



## UvA-DARE (Digital Academic Repository)

### Framework for path finding in multi-layer transport networks

Dijkstra, F.

**Publication date**

2009

**Document Version**

Final published version

[Link to publication](#)

**Citation for published version (APA):**

Dijkstra, F. (2009). *Framework for path finding in multi-layer transport networks*.

**General rights**

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

**Disclaimer/Complaints regulations**

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

# Appendix A

## Algorithm Time Complexity

### A.1 Running Time of Multi-Layer Path Finding

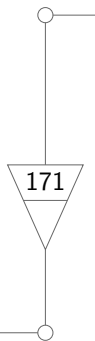
In both the Multi-Layer-Breadth-First and Multi-Layer-k-Shortest-Path algorithms (listings 7.1 and 7.4 respectively) the running time will mostly depend on the size of the queue  $Q$ . The best estimate we can give is to first estimate the length of the shortest path, and then estimate the number of paths of that length. Since the algorithms are basically a flooding mechanism, we assume that the path branches at each hop, and the number of branches after  $i$  hops is  $\mathcal{O}(r^i)$ , with  $r$  the number of branches per hop. If we ignore all suppression mechanisms, as we should for the worst-case scenario, the queue length is proportional to the number of branches,  $\mathcal{O}(r^i)$ . The number of branches per hop  $r$  is roughly proportional to the average out-degree and to the number of possible labels  $|\langle Lb \rangle|$  per layer. The average out-degree is the average number of adjacencies:

$$|\langle Adj \rangle| = \frac{|\mathcal{E}_l|}{|\mathcal{V}_l|} \quad (\text{A.1})$$

Now we can estimate the queue length.

$$\begin{aligned} \mathcal{O}(|Q|) &= \mathcal{O}(r^i) \\ &= \mathcal{O}\left(\left(\frac{|\mathcal{E}_l| \times |\langle Lb \rangle|}{|\mathcal{V}_l|}\right)^{|\mathcal{V}|}\right) \\ &= \mathcal{O}\left(\left(\frac{(|N| \times |\mathcal{Y}| + |L|) \times |\langle Lb \rangle|}{|N| \times |\mathcal{Y}|}\right)^{(|N| \times |\mathcal{Y}|)}\right) \end{aligned} \quad (\text{A.2})$$

Assuming that our network is a small world network, the average path length is  $\mathcal{O}(\log(|\mathcal{V}|))$  [p1]. While the estimate of the worst-case remain the



same, the estimate for the *average* running time would reduce  $i$  from  $|\mathcal{V}|$  to  $\log(|\mathcal{V}|)$ :

$$\begin{aligned}
 \mathcal{O}(|Q|) &= \mathcal{O}(r^i) \\
 &= \mathcal{O}\left(\left(\frac{|\mathcal{E}_l| \times |\langle Lb \rangle|}{|\mathcal{V}_l|}\right)^{\log(|\mathcal{V}|)}\right) \\
 &= \mathcal{O}\left(\left(\frac{(|N| \times |\mathcal{Y}| + |L|) \times |\langle Lb \rangle|}{|N| \times |\mathcal{Y}|}\right)^{\log(|N| \times |\mathcal{Y}|)}\right)
 \end{aligned} \tag{A.3}$$

Such exponential behaviour is typical for a NP-complete problem, such as a path-constrained path finding algorithm.

## A.2 Multi-Layer Dijkstra's Algorithm

**Listing A.1** shows MULTI-LAYER-DIJKSTRA, a variant of the Dijkstra algorithm applied to the graph  $G_s$  as defined in **section 7.5**.

This algorithm is an improvement over Dijkstra's algorithm [p11]. Dijkstra's algorithm applied to the graph  $G_s$  in **figure 7.4** would find the path  $A_{Eth} - B_{Eth} - B_{24c} - E_{24c} - D_{24c} - D_{Eth} - D_{3c7v} - E_{3c7v} - F_{3c7v} - F_{Eth} - C_{Eth}$  as shortest path from  $A$  to  $C$ . This is not correct, due to the limited capacity between  $D$  and  $E$ . The Multi-Layer-Dijkstra algorithm in **listing A.1** would find the correct shortest path,  $A_{Eth} - B_{Eth} - B_{24c} - E_{24c} - D_{24c} - D_{Eth} - D_{3c7v} - B_{3c7v} - E_{3c7v} - F_{3c7v} - F_{Eth} - C_{Eth}$ .

Nevertheless, the Multi-Layer-Dijkstra algorithm is still imperfect. For example, while it finds the shortest path from  $A$  to  $C$ , it will not find the shortest path from  $C$  to  $A$ . The reason is that this algorithm does check for the used bandwidth, but it only keeps track of the bandwidth usage as a global variable, rather than per path. This means that it adds edges to the list of used bandwidth, even if that edge later turns out not be used anymore. This condition is too strict, resulting in false negatives, such as the above.

Lines 1-6 of the meta-code initialises all vertices and edges. Line 7 inserts all vertices in the queue  $Q$ . The main algorithm starts at line 8. Line 9 extracts the vertex  $u$  from the queue that has the shortest weight (i.e.,  $d[u] \leq d[v] \forall v \neq u \in Q$ ). Vertex  $u$  can be regarded as the new scanning vertex towards destination  $v_{dst}$ . Consequently, we have to reduce the bandwidth of the last edge in the path to vertex  $u$  with the amount of consumed bandwidth (which in this case we take from the edge parameter  $B_e(e)$ ). Lines 13 and 14 make sure we only retrieve the shortest path between a source and a single destination (as

---

**Algorithm A.1** MULTI-LAYER-DIJKSTRA( $G_s, v_{src}, v_{dst}$ )

---

**Require:** Available bandwidth  $B_e(e)$  for each edge  $e$

**Require:** Required bandwidth for each vertex  $v = (n, s)$

```

1: for all vertices  $v \in \mathcal{V}_s$  do
2:    $d[v] \leftarrow \infty$  {The distance to  $v_{src}$ }
3:    $\pi[v] \leftarrow \text{NIL}$  {the predecessor vertex to  $v_{src}$ }
4: for all edges  $(u, v) \in \mathcal{E}_s$  do
5:    $b((u, v)) = B_e((u, v))$  {Still available bandwidth}
6:  $d[v_{src}] \leftarrow 0$ 
7: Queue  $Q \leftarrow \mathcal{V}_s$ 
8: while  $Q \neq \emptyset$  do
9:    $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
10:  if  $\pi[u] \neq \text{NIL}$  then
11:     $(n, s) \leftarrow u$ 
12:     $b(u, \pi[u]) \leftarrow b(u, \pi[u]) - \text{BANDWIDTH-REQUIRED}(s)$ 
13:  if  $u = v_{dst}$  then
14:    return path {Created by backtracing  $\pi[v]$ , starting with  $v_{dst}$ }
15:  else
16:    for all  $v \in \text{adj}[u]$  do {for each neighbour of  $u$ }
17:      if  $d[v] > d[u] + w(u, v)$  then {Distance is sum of weights}
18:        if  $b(u, v) \geq 0$  then
19:           $d[v] \leftarrow d[u] + w(u, v)$  {Since  $Q = \mathcal{V}_s$ , this changes the queue}
20:           $\pi[v] \leftarrow u$ 
21:           $\text{ENQUEUE}(Q, v)$ 

```

---

is our purpose). Lines 16 to 20 perform the relaxation procedure [p9] for each adjacent vertex  $v$  of  $u$ . Line 18 is not present in the original Dijkstra algorithm, but is necessary to check for enough available bandwidth on the scanned link. Line 21 is an extension to the original Dijkstra algorithm, which is necessary to cope with negative edge weights. It inserts  $v$  back into the queue, if it was previously extracted.

### A.3 Running Time of Multi-Layer-Dijkstra

The running time of [algorithm A.1](#) is slightly longer than Dijkstra's algorithm due to the extension in line 20, which re-inserts edges into the queue. If we

ignore this extension, the running time is:

$$\mathcal{O}(\text{Algorithm A.1}) = \mathcal{O}(|\mathcal{V}_s| \times \mathcal{O}(\text{EXTRACT-MIN}) + |\mathcal{E}_s| \times \mathcal{O}(\text{INSERT})) \quad (\text{A.4})$$

In here,  $\mathcal{O}(\text{EXTRACT-MIN})$  is caused by line 9 and  $\mathcal{O}(\text{INSERT})$  is caused by line 21. If the graph is sufficiently sparse ( $|\mathcal{E}| < |\mathcal{V}|^2$  [p9]), this equation reduces to  $\mathcal{O}(|\mathcal{V}_s| \times \log(|\mathcal{V}_s|) + |\mathcal{E}_s| \times \mathcal{O}(1)) = \mathcal{O}(|\mathcal{V}_s| \cdot \log(|\mathcal{V}_s|) + |\mathcal{E}_s|)$ .

Equation 7.34 estimates  $|\mathcal{E}_s| \approx (|A| + |L|) \times T^{|\mathcal{Y}|}$ , and we assume that  $\mathcal{O}(|A|) = \mathcal{O}(|N| \times |\mathcal{Y}|)$ . Equation 7.29 gives the upper limit  $|\mathcal{V}_s| \approx |N| \times |\mathcal{S}| \approx |N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}$  with  $T = \langle |T(y)| \rangle$  the average number of technologies for each stack.

The running time of algorithm A.1 with constant edge weights  $W_e(e)$  is:

$$\begin{aligned} \mathcal{O}(\text{Algorithm A.1}) &= \mathcal{O}(|\mathcal{E}_s| + |\mathcal{V}_s|) \\ &= \mathcal{O}((|A| + |L|) \times T^{|\mathcal{Y}|} + |N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}) \\ &= \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|} + |N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}) \\ &\approx \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \end{aligned} \quad (\text{A.5})$$

The running time of algorithm A.1 with variable edge weights  $W_e(e)$  is:

$$\begin{aligned} \mathcal{O}(\text{Algorithm A.1}) &= \mathcal{O}(|\mathcal{E}_s| + |\mathcal{V}_s| \cdot \log(|\mathcal{V}_s|)) \\ &= \mathcal{O}((|A| + |L|) \times T^{|\mathcal{Y}|} + |N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|} \\ &\quad \times \log(|N| \times |\mathcal{Y}| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|})) \\ &= \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|} + |N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|} \\ &\quad \times (\log(|N|) + \log(|\mathcal{Y}|) + |\mathcal{Y}| \cdot \log(T))) \end{aligned} \quad (\text{A.6})$$

## A.4 Running Time of Multi-Layer-Breadth-First

The rough estimate of the running time in the previous section does not provide many insights. In this section, we will assume that the segment of a shortest path is also a shortest path. This means we can abort any path if it contains a (node, adaptation stack) tuple that we encountered before. Such algorithm would be comparable to Multi-Layer-Dijkstra, as described above. This allows us to do a more thorough comparison of running times between path finding in  $G_l$  and path finding in  $G_s$ .

If we can abort a path if the current vertex has been processed with the same adaptations stack  $s$ , then each vertex is processed at most  $|\mathcal{S}_y|$  times,

#### A.4. RUNNING TIME OF MULTI-LAYER-BREADTH-FIRST

with  $|\mathcal{S}_y|$  the number of possible technology stacks for layer  $Y_c(v)$  of vertex  $v$ . The worst-case of  $|\mathcal{S}_y|$  is  $|\mathcal{S}_{|\mathcal{Y}|}|$ , or  $\prod_{y \in \mathcal{Y}} |T(y)|$  according to [equation 7.26](#).

Recall that average adjacency can be found by dividing the number of edges by the number of vertices in a graph ([equation A.1](#)).

According to [equation 7.35](#), the running time of [algorithm 7.1](#) is:

$$\begin{aligned}
 \mathcal{O}(\text{Algorithm 7.1}) &= \mathcal{O}(|Q|) \times \mathcal{O}(\text{loop}) = \\
 &= \mathcal{O}(|Q|) \times (\mathcal{O}(\text{DEQUEUE}) + \mathcal{O}(|adj|) \times \mathcal{O}(\text{EXTEND-PATH})) \\
 &= \mathcal{O}(|Q|) \times (\mathcal{O}(\text{DEQUEUE}) + \frac{\mathcal{O}(|\mathcal{E}_l|)}{\mathcal{O}(|\mathcal{V}_l|)} \times \mathcal{O}(\text{EXTEND-PATH}))
 \end{aligned} \tag{A.7}$$

With the restriction in place, the queue size is limited to one adaptation stack per node. Since there are at most  $|\mathcal{S}|$  adaptations stacks, the upper limit of  $|Q|$  is equal to

$$\begin{aligned}
 \mathcal{O}(|Q|) &= \mathcal{O}(|N| \times |\mathcal{S}|) \\
 &\lesssim \mathcal{O}(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|})
 \end{aligned} \tag{A.8}$$

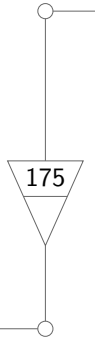
$|Q|$  is equivalent to  $|\mathcal{V}_s|$ , since  $\mathcal{V}_s$  is also determined by the number of adaptation stacks per node. We have seen in [equations 7.25](#) and [7.27](#) that the *estimate* of  $|\mathcal{S}|$  is lower than its upper limit by a factor of  $|\mathcal{Y}|$ . The *estimate average* of  $|Q|$  is equal to

$$\begin{aligned}
 \mathcal{O}(|Q|) &= \mathcal{O}(|N| \times |\mathcal{S}|) \\
 &= \mathcal{O}(|N| \times T^{|\mathcal{Y}|})
 \end{aligned} \tag{A.9}$$

These results apply to both the Multi-Layer-Breadth-First and Multi-Layer-k-Shortest-Path algorithm with restricted search space.

If we assume  $\mathcal{O}(|A|) = \mathcal{O}(|N| \times |\mathcal{Y}|)$ , and use [equation 7.12](#) as the estimate of  $|\mathcal{V}_l|$  and  $|\mathcal{E}_l|$ , we can expand [equation A.7](#):

$$\begin{aligned}
 \mathcal{O}(\text{Algorithm 7.1}) &= \mathcal{O}(|Q|) \times (\mathcal{O}(\text{DEQUEUE}) + \frac{\mathcal{O}(|\mathcal{E}_l|)}{\mathcal{O}(|\mathcal{V}_l|)} \times \mathcal{O}(\text{EXTEND-PATH})) \\
 &= \mathcal{O}(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}) \times \\
 &\quad (\mathcal{O}(\text{DEQUEUE}) + \frac{\mathcal{O}(|A| + |L|)}{\mathcal{O}(|N| \times |Y|)} \times \mathcal{O}(\text{EXTEND-PATH})) \\
 &= \mathcal{O}(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\text{DEQUEUE}) + \\
 &\quad \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\text{EXTEND-PATH})
 \end{aligned} \tag{A.10}$$



## APPENDIX A. ALGORITHM TIME COMPLEXITY

The only operations in the path extension subroutine ([algorithm 7.2](#)) which have a running time larger than  $\mathcal{O}(1)$ , are the operations on lines 6, 9 that checks for duplicate tuples (vertex, stack) using  $R$ , and the operation on line 22 which checks the number of labels per layer  $\langle Lb \rangle$ . Both operations depend if  $R$  and the labels can be sorted. If no sorting is possible, the running time is  $\mathcal{O}(\langle R \rangle) = \mathcal{O}(|Q|)$  and  $\mathcal{O}(\langle lb \rangle)$  respectively. If sorting is possible, the running time is  $\mathcal{O}(\log(|Q|))$  and  $\mathcal{O}(\log(\langle lb \rangle))$  respectively.

$$\mathcal{O}(\text{EXTEND-PATH}) = \mathcal{O}(\log(|Q|) + \log(\langle lb \rangle)) \quad (\text{A.11})$$

The original breadth first search algorithm can only deal with edge lengths of 1 ( $W_e(e) = 1$  for all  $e$ ). The advantage is that the DEQUEUE operation only takes  $\mathcal{O}(1)$ , since the queue is sorted in order of path length. If we allow different  $W_e(e)$ , this function needs to be replaced with an EXTRACT MIN operation. The time complexity of this operation becomes  $\mathcal{O}(\log(|Q|))$ , provided that the queue  $Q$  is sorted using a Fibonacci heap [p9].

$$\mathcal{O}(\text{DEQUEUE operation}) = \mathcal{O}(1) \quad (\text{A.12})$$

$$\mathcal{O}(\text{EXTRACT-MIN operation}) = \mathcal{O}(\log(|Q|)) \quad (\text{A.13})$$

The queue length  $|Q|$  highly depends on how quickly the flooding principle is suppressed by the incompatibility check and duplicate stack check. The worst-case is given in [equation A.8](#).

If we assume  $W_e(e) = 1$ ,  $\mathcal{O}(\text{EXTEND-PATH}) = \mathcal{O}(\log(|Q|))$  (no labels), the worst-case running time for [algorithm 7.1](#) becomes:

$$\begin{aligned} \mathcal{O}(\text{Algorithm 7.1}) &= \mathcal{O}(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\text{DEQUEUE}) + \\ &\quad \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\text{EXTEND-PATH}) \\ &= \mathcal{O}(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}) \times \mathcal{O}(1) + \\ &\quad \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(|Q|)) \\ &\approx \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|})) \end{aligned} \quad (\text{A.14})$$

We can now calculate the worst-case running time of [algorithm 7.1](#) for

## A.5. RUNNING TIME OF MULTI-LAYER-K-SHORTEST-PATH

variable edge weights  $W_e(e)$ :

$$\begin{aligned}
 \mathcal{O}(\text{Algorithm 7.1}) &= \mathcal{O}(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\text{DEQUEUE}) + \\
 &\quad \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\text{EXTEND-PATH}) \\
 &= \mathcal{O}(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(|Q|)) + \\
 &\quad \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(|Q|) + \log(\langle lb \rangle)) \\
 &\approx \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(|Q|) + \log(\langle lb \rangle)) \\
 &= \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \\
 &\quad \mathcal{O}((\log(|N|) + \log(|\mathcal{Y}|) + |\mathcal{Y}| \cdot \log(T)) + \log(\langle lb \rangle))
 \end{aligned} \tag{A.15}$$

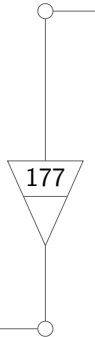
Even so, the *estimated average* running time of [algorithm 7.1](#) is:

$$\begin{aligned}
 \mathcal{O}(\text{Algorithm 7.1}) &= \mathcal{O}(|Q|) \times \mathcal{O}(\text{DEQUEUE}) + \\
 &\quad \mathcal{O}((|N| + \frac{|L|}{|\mathcal{Y}|}) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\text{EXTEND-PATH}) \\
 &= \mathcal{O}(|Q|) \times \mathcal{O}(\log(Q)) + \\
 &\quad \mathcal{O}((|N| + \frac{|L|}{|\mathcal{Y}|}) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(Q) + \log(\langle lb \rangle)) \\
 &= \mathcal{O}(|N| \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(|N| \times T^{|\mathcal{Y}|})) + \\
 &\quad \mathcal{O}((|N| + \frac{|L|}{|\mathcal{Y}|}) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(|N| \times T^{|\mathcal{Y}|}) + \log(\langle lb \rangle)) \\
 &\approx \mathcal{O}((|N| + \frac{|L|}{|\mathcal{Y}|}) \times T^{|\mathcal{Y}|}) \times (\log(|N| \times T^{|\mathcal{Y}|}) + \log(\langle lb \rangle))
 \end{aligned} \tag{A.16}$$

## A.5 Running Time of Multi-Layer-k-Shortest-Path

The running time of Multi-Layer-k-Shortest-Path ([algorithm 7.4](#)) is roughly comparable to the running time of Multi-Layer-Breadth-First ([algorithm 7.1](#)) as it is basically the same algorithm.

The only difference in the running time is caused by two factors. First, Multi-Layer-k-Shortest-Path reduces the search space by using the estimate path length. This reduces the average running time by aborting path that are unfeasible due to their length. At the same time, this change *increases* the





## APPENDIX A. ALGORITHM TIME COMPLEXITY

worst-case running time, because in the worst-case, no paths are aborted, and the running of Dijkstra's algorithm adds to the running time.

The second difference stems between the two algorithms from the fact that Multi-Layer-k-Shortest-Path operates on a larger graph than Multi-Layer-Breadth-First. While this may seem worse in terms of number of vertices, the running time is nearly equivalent in terms of devices and technology layers. In fact, any algorithm running on  $G_s$  is slightly faster than the equivalent algorithm running on  $G_l$ . The reason is that  $G_s$  contains more intrinsic information. For example, the loop check (lines 6-9 in [algorithm 7.1](#), line 6 in [algorithm 7.5](#)) is more expensive in  $G_l$  than in  $G_s$ , since it has to search through a list of possible adaptations in  $G_l$  while the adaptation function is immediately obvious from the vertex in  $G_s$ . So, while the creation of the graph  $G_s$  is more computational intensive than the creation of the graph  $G_l$ , this drawback is a benefit when running the algorithm. Finally,  $G_s$  may be slightly more efficient, since  $G_s$  can collapse multiple links on the same edge, while this is not done in  $G_l$ . Again, this advantage during the algorithm running time is offset by a disadvantage when generating the graph  $G_s$ .

The worst-case running time of Multi-Layer-k-Shortest-Path is:

$$\mathcal{O}(\text{Algorithm 7.4}) = \mathcal{O}(|\mathcal{V}_s|) + \mathcal{O}(|\langle Q \rangle|) \times (\mathcal{O}(\text{EXTRACT-MIN}) + \mathcal{O}(|adj|) \times (\mathcal{O}(\text{FEASIBLE}) + \mathcal{O}(\text{ENQUEUE}))) \quad (\text{A.17})$$

If again, we assume that  $\mathcal{O}(|\langle Q \rangle|) = \mathcal{O}(|\mathcal{V}_s|) \leq |N| \times |S|$ , and further assume  $\mathcal{O}(\text{EXTRACT-MIN}) = \mathcal{O}(\log(Q))$ ,  $\mathcal{O}(\text{ENQUEUE}) = \mathcal{O}(1)$ , and  $\mathcal{O}(\text{FEASIBLE}) = \mathcal{O}(|p|) = \mathcal{O}(\log(\mathcal{V}_s))$  then we can specify the worst-case running time for MULTI-LAYER-K-SHORTEST-PATH.

$$\begin{aligned} \mathcal{O}(\text{Algorithm 7.4}) &= \mathcal{O}(|\mathcal{V}_s|) + \mathcal{O}(|\mathcal{V}_s|) \times (\mathcal{O}(\log(\mathcal{V}_s)) + \\ &\quad \frac{\mathcal{O}(\mathcal{E}_s)}{\mathcal{O}(\mathcal{V}_s)} \times (\mathcal{O}(\log(\mathcal{V}_s)) + \mathcal{O}(1))) \\ &\approx \mathcal{O}(|\mathcal{V}_s|) \times \mathcal{O}(\log(\mathcal{V}_s)) + \mathcal{O}(|\mathcal{E}_s|) \times \mathcal{O}(\log(\mathcal{V}_s)) \\ &= \mathcal{O}((|\mathcal{V}_s| + |\mathcal{E}_s|) \times \log(\mathcal{V}_s)) \\ &= \mathcal{O}((|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|} + (|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \\ &\quad \log(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|})) \\ &\approx \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|} \times \log(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|})) \end{aligned} \quad (\text{A.18})$$

The *estimated average* running time for MULTI-LAYER-K-SHORTEST-PATH

## A.5. RUNNING TIME OF MULTI-LAYER-K-SHORTEST-PATH

is

$$\begin{aligned}\mathcal{O}(\text{Algorithm 7.4}) &= \mathcal{O}(|\mathcal{V}_s| + |\mathcal{E}_s|) \times \log(\mathcal{V}_s) \\ &= \mathcal{O}\left(|N| \times T^{|\mathcal{Y}|} + \left(|N| + \frac{|L|}{|\mathcal{Y}|}\right) \times T^{|\mathcal{Y}|}\right) \times \log(|N| \times T^{|\mathcal{Y}|}) \\ &\approx \mathcal{O}\left(|N| + \frac{|L|}{|\mathcal{Y}|}\right) \times T^{|\mathcal{Y}|} \times \log(|N| \times T^{|\mathcal{Y}|})\end{aligned}\tag{A.19}$$