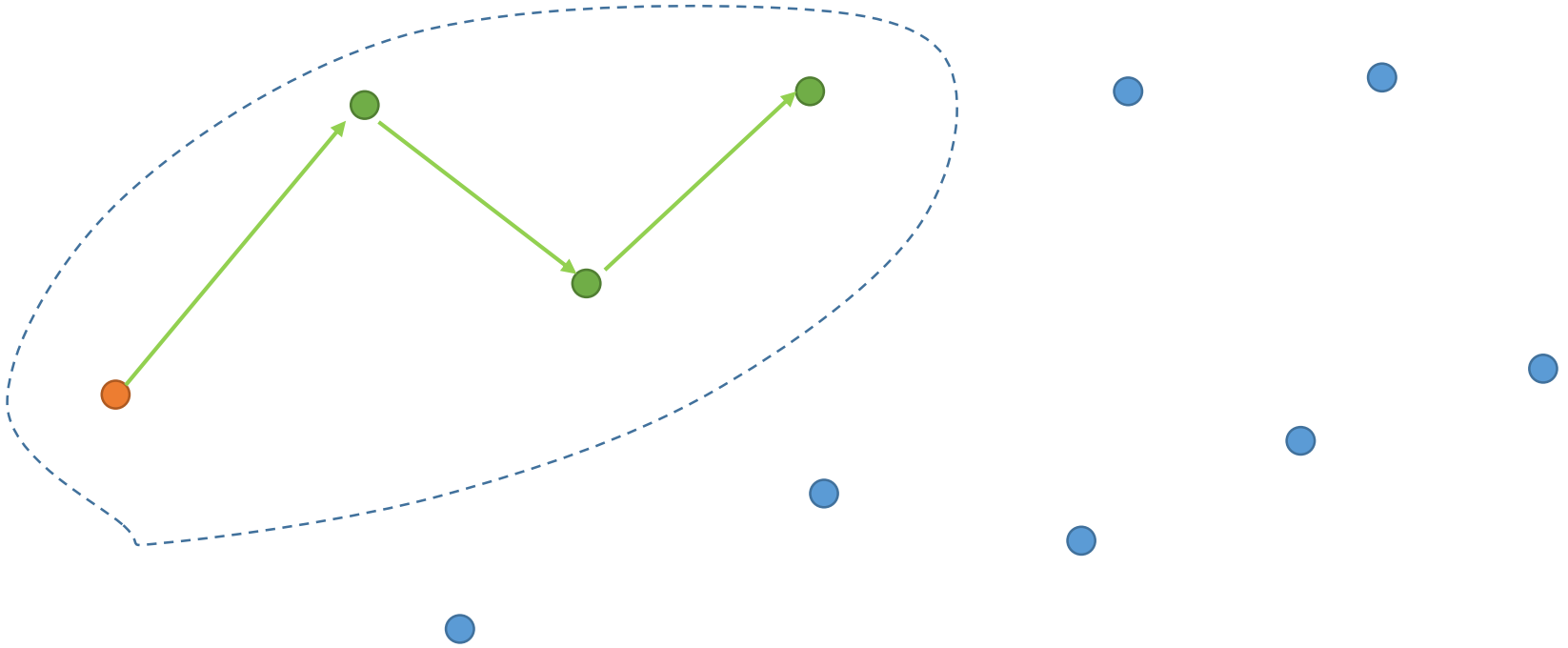
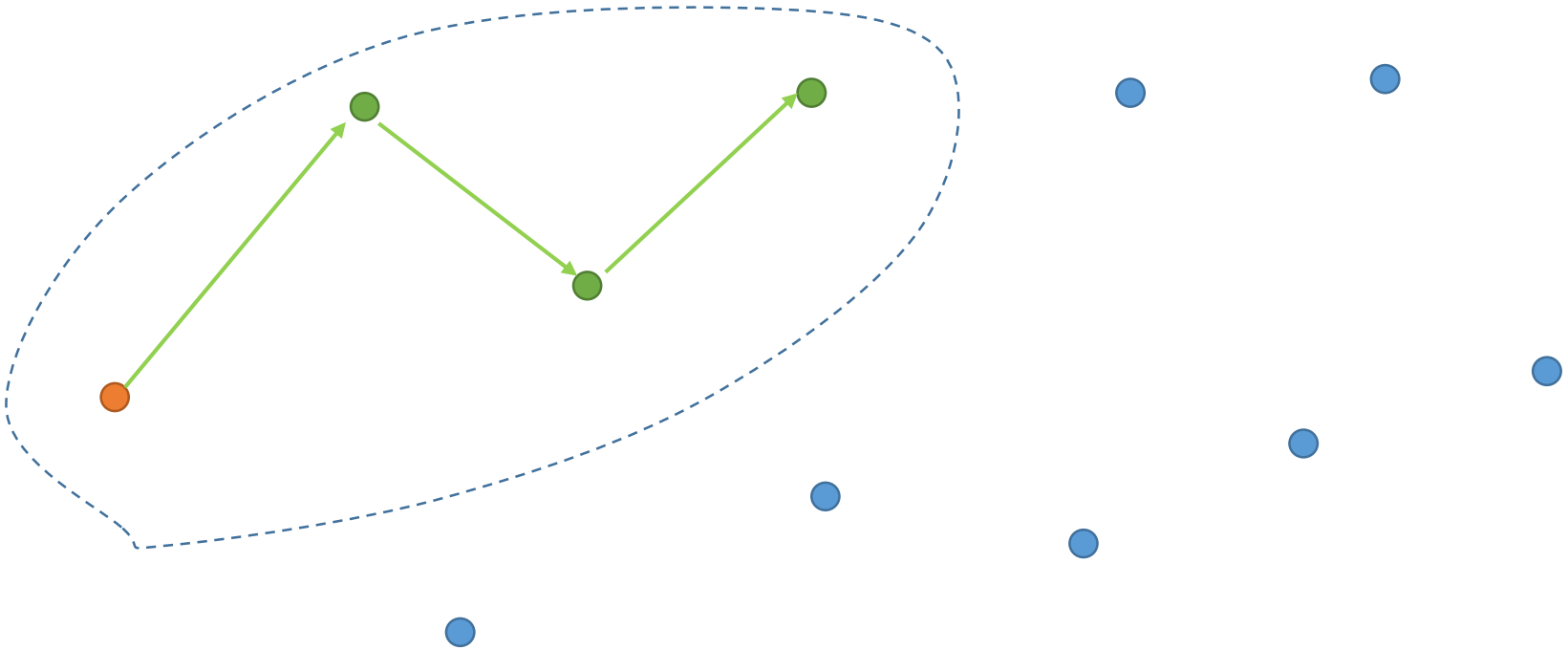
# Continuum Between Prim MST and Dijkstra SSSP

Shiliang Ma



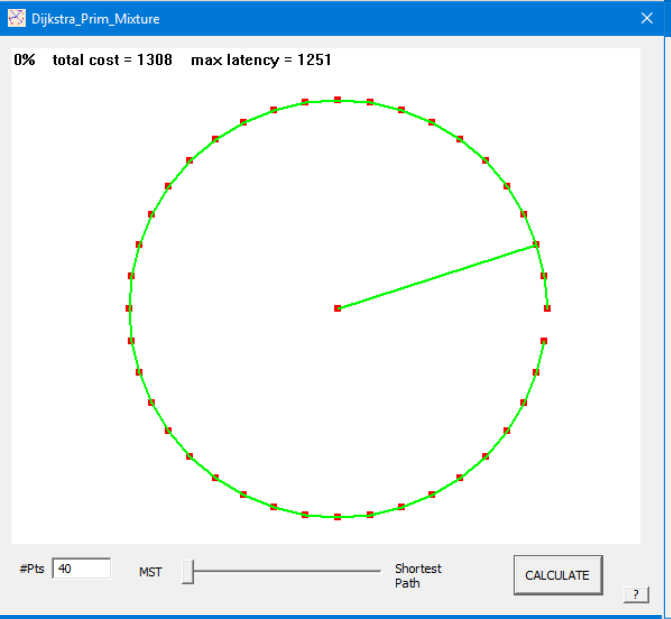
Both Prim and Dijkstra use greedy algorithm
Prim finds the next point with the shortest distance from the added region
Dijkstra find the next point with the shortest distance from the source, which is the distance from the region plus its connected point's distance to the source
Or we can say
Prim: min(distance to region)
Dijkstra: min(distance to region + region point's distance to source)
Prim minimizes total cost
Dijkstra minimizes maximum latency

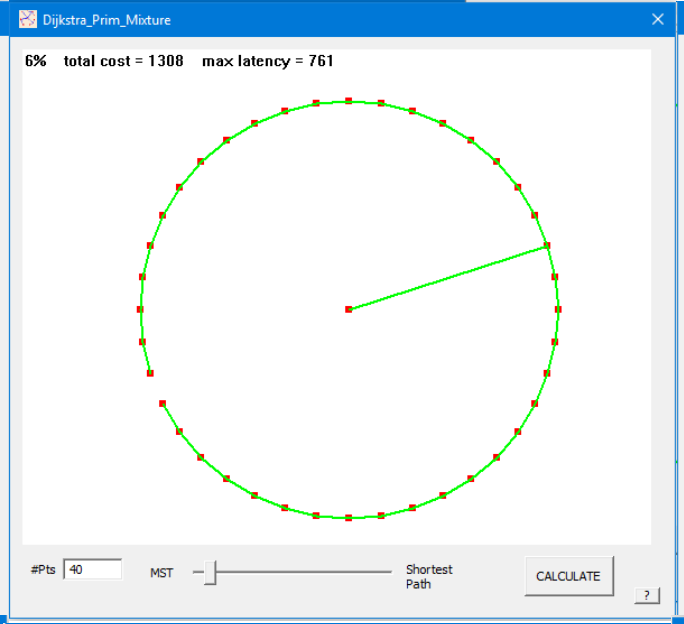


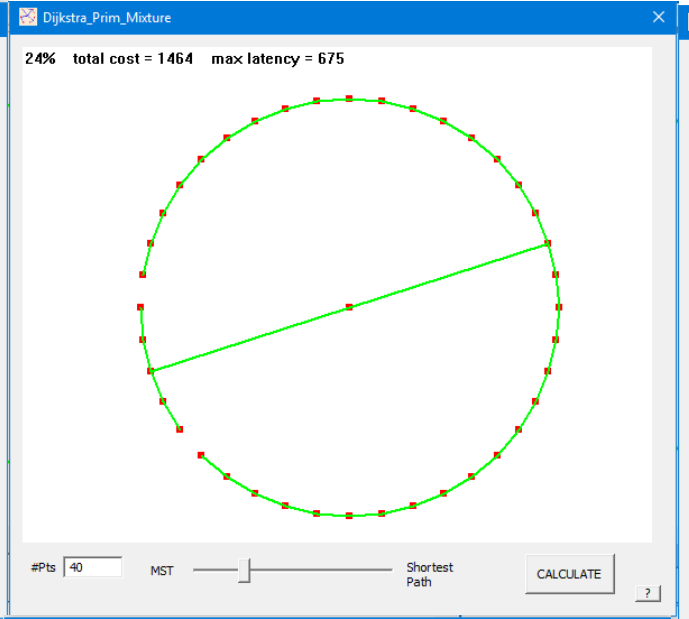
There is a way to make trade off between total cost and maximum latency, that is a mixture of Prim and Dijkstra The key is to minimize
(1-$\epsilon$)(distance to region)+($\epsilon$)(distance to region + region point's distance to source)
Where $\epsilon \in [0,1]$, when $\epsilon = 0$, it returns PRIM MST, while when $\epsilon = 1$, it returns DIJKSTRA SSSP

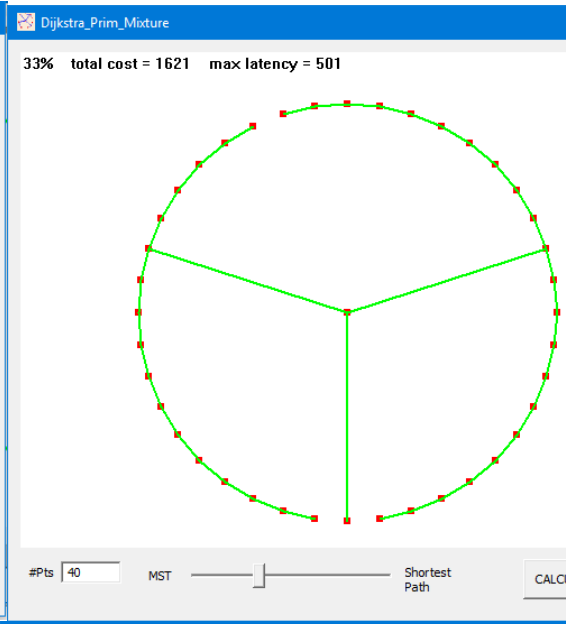
```
int index_in, index_out;
int LeastDistance;
int currentDistance;
int medRes;
for (int iteration = 1; iteration < sz; iteration++)
{
    LeastDistance = INT_MAX;
    for (int in = 0; in < sz; in++)
    {
        if (m_LinkedPts[in].PrevPointID != NOTCONNECTED)
        {
            for (int out = 0; out < sz; out++)
            {
                if (m_LinkedPts[out].PrevPointID == NOTCONNECTED)
                {
                    medRes = CalcDistance(m_Points[in], m_Points[out]);
                    currentDistance = (100-m_intWeight)*medRes + m_intWeight*(m_LinkedPts[in].TotalDistanceToSource + medRes);
                    if (currentDistance <= LeastDistance)
                    {
                        LeastDistance = currentDistance;
                        index_in = in;
                        index_out = out;
                    }
                }
            }
        }
    }
    m_LinkedPts[index_out].PrevPointID = index_in;
    m_LinkedPts[index_out].TotalDistanceToSource = m_LinkedPts[index_in].TotalDistanceToSource + CalcDistance(m_Points[index_in], m_Points[index_out]);
}
```

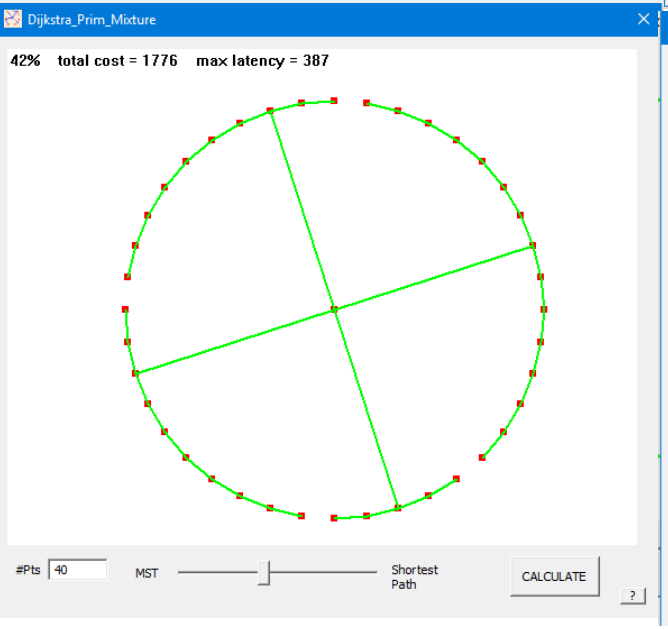
Above is the key code, where 3 for loops are used. Because of the introduction of $\epsilon$, the greedy algorithm no longer applies, therefore, one extra for loop is needed to check every possible value. The total run time is increased to $O(V^3)$

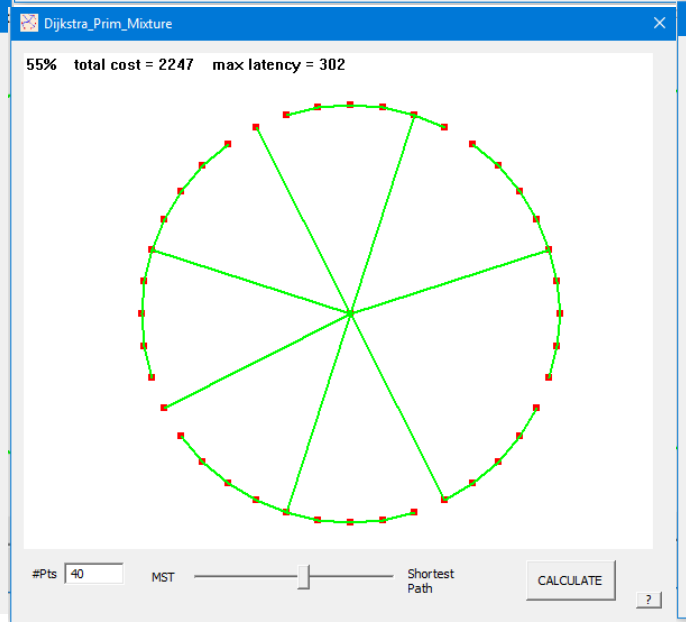


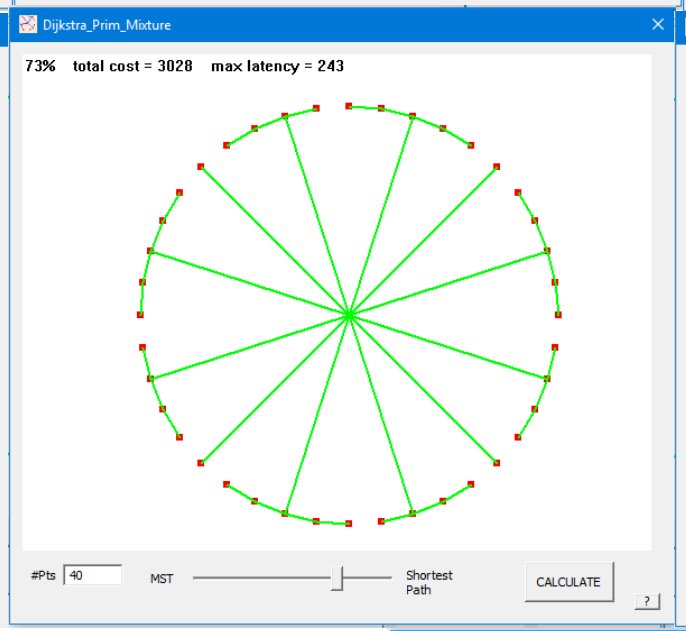



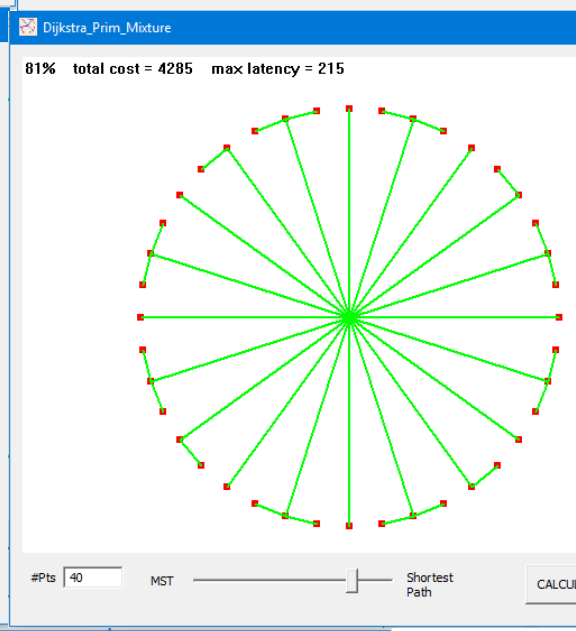



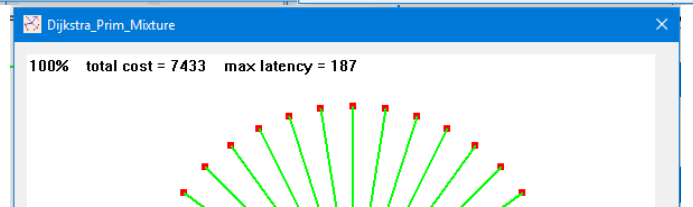




It can be seen that as the graph changes from MST to SSSP, the total cost increases while max latency decreases