**Overview of the Algorithms**

A network consisting of a set of nodes N and a set of arcs A is represented as a

directed graph G = (N, A), with

* n = |N| denoting the number of nodes
* m = |A| denoting the number of directed arcs

Each arc (i, j) is represented as an ordered pair of nodes traveling from node i to node j.

Each arc (i, j) has an associated numerical value,

dij, representing the distance or cost incurred by traversing the arc.

----

In this article,

it is assumed that it is possible to travel between a pair of connected nodes i and j in both directions, and hence

two directed arcs (i, j) and (j, i) are distinguished for each pair of connected nodes i and j.

**The one-to-all shortest paths are normally represented as a directed out-tree rooted at the source nodes.**

**This directed tree is referred to as a shortest path tree.**

Unless otherwise noted, a **shortest path tree** and **one-to-all shortest paths** are used interchangeably in this article.

----

Central to all shortest path algorithms discussed in this article is the **labeling Method**.

Both the label-setting algorithms and label-correcting algorithms iteratively employ the labeling method to construct a shortest path tree.

**In the labeling method, an out-tree from the source node “s” is constructed and improved until no further improvements can be made.**

At termination, the shortest path from “s” to node “i” is represented by a unique path from “s” to “i” on the out-tree.

While constructing and improving the out-tree, three pieces of information are maintained for each node “i” in the labeling method:

* a distance label d(i)
* a parent node p(i)
* a node status S(i)∈{unreached, labeled, scanned}

**(Distance Label)**

As the labeling method progresses, the **distance label d(i)** is used to record the upper bound of the s-to-i path distance in the out-tree.

Upon termination of the algorithm, the distance label d(i) represents the shortest path distance from node “s” to node “i”.

**(Parent Node)**

The **parent node** **p(i)** records the node that immediately precedes node “i” in the out-tree.

**(Node Status)**

A node is said to be **unreached** if its distance label has a value of infinite (sufficiently large).

A node is said to be **labeled** if its distance label that has been updated at least once, that is, its distance label is different from infinite.

A node is said to be **permanently labeled** if its distance label represents the final and optimal shortest pathfrom the source node.

A node is considered **scanned** if it has undergone a scanning operation(explained below) and its distance label has subsequently not been updated.

----

The labeling method begins by setting

* d(i) = ∞,
* p(i) =∅
* S(i) = unreached

for every node “i”.

To initialize the method, information associated with a source node “s” are set as

* d(s) = 0
* S(s) = labeled

The method progresses by applying a scanning operation to labeled nodes commencing from “s”

* until the destination node is reached and permanently labeled (DIKBA)
* until all nodes have been scanned and permanently labeled (TWO-Q)

The method terminates with a shortest path tree over either

a subset of nodes (DIKBA)

or

the entire set of nodes (TWO-Q).

In scanning node i, an attempt is made to lower the distance labels for any node j such that (i, j) ∈ A. If the distance label for node j can be lowered, the out-tree is adjusted by changing node j’s parent node label by setting p(j) = i and node j is considered labeled. Formally, the scanning operation for node i can be defined as follows:

for all (i, j) ∈ A do

begin

if d(i) + dij < d(j) then

begin

d(j) = d(i) + dij

p(j) = i

S(j) = labeled

end

end

set S(i) = scanned

Label-setting and label-correcting algorithms are based on the labeling method and

employ the scanning operation as described.

**The two algorithms differ primarily in the data structures used for**

* **managing the set of labeled nodes**
* **selecting nodes for scanning**

Brief overviews of the data structures for DIKBA and TWO-Q are given below.

A more detailed description of the data structures and procedures related to these two algorithms can be found in Zhan (1997).

----

In the original Dijkstra algorithm, the node selected for scanning is a labeled node with the minimum distance label.

A result of this selection criteria is that, once a node is scanned, it becomes permanently labeled and is never again selected for scanning.

In Dijkstra’s original implementation of the algorithm, the set of labeled nodes is managed as an unordered list.

This is clearly a bottleneck operation since all labeled nodes must be checked during an iteration in order to select the node with the minimum distance label.

A natural enhancement of the original Dijkstra algorithm is to maintain the labeled nodes in a data structure such that the nodes are either exactly or approximately sorted according to distance label.

The bucket data structure is one such structure.

----

In the approximate bucket implementation of the Dijkstra algorithm (DIKBA), a bucket “i” contains those labeled nodes whose distance labels are within the range of [i∗β, (i+1)∗β−1], where β is a chosen constant.

The bucket structure is considered approximate since the values of the distance labels in a bucket are not exactly the same, but are within a certain range.

Nodes within each bucket are maintained in a FIFO ordered list.

Algorithm DIKBA requires a maximum of (C/β) + 1 buckets and has worst case complexity of O(mβ + n(β + C/β)), where C is the length of the longest arc.

It is important to note that, in the DIKBA implementation, a node can be scanned more than once, but never more than β times.

For example, a node can be scanned out of its current bucket and then become labeled during the scanning operation of another node in the same bucket.

When all nodes within a current bucket have been completely scanned out, their distance labels become permanent.

For the tested implementation of DIKBA, the bucket width parameter β was set in the same fashion as in Cherkassky et al. (1996), i.e. the number of buckets (C/β) was set up to, but not exceeding, 211.

----

Label-correcting algorithms manage the set of labeled nodes in a very different fashion.

In TWO-Q, the labeled nodes are stored in one of two ordered lists (or queues), Q1 and Q2.

By definition, a node becomes labeled if its distance label is changed during a scanning operation.

In the TWO-Q algorithm, a node that becomes labeled is treated differently depending on whether or not it has ever been scanned before.

If it has been scanned before, it is assigned to the tail of Q1.

If it has never been scanned before, it is assigned to the tail of Q2. The node selected for scanning is always the head node of Q1 or, in the case that Q1 is empty, the head of Q2.

When a node is scanned, it is removed from the set of labeled nodes.

However, if the node’s distance label is later changed by a scanning operation, the node is returned to the set of labeled nodes.

For this reason, the nodes are not considered permanently labeled until the entire set of labeled nodes (consisting of Q1 and Q2) is empty.

The worst case complexity of the TWO-Q algorithm is given as O(n2m).

----

The differences in the running times associated with DIKBA and TWO-Q depend on the number of scans and the time required to select nodes for scanning.

Dijkstra-based algorithms tend to have either exactly one or slightly greater than one (as in the general case of DIKBA) scan per permanently labeled node.

The TWO-Q implementations can have considerably more scans per node, however, the effort required to select a node for scanning is minimal.

In the case of DIKBA, however, buckets must be checked sequentially until one is found that contains a labeled node eligible for scanning.

Having to check an excessive number of empty buckets can be costly in terms of wasted effort.

----