Network Software Modelling Assignment 1

Student:
Louis Carnec

Student #:
15204934

March 11, 2017

Louis Carnec
15204934

1. Unidirectional Dijkstra Algorithm

The shortest path problem involves finding a path between; a source and destination node, a source and all other nodes, or all other nodes and a single node, such that the weight of the edges separating them is minimised. Although we would assume that a point to point problem would be faster than a single source, the time complexity is actually the same is only reduced by a constant factor [3]. The problem assumes that no negative cycles exist, otherwise no solution exists as the minimum would be negative infinity. Dijkstra's algorithm uses a labelling method. A set of permanent nodes is initialised to which nodes whose shortest path (SP) is known are added. From the start, it includes the source node whose SP we know (distance of zero). A list S of temporary nodes is kept, these nodes are labelled with their estimated distance or with an infinity label if they have not yet been visited. While the list of temporary nodes is not empty, a scan procedure will be applied to the labelled node u with the lowest estimated distance from source (s) coming from the temporary set. The scan procedure involves updating the estimated distance to the source node of each neighbouring node v (of u). Once the scan procedure is finished, node u is marked as permanent. By scanning nodes that are closest first and then assigning them to the permanent set (where the distance to that node cannot change), the algorithm ensures that each node can only be scanned once. The actual runtime of the algorithm can be shortened in the point to point case by terminating once the destination node has been scanned. The original Dijkstra's algorithm using an unordered list has time complexity $O(v^2)$. The outer loop, choosing the node v with lowest distance takes $O(v)$ time. Each iteration of the inner loop takes at most $O(v)$ time as updating the distance of (at most) v nodes takes $O(1)$ time. Time complexity can be reduced to $O(elogv)$ using a binary heap, where $O(logv)$ is the time taken for a node to filter through the heap.

2, 3 and 4. Bidirectional Dijkstra Algorithm [1]

The bidirectional Dijkstra algorithm can reduce running time for point to point shortest path search. The algorithm runs forward (source to destination) and backward (destination to source) search simultaneously (alternates between them). The data structures are slightly different as the permanent set P, temporary set S, parent pointers and distance estimates tuples need to be kept for both the forward and backward search. The labelling and scanning procedures are the same as the unidirectional Dijkstra, however in the backward search the nodes are the other way around, the adjacency graph is reversed, instead of looking for the least-weight edge in the adjacency matrix from u to v, it searches the least-weight edge from v to u. The algorithm stops when a node has been scanned in both directions, that is in practice when a node has been scanned from the list S in one of the directions and is in the permanent set P in the other search direction. However, the shortest path may not necessarily pass through the node the algorithm terminated on, a shorter path may exist. Once terminated, the algorithm must look for the node x in the distance tuple where the sum of the distance to x in the forward search and backward search is minimised ($min\ d_f(x) + d_b(x)$), the shortest path passes through

this node. See Figure 1, Dreyfus [2] shows that the algorithm would terminate on node F through which the path from s to t is of length 10, when in fact the shortest path goes through C with length 9. Like in Dijkstra's original algorithm the parent pointers are used to find the path, the path from x to source is found from the forward search distance tuple and from x to destination in the backward search distance tuple.

Like Dijkstra's algorithm there must be no negative cycles for a solution to exist, thus we run a graph where the edge weights are non-negative. A path between the source and the destination must exist, in my implementation edge weights are uniformly distributed between 0.0 and 1.0, thus some edges have weight zero and it is possible for no shortest path to exist between source and destination. The shortest path from s to t must also be the shortest path from t to s, that is a reverse graph must exist where $G_{uv} = G_{vu}$, as a result the algorithm still works in the case of directed graphs.

Figure 2 demonstrates the difference between the algorithms where the left diagram is the search from s to t of the unidirectional Dijkstra algorithm and the right diagram the search from s to t and t to s of the bidirectional Dijkstra. The sphere around s in the left diagram is the search space for the unidirectional Dijkstra, search expands in a sphere like manner as it iterates across the nodes by order of least weight. Because bidirectional Dijkstra expands from both source and destination, the spheres are half the radius of the unidirectional Dijkstra's sphere. Whereas unidirectional Dijkstra terminates when the target t is scanned, bidirectional Dijkstra terminates when some node w (in the case of our example diagram in Figure 2, node f) has been scanned in both the forward and backward direction. It is more efficient because the volume of search space ($2 \times \pi \times r/2 \times r/2$) is half of unidirectional Dijkstra ($\pi \times r \times r$). Worst-case time complexity for bidirectional Dijkstra is still $O(v^2)$ however, the inner loop is worst case $O(v)$ (v by $O(1)$ distance updates) and the outer loop is still $O(v)$ for the forward and backward search.

5, 6 and 7. Implementation

In implementing both algorithms we use a binary heap for the temporary priority list, allowing us to pop out the lowest weight node from the left of the list, this reduces the worst-case time complexity from $O(v^2)$ to $O(elogv)$. The inner for loop's statements are conducted $O(v + e)$ times where the outer for-loop is has v nodes and e edges for those nodes in the inner-loop. Pushing the heap within the inner-loop takes $O(logv)$ time. Worst-case complexity is thus $O(elogv)$.
For large graphs, the probability that no path exists is higher (some edges might have weight of 0). See Table 1 and Figures 3 and 4 for run-times of the algorithm implementations on graphs of size 5 to 10,000 nodes. Note: the times are for one run only, not an average of multiple runs therefore these results are not conclusive.
In conclusion it seems from observing run time results for different runs (and the run presented here in t Table 1) that the bidirectional Dijkstra algorithm using a binary heap is in practice faster than the unidirectional Dijkstra using a binary heap. From

Figure 4 we can see that the larger the graph, the greater the reduction in runtime from using the bidirectional version.

Louis Carnec
15204934

Appendix

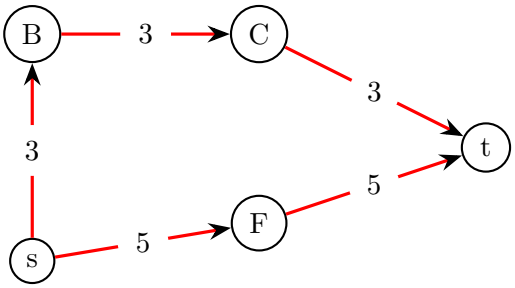

Figure 1: Bidirectional Dijkstra Diagram: shortest path is s→B→C→t

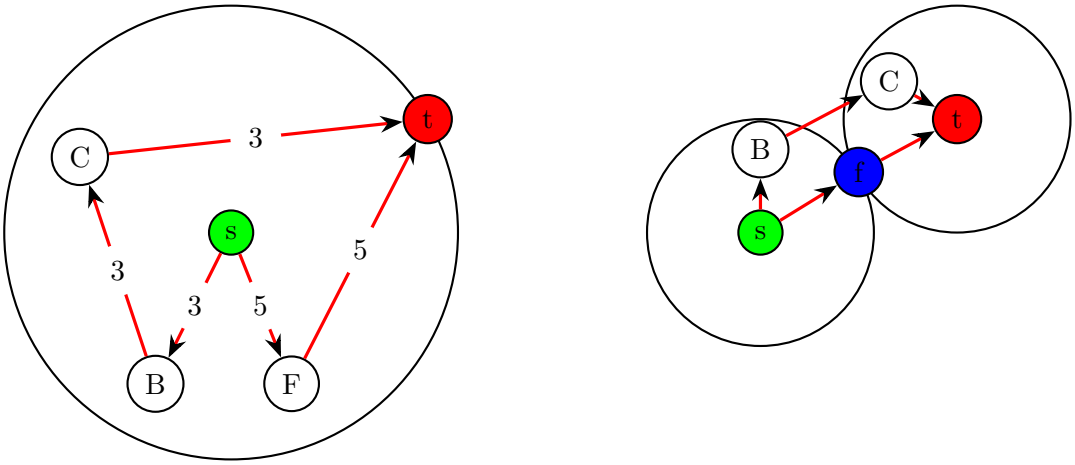

Figure 2: Unidirectional vs Bidirectional Dijkstra Diagram

Louis Carnec
15204934

Table 1: Run-time for unidirectional and bidirectional Dijkstra algorithms using binary heap data structure

n	Unidirectional Dijkstra	Bidirectional Dijkstra
5	5.817413330078125e-05	5.0067901611328125e-05
10	6.508827209472656e-05	0.0001049041748046875
20	0.00014209747314453125	0.00019097328186035156
50	0.0004968643188476562	0.0005588531494140625
100	0.001569986343383789	0.0014600753784179688
1000	0.14145898818969727	0.036271095275878906
10000	10.384804010391235	1.105468988418579

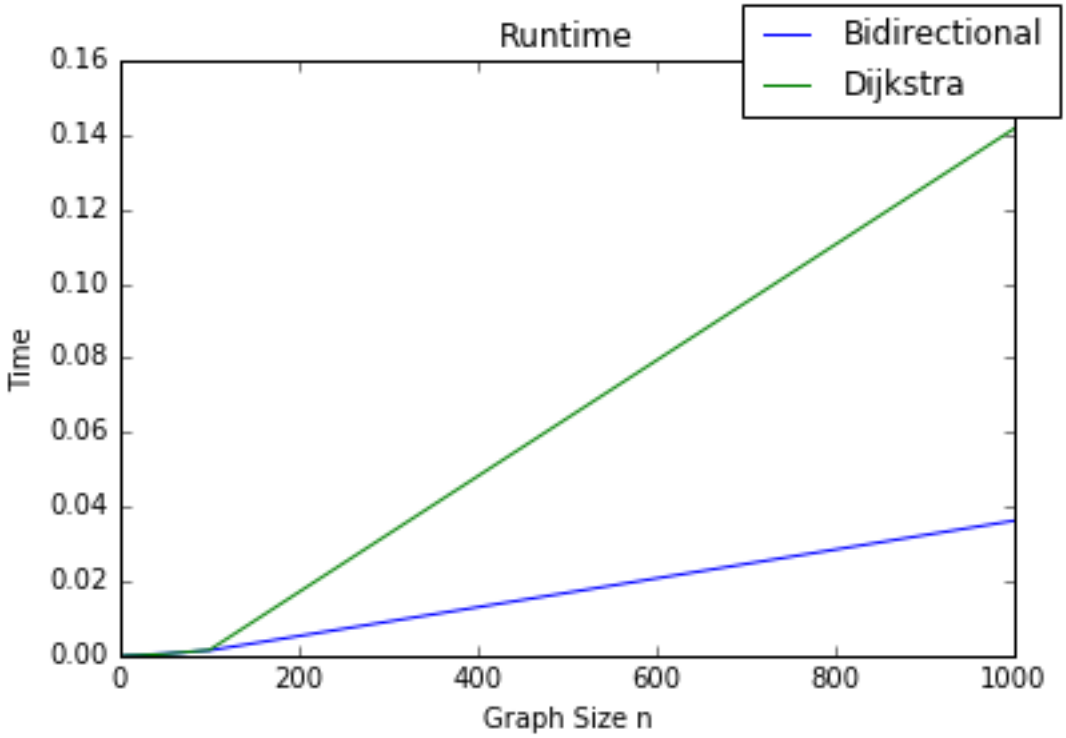

Figure 3: Runtime Unidirectional and Bidirectional Dijkstra with binary heap

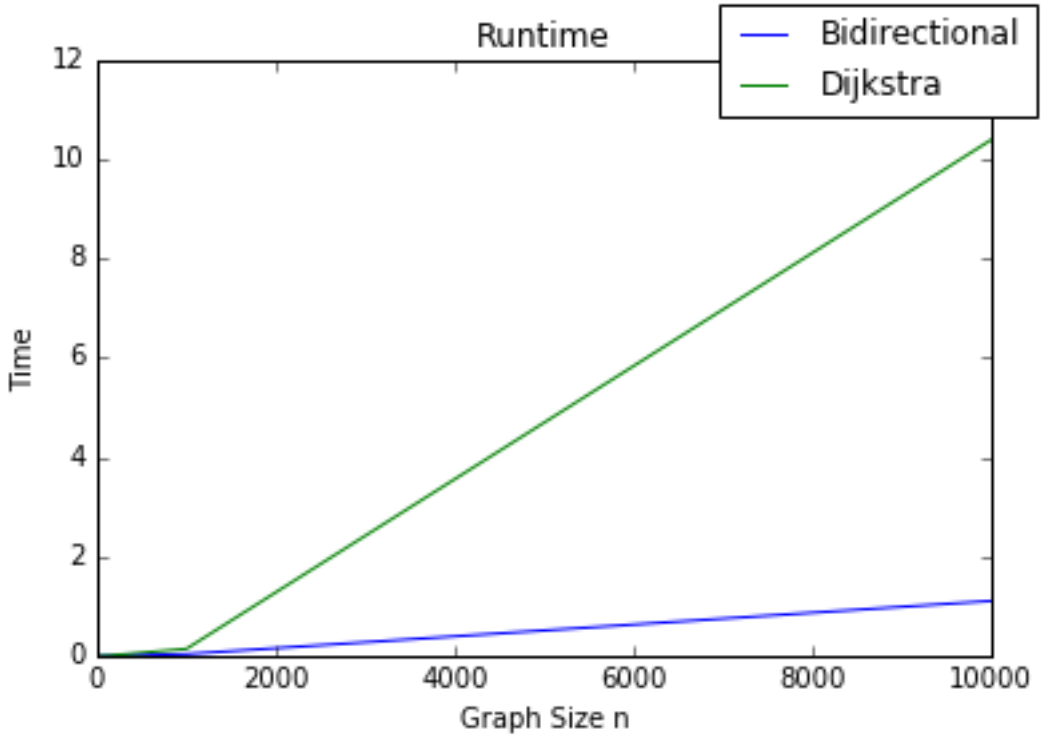

Figure 4: Runtime Unidirectional and Bidirectional Dijkstra with binary heap

Louis Carnec
15204934

References

[1] Mitopencourseware lecture 18: Speeding up dijkstra.

[2] Dreyfus, S. E. An appraisal of some shortest-path algorithms. *Operations research 17*, 3 (1969), 395–412.

[3] Johansson, D. An evaluation of shortest path algorithms on real metropolitan area networks, 2008.