Dijkstra-DBSCAN: Routing to Connect Your Big Data

Yang Zhang, Prof. Lee. D. Han
The University of Tennessee, Knoxville

Scope

A fast, accurate, flexible big data clustering algorithm for various decision making purposes, such as :

- Incident hot spot detection
- Lane safety management
- Location selection for resource deployment
- Investment location selection, etc.

Dijkstra-DBSCAN is:

- **Fast**: low computational cost (O(n)); easy to parallel.
- **Accurate**: based on real road network.
- **Flexible**: supports any surface street configuration: freeway, freeway junctions, interchanges, roundabouts, and other complicated combinations.

Big Data Challenge

Big Data Explosion VS Slow and Un-parallable Algorithms

- Growing availability of real time vehicle positional data from various sources, such as GPS, smart phones, RFID devices, blue tooth, is creating new opportunities of using spatial clustering techniques to study the traffic flows in both local and inter-regional scale.
- Traditional spatial algorithms, such as Kernel Density Estimation, K-means, hierarchical methods are at least $O(n^2)$ complexity, and hard to parallelize to utilize distributed computing resources.
- Moreover, most previous methods ignore the underlying network connection of traffic data, making the results inaccurate for decision making purposes.

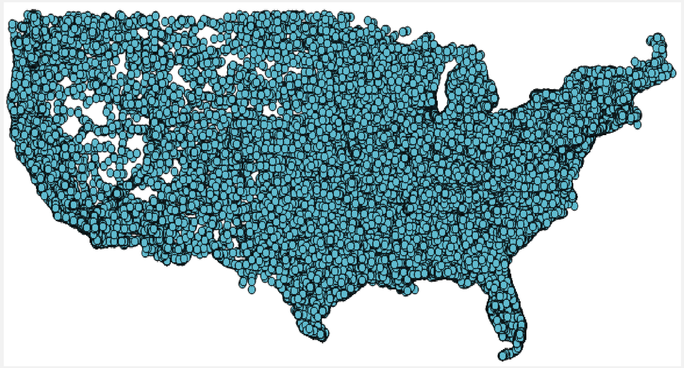

Figure 1. Massive location based data are constantly emerging over large space scales

Figure 2. Road network means accuracy. (Left) Three points are identified as a cluster by a general spatial clustering algorithm. (Right) But actually they are not closely connected.

Dijkstra-DBSCAN

```
Modified_Dijkstra(G, s, ε ){
    d[s] ← 0
    Q = {s}
    neighbor_list = {}
    while Q is not empty:
        u ←Q[0]
        Q←Q- {u}
        for each neighbor v of u:
            if d[u]+e(u,v) < d[v]:
                d[v] = d[u]+e(u,v)
                Q←Q ⋃ {v}
        sort Q in ascending order
        if d[Q[0]] > ε :
            return neighor_list
        else:
            neighbor_list←neighbor_list ⋃ Q[0]
        mark Q[0] as visited
}
```

```
Dijkstra_DBSCAN (D, ε , min pts ){
    id ← 0
    for each point p in dataset D:
        N(p) ← Modified_Dijkstra(G, p , ε )
        if |N(p)| ≥ min pts :
            mark p as core
    for each unvisited core point o :
            mark o as visited
            C_id = {o}
            id++
            ExpandCluster(o)
}
ExpandCluster(o){
    for each unvisited p' in N(o) :
        mark p' as visited
        C_id = C_id ⋃ {p'}
        if p' is core
            ExpandCluster ( p')
}
```

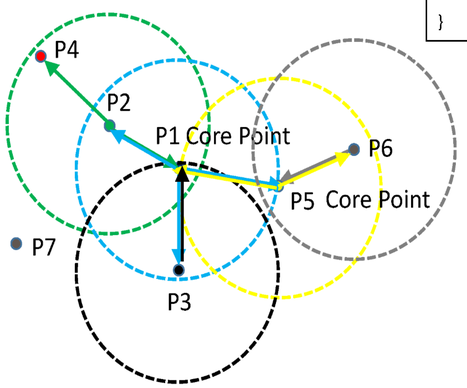

Figure 3. Cluster Expansion

Figure 4. Pseudo code of Dijkstra-DBSCAN and Modified Dijkstra

Case Study: 5-Year Fatalities of the entire U.S.

The road network contains **27,145,945** links and **17,464,790** nodes. **152,089** traffic fatalities of 5 years (2009 to 2013) provided by Fatality Analysis Reporting System (FARS).

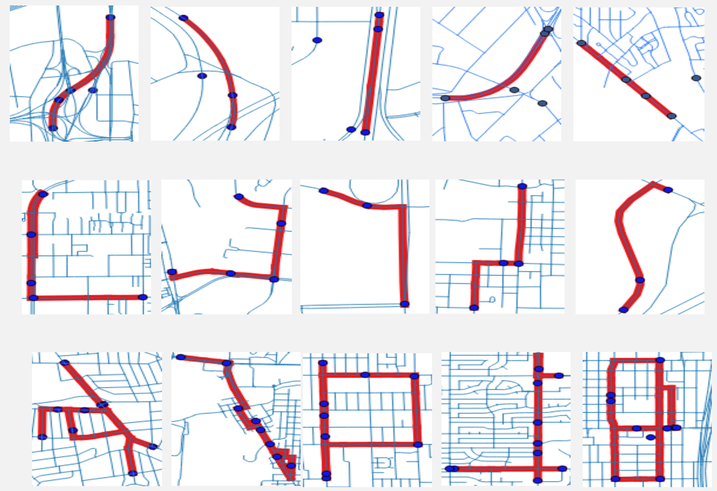

Figure 5. Sample Clusters

Paralleled Dijkstra-DBSCAN

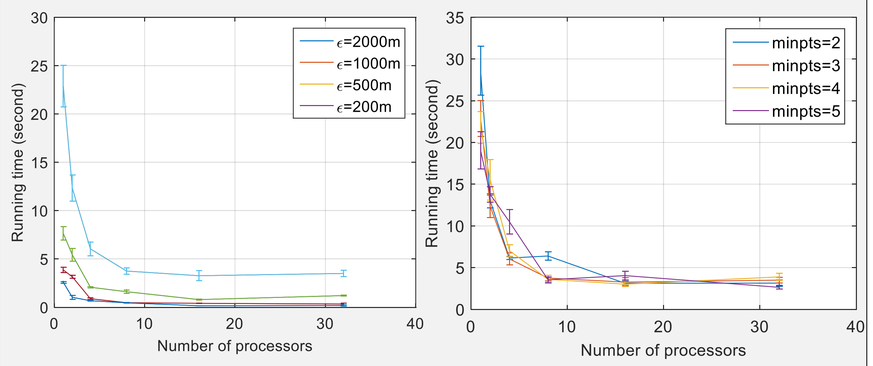

Figure 6. Benchmarks of Parallel Dijkstra-DBSCAN. (a) with different epsilon values; (b) with different minpts values.

Ongoing Work: Fuzzy Dijkstra-DBSCAN for cluster ranking

- We are developing the fuzzy version of Dijkstra-DBSCAN, which will provide an quantitative measure for ranking clusters based on their densities. This problem becomes essential when investment budget is so limited that only a few locations will be selected to be selected.