Graph Theory for Alignment Algorithms

DAGs, Shortest paths, Longest paths

A directed graph is DAG (directed acyclic graph) if it is connected and has no directed cycles.

- DAGs have very special properties allowing more efficient algorithms.
- The Edit Graphs we construct for local and global alignments are DAGs.

Dijkstra's Algorithm
Shortest paths algorithm from one "source" vertex to all the vertices in the graph.

- We present the algorithm for directed graphs with edges labeled by positive costs/weights.

- Let \( G = (V, E) \) be a directed graph.

- \( V \) is the set of vertices/nodes.

- \( E \) is the set of edges.

- Every edge \((i,j) \in E\) has a non-negative cost/weight/length.

- One node is given as the "source" node.

**Problem Formulation**

**Given:** \( G = (V, E) \), \( V = \{1, 2, \ldots, n\} \)

1 is the source node.

\( L[(i,j)] \) is the cost of edge \((i,j) \in E\).
Compute

The cost of the shortest path from the source node to each of the other nodes in G. And find the shortest path as well.

Intuition about the algorithm:
- There are 2 sets C and S.
  - S = set of nodes already chosen.
  - C = set of candidate nodes.

- At any step of the algorithm, S contains all the nodes whose minimal cost from the source node is known (computed).
- C contains the rest of the nodes not in S. At every step we choose a node v in C with the cost of the path from source to v being minimal (a greedy step) and we add v to S.
A path from the source node to a node \( v \) is **special** if all intermediate nodes along the path from source to \( v \) are in \( S \).

The algorithm uses a matrix \( D \) that stores the length of the **shortest special path** for each vertex in \( G \).

When we add \( v \) (a new node from \( C \)) to \( S \) then the **shortest special path** to \( v \) is also the **shortest path** in \( G \).

The algorithm terminates with all nodes in \( S \).

\( D \) contains at termination the **shortest path** costs for all vertices in \( G \).

\[ V = \{1, 2, \ldots, n\} \]

\[ L_{ij} \geq 0 \text{ if edge } (i, j) \in E \]

\[ L_{ij} = \infty \text{ if edge } (i, j) \notin E. \]
Dijkstra's Algorithm

Begin Initialization

\[ S = \{1\} \]
\[ C = \{2, 3, \ldots, n\} \]

For \( i = 2 \) to \( n \) Do

\[ D[i] = L[i, n] \]

Greedy Loop

Repeat \( n-2 \) times

\[ v = \text{node in } C \text{ with min value } D[v] \]
\[ S = S \cup \{v\} \]
\[ C = C \setminus \{v\} \]

For each \( w \in C \) Do

\[ P[i] = \text{backtracking pointers matrix} \]
\[ P[i] = 1, i = 2, 3, \ldots, n \]
\[ P[2, \ldots, n] \]
\[
\text{IF } D[w] > D[v] + \Delta[v, w] \\text{ THEN } \\text{END}
\]

\[D[w] = D[v] + \Delta[v, w] \\text{ END} \quad \Delta[v, w] = v\]

**An example**

**G:**

The execution of the algorithm on **G**:

<table>
<thead>
<tr>
<th>STEP</th>
<th>V</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initialization</strong> -</td>
<td>${2, 3, 4, 5}$</td>
<td>$[50, 30, 109, 10]$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>${2, 3, 4}$</td>
<td>$[50, 30, 20, 10]$</td>
</tr>
</tbody>
</table>
Backtracking: Finding the shortest paths

1. We use the matrix P of back pointers

2. P[v] contains the vertex in the shortest path from the source to v that proceeds v in the shortest path.

3. To find the complete shortest path from v, follow the pointers backwards till we get to the source vertex.
SHORTEST PATHS PROBLEM
&
LONGEST PATHS PROBLEM
in
DIRECTED ACYCLIC GRAPHS (DAGs)

Note. A longest path between two vertices s and t in a weighted graph G is the same as a shortest path in the graph -G (i.e., the graph obtained from G by changing every weight to its value multiplied by -1).

Therefore, if shortest paths can be found for -G then longest paths can also be found in G.
In a DAG, there are no directed cycles and so there are no negative cost cycles, so the longest paths can be found in $O(1V1 + |E|)$ linear time by applying the linear time algorithm for shortest paths in $-G$.

**Topological Sorting**

A topological sort or topological order of a directed graph is a linear ordering of its vertices such that for every directed edge from vertex $u$ to vertex $v$, vertex $u$ comes before $v$ in the topological order.
THEOREM

A topological order exists for a directed graph if and only if the graph has no directed cycles, i.e. it is a DAG.

Ex.

A DAG

topological order

L = [1, 3, 5, 4, 2, 3]
**Shortest Paths in DAGs via Topological Order**

\[ G = (V, E) \text{ a DAG} \]

\[ |V| = m, \quad V = \{1, 2, \ldots, m\} \]

\[ L = \text{a topological order for } G \]

\[ L[1], L[2], \ldots, L[m] \]

**Shortest Paths Algorithm**

\[ \text{array of length } m \]

\[ d = \text{will contain at termination the shortest paths distance costs from node 1 to all other vertices} \]

\[ d[1] = 0, \quad d[t] = \infty, \quad 2 \leq t \leq m \]

\[ \text{an array for backtracking pointers} \]
\[ p[v_j] = -\infty \quad 1 \leq v \leq n \]
\[ p[v_j] = \text{the node that is} \]
\[ \text{the predecessor of } v \]
\[ \text{in the shortest path} \]
\[ \text{from 1 to } v \]
\[
\begin{align*}
\delta[v_j] &= 0 \\
L &= \text{topological order} \\
\delta[i] &= \infty, \quad 2 \leq i \leq n
\end{align*}
\]

\text{For} \quad i = 1 \quad \text{to} \quad n

\[ u = \delta[i_j] \]

\text{For every } v \text{ such that } (u,v) \in E

\text{let } w = \text{weight}(u,v)

\text{if } \delta[v_j] > \delta[u] + w

\text{then}
\begin{align*}
\delta[v_j] &= \delta[u] + w \\
p[v_j] &= u
\end{align*}

\text{Linear time alg} \quad O(N + M)

N \geq n

M = \# \text{ of edges}
\[ L = \{1, 3, 5, 4, 2\} \]
\[ d[13] = 0 \]

\[ i = 1 \quad u = 1 \]
\( (1, 2) \quad w = 50 \quad d[12] = \infty \Rightarrow P_{12} = 0 \)
\( (1, 3) \quad w = 30 \quad P_{13} = 1 \)
\( (1, 4) \quad w = 100 \quad P_{14} = 1 \)
\( (1, 5) \quad w = 30 \quad P_{15} = 1 \)

\[ i = 2 \quad u = 3 \]
\( (3, 2) \quad w = 5 \quad d[23] = 35 \)
\( (7, 4) \quad w = 50 \quad > 30 + 5 \)
\( P_{23} = 3 \quad P_{24} = 35 \quad P_{43} = 100 \)

\[ i = 3 \quad u = 5 \]
\( (5, 4) \quad w = 80 \quad > 50 + 10 \)
\( P_{54} = 5 \quad P_{45} = 20 \)

\[ i = 4 \quad u = 4 \]
\( (4, 2) \quad w = 35 \quad d[23] = 35 \)
i = 5  u = 2

\[ d[17] = 0 \]
\[ d[23] = 35 \]
\[ d[27] = 30 \]
\[ d[43] = 20 \]
\[ d[57] = 10 \]

\[ \sum 23 = 35 \]
\[ P32J = 4 \]

no edges left

**Topological Sorting**

A topological sort or topological order of a directed graph is a linear ordering of its vertices such that for every directed edge from vertex u to vertex v, vertex u comes before v in the topological order.

**Theorem**
A topological order exists for a directed graph if and only if the graph has no directed cycles, i.e. it is a DAG.

Ex.

A DAG

\[ l = [1, 3, 5, 4, 2, 3] \]

In a DAG a topological order can be found in linear time.
Algorithm for finding a topological order for a DAG.

Input: DAG \( G = (V, E) \)

Output: \( L \) a topological order of \( G \)

begin
\( L = \emptyset \)
\( S = \) set of vertices with no incoming directed edges

while \( S \neq \emptyset \) do

remove vertex \( u \) from \( S \)
add \( u \) to \( L \) at its tail

for each node \( v \) and edge \( e = (u, v) \in E \)

remove edge \( e \) from graph

if \( v \) has no incoming edges

end
THEN put \( v \) into \( S \)

output \( L \) (a topological order)

END.

Ex. Consider a DAG \( G \) (as in Alignnet)

\[
\begin{align*}
L &= \{3\} \\
S &= \{3, 1\} \\
L &= \{1\}
\end{align*}
\]
A topological for $G$