Redesign of Bellman-Ford Shortest Path Algorithm

Ahmed Musa
Sudan University for Science and Technology, Sudan

Abstract: This paper presents redesigns of Bellman-Ford Shortest Path algorithm. Bellman-Ford algorithm performs \(|V| - 1\) scans over the list of edges to relax them, it has no cases. The redesigned algorithms perform only one scan in the best case and \(|V| - 1\) scans in the worst case. The paper also shows that Bellman-Ford and its redesigns can be modified to compute point to point shortest paths.

Key words: Loop invariant, breadth first search, Dijkstra, bidirectional.

1. Introduction

The shortest path between two points \((u, v)\) in a weighted directed or undirected graph \(G(V, E, w)\) is the path which satisfies the condition:

For each point \(w \in V(G)\) research.

\[ P_s(u, w) + P_s(w, v) \geq P_s(u, v) \]

Where \(P_s(u, v)\) is the minimum weight or cost of traveling from point \(u\) to point \(v\). The weight of the path is the sum of the weights of its edges. Through this paper we assume that all the edge-weights are real values greater than zero.

When we want to find the shortest path between two particular points we have the single pair shortest path problem or point to point shortest path problem it is the most common in practice, if we want to find the shortest paths from one particular point to all the other points then we have the single source shortest path problem, there is also the single destination shortest path problem, and finally we have the all pairs shortest path problem when we want to find the shortest paths between all pairs of points.

2. Bellman-Ford Algorithm

2.1 The Algorithm

Bellman-Ford is a simple single source shortest path algorithm. The input to the algorithm is a weighted graph and a source point, for each point \(u\) the algorithm computes the shortest path from source \(P[u]\) and the predecessor of \(u\) in the path from source \(\pi[u]\).

The algorithm performs \(|V| - 1\) scans over the list of edges to check whether for each edge \((u, v)\) the following inequality is satisfied:

\[ P[u] + w(u, v) \geq P[v] \]

Where \(w(u, v)\) is the weight of edge \((u, v)\).

In the case of unsatisfactory the algorithm corrects \(P[v]\) to become \(P[u] + w(u, v)\), and \(\pi[v]\) to become \(u\).

The operation check inequality and correct the path in the case of unsatisfactory is known as the edge relaxation.

2.2 Redesign

Obviously Bellman-Ford has been designed for and only for the worst case, that is the case in which we have one shortest path with length \(|V| - 1\) edges and those edges are traversed in an order reversing the order in which they are encountered in the list.

Below is the algorithm in pseudo-code

Algorithm Bellman-Ford.

Input: Graph \(G(V; E; w)\) and source point \(s\)

\[
\begin{align*}
&\text{for } u \in V \text{ do } P[u] \leftarrow \infty, P[s] \leftarrow 0 \\
&\text{for } u \in V \text{ do } \pi[u] \leftarrow \text{null}, \pi[s] \leftarrow s \\
&\text{for } i = 1 \text{ to } |v| - 1 \text{ do} \\
&\hspace{1em} \text{for } (u, v) \in E \text{ do} \\
&\hspace{2em} \text{if } P[u] + w(u, v) \leq P[v] \\
&\hspace{3em} \text{then } P[v] \leftarrow P[u] + w(u, v) \text{ and } \pi[v] \leftarrow u
\end{align*}
\]

For graphs represented in adjacency lists the graph will be scanned point by point, the control statement of the inner loop can be replaced by the statement:

\[
\begin{align*}
&\text{for } u \in V \text{ do} \\
&\hspace{1em} \text{for } v \in N(u) \text{ do}
\end{align*}
\]

Where \(N(u)\) is the set of points adjacent to \(u\).
reachable from source satisfy the shortest path condition. The pseudo-code of the redesigned algorithm is below

**Algorithm Bellman-Ford 2008**

**Input:** Graph \( G(V;E;w) \) and source point \( s \)

for \( u \in V \) do \( P[u] \leftarrow \infty \); \( \pi[s] \leftarrow 0 \)

for \( u \in V \) do \( \pi[u] \leftarrow \text{null} \); \( \pi[s] \leftarrow s \)

boolean corrections

while (corrections) do

  corrections \( \leftarrow \) false

  for \( (u; v) \in E \) do

    if \( P[u] + w(u; v) < P[v] \)

      then \( P[v] \leftarrow P[u] + w(u; v) \)

      \( \pi[v] \leftarrow u \)

    corrections \( \leftarrow \) true

  \( \text{corrections} \leftarrow \text{false} \)

1: By simply labeling a point if its path is corrected and reset the label after scanning that point.

**Algorithm Bellman-Ford 2012.**

**Input:** Graph \( G(V;E;w) \) and source point \( s \)

for \( u \in V \) do \( P[u] \leftarrow \infty \); \( \pi[s] \leftarrow 0 \)

for \( u \in V \) do \( \pi[u] \leftarrow \text{null} \); \( \pi[s] \leftarrow s \)

for \( u \in V \) do \( \text{corrected}[u] \leftarrow \text{false} \), \( \text{corrected}[s] \leftarrow \text{true} \)

boolean corrections

while (corrections) do

  corrections \( \leftarrow \) false

  for \( u \in V \) do

    if \( \text{corrected}[u] \) then

      for \( v \in N(u) \) do

        if \( P[u] + w(u; v) \leq P[v] \)

          then \( P[v] \leftarrow P[u] + w(u; v) \)

          \( \pi[v] \leftarrow u \)

        corrected[v] \( \leftarrow \) true

      corrections \( \leftarrow \) true

    \( \text{corrected}[u] \leftarrow \text{false} \)

2: We can en-queue the labelled points and scan them in the order in which they were labelled instead of the order in which they are listed, points can be en-queued and de-queued more than once, the algorithm will terminate when the queue is empty.

This way we have an algorithm with a different loop invariant; it can be regarded as a redesign of the breadth first search for computing shortest paths in weighted graphs.

**Algorithm Breadth First Search 2010**

**Input:** Graph \( G(V;E;w) \) and source point \( s \)

for \( u \in V \) do \( P[u] \leftarrow \infty \); \( \pi[s] \leftarrow 0 \)

for \( u \in V \) do \( \pi[u] \leftarrow \text{null} \); \( \pi[s] \leftarrow s \)

for \( u \in V \) do \( \text{corrected}[u] \leftarrow \text{false} \), \( \text{corrected}[s] \leftarrow \text{true} \)

queue \( Q \)

en - queue(s)

while \( (Q \not\not\emptyset) \) do

  \( u \leftarrow \text{dequeue}(Q) \)

  for \( v \in N(u) \) do

    if \( P[u] + w(u; v) \leq P[v] \)

      then \( P[v] \leftarrow P[u] + w(u; v) \)

      \( \pi[v] \leftarrow u \)

      \( \text{corrected}[v] \leftarrow \text{true} \)

    if \( \text{corrected}[v] \)

      then \( \text{corrected}[v] \leftarrow \text{true} \)

    en - queue(v)

\( \text{corrected}[u] \leftarrow \text{false} \)

It worth mentioning that we shall have Dijkstra’s algorithm if we take a more restrictive step by designing the algorithm to scan the point with the minimum shortest path cost between the labelled points.

3. Point to Point Shortest Paths

3.1 Unidirectional Search

Single source shortest path algorithms can be applied to compute point to point shortest paths, one way is to run a single source algorithm after modifying it to terminate when the path to destination point is shorter than any path corrected recently.

The unidirectional point to point algorithm will be as shown below

**Algorithm Unidirectional Bellman-Ford 2012.**

**Input:** Graph \( G(V;E;w) \), source point \( s \), and destination point \( t \)

for \( u \in V \) do \( P[u] \leftarrow \infty \); \( \pi[s] \leftarrow 0 \)

for \( u \in V \) do \( \pi[u] \leftarrow \text{null} \); \( \pi[s] \leftarrow s \)

for \( u \in V \) do \( \text{corrected}[u] \leftarrow \text{false} \), \( \text{corrected}[s] \leftarrow \text{true} \)

\( \text{min path} \)

while \( (\text{min path} < P[t]) \) do

  \( \text{min path} \leftarrow \infty \)

  for \( u \in V \) do

    if \( \text{corrected}[u] \) then

      for \( v \in N(u) \) do

        if \( P[u] + w(u; v) < P[v] \)

          then \( P[v] \leftarrow P[u] + w(u; v) \)

          \( \pi[v] \leftarrow u \)

          \( \text{corrected}[v] \leftarrow \text{true} \)

    \( \text{corrected}[u] \leftarrow \text{false} \)

The algorithm maintains the same loop invariant as the other single source variants of Bellman-Ford, it is different in that it terminates after a number of scans equal to the number of sub-paths like those described in section 2.1 which are composing the shortest path from source to destination.
This modification we have just applied on Bellman-Ford 2012 can be applied to all the other algorithms in the previous sections.

3.2 Bidirectional search

To improve the performance of single source algorithms when they are applied to the point to point problem, two searches can be run simultaneously possibly in parallel at the source and the destination, this technique is known as the bidirectional search.

In this section we are going to modify Bellman-Ford 2012 so that it can be run in bidirectional search, again the modification can be applied to the other algorithms.

In bidirectional search we shall have two lists of shortest path costs, from source \( P_s[ ] \) and to destination \( P_t[ ] \), the algorithm will scan a point if one or both of its paths from source and to destination has been recently corrected, this means that we shall have two lists of labels \( corrected_s[ ] \) and \( corrected_t[ ] \). The algorithm will repeat graph scans and maintain the minimum summed value of \( P_s[ ] \) and \( P_t[ ] \) from the points corrected in both directions as a shortest path from source to destination \( P_{st} \), the algorithm will terminate when \( P_{st} \) is less than the sum of the minimum path corrected from source and the minimum path corrected to destination recently. The algorithm shown below is for directed graphs we assume that for every point \( u \) we have a list of outgoing edges \( N^+(u) \) and a list of incoming edges \( N(u) \).

**Algorithm** Bidirectional Bellman-Ford 2012.

**Input:** Graph \( G(V; E; w) \), source point \( s \), and destination point \( t \).

for \( u \in V \) do \( P_s[u] \leftarrow \infty \); \( P_t[s] \leftarrow 0 \)

for \( u \in V \) do \( corrected_s[u] \leftarrow false \), \( corrected_t[u] \leftarrow true \)

for \( u \in V \) do \( _u \leftarrow null \); \( _s \leftarrow s \)

for \( u \in V \) do \( P_t[u] \leftarrow \infty \); \( P_t[t] \leftarrow 0 \)

for \( u \in V \) do \( corrected_t[u] \leftarrow false \), \( corrected_t[u] \leftarrow true \)

\( min_s \)

\( min_t \)

\( P_{st} \leftarrow \infty \)

while \( (min_s + min_t < P_{st}) \) do

\( min_s \leftarrow \infty \)

\( min_t \leftarrow \infty \)

for \( u \in V \) do

if \( corrected_t[u] \) then

for \( v \in N^+(u) \) do

if \( P_s[u] + w(u; v) < P_t[v] \)
then \( P_t[v] \leftarrow P_t[u] + w(u; v) \)

\( \pi[v] \leftarrow u \)

\( corrected_t[v] \leftarrow true \)

if \( P_s[v] < min_s \)
then \( min_s \leftarrow P_t[v] \)
if \( P_s[v] + P_t[v] < P_{st} \)
then \( P_{st} \leftarrow P_s[v] + P_t[v] \)

References