are nodes in Q whose $dist[]$ is no longer ∞, i.e., some node which was removed from Q relaxed/changed their $dist[]$ value.

Based on this, the three stages of UCS can be recognized in DA. *Unknown* nodes are *all* the nodes in Q with $dist[] = \infty$. *Opened* nodes are all the nodes in Q with $dist[] \neq \infty$. *Closed* nodes are those in S. The different sets of nodes of the two algorithms are shown in figure 1.

Duplicate pruning

The *relax* operation in DA (lines 9-10) is identical to the duplicate pruning mechanism in UCS. In DA, $dist[x]$ is decreased if the known path to x via its parent is smaller than the previously known path (which might be ∞). This is exactly the duplicate check of UCS. A node is generated by adding the edge cost to the g-value of its parent. Then, we compare this to the best previously known path and keep only the copy with the minimum g-value.

Differences between DA and UCS

DA and USC are different in many aspects. We discuss this in this section.

Applicability

In DA, *all* the vertices of the input graph are initially inserted to Q. Therefore, in this formalization DA is only applicable in *explicit graphs* where the entire graph (e.g., the set of vertices V and the set of edges E) is explicitly given as input. By contrast, UCS only needs the description of the source vertex along with a list of applicable operators. UCS starts with the source vertex and gradually traverses the necessary parts of the graph. Therefore, it is applicable for both *explicit graphs* and *implicit graphs*.

Memory consumption

The main difference between DA and UCS is which nodes are stored in Q. This directly influences the memory needs. In DA, *all* nodes of the graph are initially inserted to Q. Q always includes *all* the *Unknown* nodes and all the *Opened* nodes. S includes all other nodes (*Closed* nodes). Q starts with $|V|$ nodes and they are dynamically moved from Q to S. However, the memory needs of the data structures of DA is $O(|S| + |Q| = |V|)$ at all times because each node is either in Q or in S. In other words, the entire graph is always stored in the different data structures of DA.

By contrast, while CLOSED of UCS is identical to S of DA, OPEN of UCS is much smaller than Q of DA. OPEN only includes nodes with $dist[] \neq \infty$ but no nodes with $dist[] = \infty$. This is a great advantage of UCS over DA. In UCS, OPEN and CLOSED start with sizes 1 and 0, respectively. They grow until the algorithm halts when a goal was expanded in the *source-target shortest-path* problem or until $|CLOSED| = |V|$ in the *single-source shortest-path* problem. So, at any given time step, the memory needs of DA is larger than that of UCS. For example, assume a case of a very large graph where the goal node is relatively close to the source node. UCS will clearly demand a much smaller amount of memory than DA in this case.

Running Time

The same reasoning applies to the time overhead of manipulating OPEN or Q. In general, the time complexity for all operations of a priority queue Q (i.e., $insert()$, $exctract\_min()$ and $change\_priority()$) for commonly used data structures such as binary heaps is $O(log(n))$ where n is the number of nodes in Q.[2]

It is easy to see that at all times at least one node with $dist \neq \infty$ must exist in Q. Therefore, no node with $dist[] = \infty$ will ever be chosen for expansion by DA. However, in most steps, Q also includes many nodes with $dist[] = \infty$. These nodes will incur a great amount of overhead when performing operations on Q despite the fact that they are not logically needed as they will never be chosen. By contrast, in UCS, OPEN only includes nodes with $dist \neq \infty$, i.e., only the nodes that are logically needed and might be chosen for expansion. Therefore, a much smaller amount of overhead will be caused by the queue of UCS.

For example, consider a graph with one million nodes. Assume that currently 20 nodes are closed and 40 nodes are opened. Q of DA includes nearly 1 million nodes and a basic operation might take $O(log(1,000,000) \approx 20)$ operation. OPEN of UCS only includes 40 nodes (i.e., and a basic operation takes only $O(log(40) \approx 5)$ steps.

Consider the common case where Q is implemented with the *binary heap* data structure. The heap condition might be violated by a node x when (1) its priority is changed by $change\_priority()$, (2) when it is inserted to the heap by $insert()$ operation, or (3) when it replaces the root when the root was removed by $exctract\_min()$. In all these case, a chain of $swap$ operations is performed and x is propagated up or down the heap until the heap condition is met again. In larger heaps (as in DA) the chain of swaps might be larger.

In particular, consider the following observation regarding a node v. In DA, v will be initially inserted to Q

[2]There are advanced data structures, e.g., *Fibonacci heaps* where better bounds can be given for some of the operations.

with $dist[v] = \infty$. Then, if/when v is opened (i.e., $dist[v]$ is changed from ∞), v will incur the overhead of a $change\_priority()$ operation. v will now propagate through other nodes (most of them with $dist[] = \infty$ too) until it reaches its location in the heap. In addition, whenever a *relax* operation further decreases $dist[v]$, $change\_priority()$ is called again. By contrast, in UCS v will only be inserted to OPEN when it is first generated (Opened) with its non-∞ value, canceling the need to perform the first $change\_priority()$ operation and canceling the need to propagate through many nodes with $dist[] = \infty$. A $change\_priority()$ operation will only be called during the duplicate pruning process when a shorter path to v was found (equivalent to the *relax* operation). So, when $|V|$ vertices exist in the graph, DA incurs $|V|$ $insert()$ operation plus $|V|$ $change\_priority()$ operations when the $dist[]$ of these vertices is decreased from ∞. For the same set of operations UCS will only incur $|V|$ $insert()$ operations.

	Max-Q	Swaps	Time
Single-source all shortest paths			
DA	900,435	15,579,369	13.778
UCS	1,802	8,040,484	8.444
Source-target shortest path			
DA	900,435	7,910,200	7.473
UCS	1,469	3,900,530	4.234

Table 1: DA Versus UCS on a small map

Experimental results To demonstrate these differences, we performed simple experiments with the two algorithms on a four-connected squared grid of size 1000×1000 where 10% of the cells were randomly chosen as obstacles. In order to have variable edge costs, the cost of an edge was uniformly randomized to be an integer between 1 and 10. We performed two sets of experiments. In the first set, we randomly chose a source vertex and a goal vertex. Then, both DA and UCS were executed until the shortest path between these two vertices was found. In the second set, only a source vertex was chosen and the algorithms were executed until the shortest paths to *all* vertices in the graph were found. Table 1 shows results averaged over 100 such instances. The first column gives the largest number of vertices kept in the priority queue at any given time. It is easy to see that UCS used a much smaller queue. The second column counts the number of *swap* operations performed in the priority queue in order to preserve the heap condition when $insert$, $change\_priority()$ or $extract\_min()$ were called. Clearly, UCS performs a smaller number of those. Finally the time in seconds needed to solve the problem. Again, UCS outperforms DA in both experiments.

Pedagogical aspects

Consider a basic *Algorithms* course usually given to second year students in computer science or related fields. The first graph algorithms studied are *breadth-first search* and *depth-first search*. Then, the course moves to finding paths in general weighted graphs and the *single-source shortest path* problem is presented. DA is then introduced to the students. Probably, its relation to breadth-first search is not even mentioned or very briefly mentioned. DA, as a stand alone algorithm is rather complex. In DA, the *frontier* of the search is not directly defined - as both *Unknown* and *Opened* nodes are in Q and there is no distinction between them in the way DA is described. Therefore, the different invariants in DA, which actually mimic a best-first search behavior of the frontier are not easy to recognize and understand by students. Proving optimality and the need for non-negative edges needs either an indirect definition of the frontier or an indirect description of the UCS invariants (given above). This will certainly demand large amount of time and effort from both the instructor and the student. Probably, only if a student later takes an *Introduction to AI* course, the best-first search framework is taught and UCS is shown as a very simple special case, and its relation to DA might be mentioned.

By contrast, assume that at this point of the course, instead of DA the instructor presents UCS in two stages. First, the best-first search framework is introduced. The students will not have any problem to understand this framework as they have just studied breadth-first search, which is a simple special case. Second, the instructor introduces UCS as a generalization of breadth-first search to weighted graphs. In order to understand UCS, a student only needs to realize the difference between a FIFO queue and a priority queue, and, between the number of edges in a path $d(n)$ and their weighted sum $g(n)$. Most importantly, the notion of the *frontier* of the search is an inherent component of the structure of best-first search - as the best node of the frontier is chosen for expansion. The frontier is easy to recognize as it is the exact content of OPEN. The frontier nodes in UCS have many important attributes and invariants, as detailed above. These invariants directly explain why UCS provides the optimal solution and why this is only true with non-negative edges as detailed above.

One might claim that the way DA is described where *all* vertices are initially placed in Q is easier to understand than the way it is done lazily in UCS. I disagree. But, even if this claim is true, still, breadth-first search can only be described according to the best-first search framework. The reason is that breadth-first search uses a FIFO queue and thus adding nodes to the queue *must* be done lazily. Given that breadth-first search is described this way, I think that using the same framework of popping a node from the priority queue and inserting all its neighbors would be easier for students to understand. Yet, in most textbooks, breadth-first search is taught along the best-first search framework while DA is taught differently. This causes pedagogical trouble to students and to their instructors.

Practical implementation

Consider a case where an implementer needs to solve the shortest path problem when a heuristic estimation is not given or when the implementer does not know how to use it (e.g., is not even familiar with A*). In such cases, the implementer probably remembers that DA exists and starts to implement it. Then, the implementer might realize that the Q as defined by DA is not efficient and modify the code to only include *Opened* nodes in Q, or in other words, converts DA into UCS. In many other cases, the implementer implements UCS to begin with. Both implementers do not know the term UCS and continue to use the term DA for describing their implementation. This is probably true for scientific papers too. Furthermore, even within the heuristic search community the differences between DA and UCS is not always recognized and/or mentioned. DA is a very well known term and researchers might use this term even if in practice they implement UCS.

My suggestion

The late Dijkstra wrote a very famous paper by the name of “A case against the goto statement” (Dijkstra 1968). Along the same way, this paper can be seen as ”A case against Dijkstra’s algorithm”. My main claim and suggestion is that DA as presented in textbooks should not be taught in classes and not be used by implementers. Instead, UCS should be introduced, taught and used.

I am definitely not arguing against using the name “Dijkstra’s algorithm” for UCS. Dijkstra was the first to mention this logical behavior.“Dijkstra’s algorithm” is a very common term which is widely used and calling an algorithm or a hypothesis after the person who invented it, is very common in science. My claim is that the UCS framework should be used instead of that of DA.

In addition, I also suggest use the term *weighted breadth-first search* instead of UCS. As explained above, the two algorithms are very similar where the only difference is the the weight of the edges.

What would Dijkstra say?

It would have been be very interesting to hear Dijkstra’s opinion on this matter. Unfortunately, he passed away in 2002. However, I went back and read his original seminal paper: “A Note on Two Problems in Connexion with Graphs” (Dijkstra 1959).[3] The paper is very short and only includes one page for this fundamental algorithm. Amazingly, Dijkstra recognizes exactly the three sets of vertices (*Unknown*, *Opened* and *Closed*) described above. He denotes them as the sets C, B and A, respectively. Dijkstra (who denotes the source vertex by P) specially writes that in the main loop, “the node with minimum distance from P is transferred from set B to set A”. While he does not give an exact pseudo code, this can be interpreted more like UCS which only keeps the frontier (set B) in the queue but not all nodes (those with infinity) as in DA. To be on the safe side, one might say that Dijkstra was vague about the matter in question. But certainly he never wrote specifically that *all* nodes should be added to the queue at the initialization phase. Therefore, this paper is targeted against the way DA is commonly described but not against Dijkstra himself.

I would to point out the following two notes. First, Dijkstra specifically addresses the *source-target shortest path*. In his words, he is interested to:“find the path of minimum total length between two given nodes P and Q”. Probably the extension to *single-source shortest path* was made later by others. Second, when Dijkstra wrote his paper, priority queues did not yet exist. He probably meant to use a *sorted-list* of the elements. In this case the difference in the time overhead between DA and UCS is less significant. In DA, all nodes with $dist[] = \infty$ are at the tail of the list and they cause no overhead. Nodes with $dist[] \neq \infty$ should only propagate through nodes at the head of the list. When the priority of a node with $dist[] = \infty$ is first changed to $dist[] \neq \infty$ it can jump at a constant cost to the head of the list, or to a pointer indicating the border between $non-\infty$ and ∞ nodes. However, when priority queues were introduced, including nodes with $dist[] = \infty$ makes a huge difference on the running time as explained above. Based on this, the following conjecture might be a possible historical explanation. In the early days, it was not that harmful (time-wise) to insert *all* nodes to Q. Thus, for some reason textbooks decided to adopt this formalization. Then, when priority queues were introduced, the DA formalization was so common and was never reconsidered.

Algorithms books are usually written by theoretical computer scientists. They might claim, that in the worst-case (e.g. when the graph is fully connected) the queue will look similarly in both DA and UCS so from the theoretical point of view, there is no difference. However, as explained in this paper, in practice there are many advantages in using UCS and not DA.

Acknowledgments

I would like to thank Robert C. Holte and Carlos Linares Lopez for helpful discussions. Special thanks to Eyal Weissman and Shahar Kosti for helping with the experiments.

References

Aho, A. V.; Hopcroft, J. E.; and Ullman, J. D. 1987. *Data Structures and Algorithms*. Addison-Wesley.

Cherkassky, B. V.; Goldberg, A. V.; and Radzik, T. 1996. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming* 73:129–174.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2001. *Introduction to Algorithms*. Cambridge, Massachusetts: MIT Press. 2nd edition.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

Dijkstra, E. W. 1968. A case against the go to statement - was published as a letter entitled go-to statement considered harmful. *Commun. ACM, 11* 3:147–148.

Felner, A.; Zahavi, U.; Holte, R. C.; Schaeffer, J.; Sturtevant, N. R.; and Zhang, Z. 2010. Inconsistent heuristics in theory and practice. *Artificial Intelligence Journal* 175:1570–1603.

Sedgewick, R., and Wayne, K. 2011. *Algorithms*. Pearson Education. 4th edition.

Sniedovich, M. 2006. Dijkstras algorithm revisited: the dynamic programming connexion. *Control and Cybernetics* 35(3):599–620.

[3]This paper can be found at *http://www-m3.ma.tum.de/foswiki/pub/MN0506/WebHome/dijkstra.pdf*.