Δ-Stepping SSSP: A topology-driven processing (e.g Bellman Ford SSSP) is suitable for low-diameter graphs. Road networks defy both this since they have high-diameter. Due to high diameter, the number of iterations can be very high (e.g., California road network has a diameter of 849 requiring at least those many iterations). A generalization of these two extremes is Δ-Stepping, wherein vertices are ordered based on distance values leading to quicker fixpoint computation, but also some redundant computation is allowed to improve concurrency.

The Δ-Stepping SSSP performs well on a multi-core CPU, especially on road networks. In the Δ-Stepping SSSP Algorithm, vertices are ordered using a set of *worklists* called *buckets* representing priority ranges of Δ, where Δ is a positive value. The *bucket* $B[i]$ will have vertices whose current *distance* value is given by $(i-1) \times \Delta \leq distance < i \times \Delta$. The *buckets* are processed in an increasing order of index value *i* and a bucket $B[i]$ is processed only after bucket $B[i-1]$ is processed. Algorithm 0.1 shows the Δ-Stepping SSSP algorithm.

The function *Relax* takes as argument a vertex *v* and an integer value *x*. If the current *distance* of *v* is greater than *x*, the vertex *v* is removed from the current bucket *B[distance(v)÷Δ]* and it is added to the bucket *B[x ÷Δ]*. Then the *distance* of vertex *v* is reduced to *x* (line 5).

The *SSSP* algorithm works in the following way. Initially for each vertex *v* in the graph, two sets *heavy* and *light* are computed, where

heavy(v) = (∀ (v, w) ∈ E) ⋀ (*weight(v, w)* >Δ)

light(v) = (∀ (v,w) ∈ E) ⋀ (*weight(v, w)* ≤ Δ)

Then the *distance* of all the vertices is made ∞ (Lines 9–13). The core of the algorithm starts by relaxing the *distance* value of source vertex *s* in line 14 with a *distance* value of zero. This adds the source vertex to *bucket* zero (line **??**). Then the algorithm enters the while loop in lines 17 to 29, processing *buckets* in an increasing order of index value *i*, starting from zero.

An important feature of the algorithm is that, once the processing of *bucket* $B[i]$ is over, no more elements will be added to the bucket $B[i]$, when the buckets are processed with increasing values of index *i*. A *bucket* $B[i]$ is processed in the while loop (lines 19 to 24). At first, for all `vertex` v in *bucket* $B[i]$, all edges $v \longrightarrow w$ ∈ *light*(v) are considered and the pair (w, *distance*(v) + *weight*(v, w)) is added to the `Set` *Req*. This is followed by the `Set` *S* added with all the elements in $B[i]$ (Line 21) and $B[i]$ made empty (Line 22). Then *Relax()* function is called for all elements in `Set` *Req*. This adds new elements to multiple buckets. It can add a vertex *w* to a bucker B[k] where k ≥ i. The bucket to which the vertex *w* is added is (dist[v]+weight(v,w))÷Δ.

The vertex *w* is added to the bucket B[i] if *distance*[w] ≥ $i \times \Delta$ and there can be element $(w, x) \in Req$ where $x < i \times \Delta$. Here $x = distance(v) + weight(v, w)$ for an edge $v \longrightarrow w$. Once the *bucket* $B[i]$ becomes empty after a few iterations, the program exits the while loop (Lines 19 to 24). Now all edges $v \longrightarrow w \in$ *heavy*(v) are considered and the pair (w, *distance*(v) + *weight*(v, w)) is added to the `Set` *Req*(Line 26). The edges in `Set` heavy have weight > Δ and this makes $\forall(v, x) \in Req\ x > i \times \Delta$. So new elements will be added to bucket $B[j]$ when the `Set` *Req* is relaxed where $j > i$ (Line 27). Now value of *i* is incremented by one (Line 28) and algorithm starts

Algorithm 0.1: Δ-Stepping SSSP algorithm

```
Relax( vertex v, int x) {
    if( (x <distance(v) ) ){
        Bucket B[ distance(v)÷ Δ] = B[distance(v)÷ Δ] \ v;
        Bucket B[ x÷ Δ] = B[x÷ Δ] ∪ v;
        distance(v) = x;
    }
}
SSSP (Graph G) {
    foreach( each v in V ) in parallel {
        Set heavy(v) = { (v, w) ∈ E: weight(v, w) >Δ }
        Set light (v) = { (v, w) ∈ E: weight(v, w) <= Δ }
        distance (v) = INF; // Unreached
    }
    relax(s, 0); // bucket zero will have source s.
    i = 0;
    // Source vertex at distance 0
    while( NOT isEmpty(B) ){
        Bucket S = ϕ;
        while( B[i] ≠ ϕ ){
            Set Req = {(w, distance(v) + weight(v, w)) : v ∈ B [i] ⋀ (v, w) ∈ light(v)};
            S = S ∪ B[i];
            B[i] = ϕ;
            foreach ((v, x) ∈ Req) in parallel Relax(v, x);
        }
        //done with B[i]. add heavy weight edge for relaxation
        Req = { (w, distance(v) + weight(v, w)): v ∈ S ⋀ (v, w) ∈ heavy(v) }
        foreach((v, x) ∈ Req)in parallel relax(v, x);
        i = i + 1
    }
}
```

processing *bucket* $B[i+1]$. Algorithm terminates when all the *buckets* $B[i]$, i≥0 are empty. The performance of the algorithm depends on the input graph and the value of the parameter Δ, which is a positive value. For a Graph $G(V, E)$ with random edge weights, maximum node degree d (0<d≤1), the sequential Δ-stepping algorithm has a time complexity of O($|V|$ + $|E|$ + d × P), where P is the maximum SSSP distance of the graph. So, this algorithm has running time which is linear in $|V|$ and $|E|$.