Union-Find Parallel DBSCAN

Daniel Woldegiorgis
Advisor: Professor Tupper

Introduction

Given a set of objects, clustering is basically concerned with getting meaningful subclasses, known as clusters, such that we minimize intra differences and maximize inter differences of the subclasses. Most of the clustering algorithms that are in use today can be categorized into one of four types: partitioning-based, hierarchy-based, grid-based, and density-based. Among the multiple algorithms in each of these categories, DBSCAN has gained popularity in spatial clustering because of its ability to deal with arbitrarily shaped clusters with noise such as the synthetic aggregate dataset below.

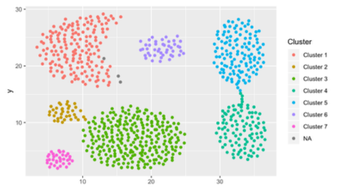


Fig 1. DBSCAN clustering of synthetic aggregate data

DBSCAN Algorithm

The central idea of the DBSCAN algorithm is that for each point in a cluster, the neighborhood within a given radius (*eps*) has to contain at least a minimum number of points (*minPts*).

Algorithm DBSCAN
Input: A collection of points X, *eps* distance, minpts.
Output: A vector of cluster assignments.

procedure DBSCAN($X, eps, minpts$)
  for each unvisited point $x \in X$ **do**
    Mark x as visited
    N = NEIGHBORS(x,eps)
    if | N | < minpts **then**
      mark x as noise
    else
      C.assign(x)
      for each point y ∈ N **do**
        $N = N - y$
        if y is not visited **then**
          mark y as visited
          Y = NEIGHBORS(y,eps)
          if | Y | ≥ minpts **then**
            N = N ∪ Y
        if y is not yet a member of any cluster **then**
          $C = C \cup y$

Parameter Selection Heuristics

One of the challenges involved when using DBSCAN is the high dependence of the quality of the clustering results on the choice of its input parameters, *eps* and *minPts*.

- For *minPts*: Using domain knowledge to determine how dense a certain cluster should be.
- For *eps*: Creating a KNN(k nearest neighbors) distance plot to find the elbow in the graph which will serve as our epsilon value, given a choice of k *minPts*.

Note: The heuristics mentioned here are not optimal for all cases therefore using cluster validation techniques to confirm the choice of parameters may be necessary. If one seeks automatic parameter selection, there is a wonderful method introduced in (Karami & Johansson, 2014) that uses differential evolution.

Methods

The parallel DBSCAN algorithm implementation was done in C++ using the OpenMP framework.

What is OpenMP?
OpenMP is a parallel programming framework used for multithreaded parallel processing. Most of the constructs in OpenMP are compiler directives like those below that apply to a structured block.

- #pragma omp construct[clause [clause]]
- #pragma omp parallel num_threads(8)

Case for Parallelization
Despite its many advantages DBSCAN becomes less and less applicable as the input size increases because of the increased number of neighborhood queries that result during each pass of the algorithm. To remedy this the KD-data structure which allows for efficient neighborhood searches was used when performing the epsilon neighborhood lookup of a given point. This transition made the overall time complexity of a neighborhood query from O(n) to O(log n) in the worst case.

Challenges in Parallelization

- Breaking the Inherent data access order
- Possibility of race conditions when merging the trees at the end of the Parallel DBSCAN.

Results

Below are the results of performing spatial clustering on the JSM 2018 dataset for 07/01/14 using the classical DBSCAN as well as the OpenMP parallel DBSCAN algorithm. We observed that the cluster assignments obtained from the two algorithms are identical. The output obtained looks as shown below in Fig 2. The clustering assignment was done based on their geographical proximity in terms of longitudes and latitudes as well as the proximity of their Humidity, Sea level pressure and Temperature values on a given day.

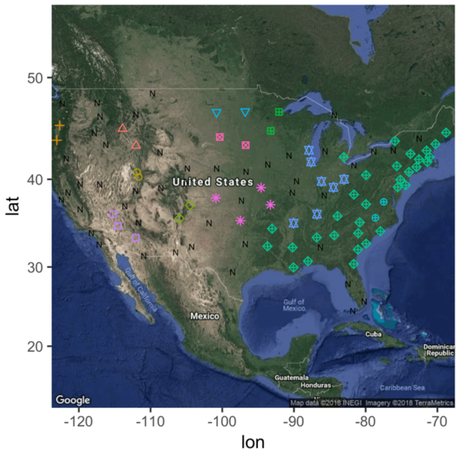


Fig 2. Spatial Clustering using Parallel DBSCAN

Algorithm

Algorithm Parallel DBSCAN
Input: A collection of points X, *eps* distance, minpts.
Output: A vector of cluster assignments.

procedure PARALLELDBSCAN($X, eps, minpts$)
  for t = 1 to p in *parallel* **do**
    for each point $x \in X_t$ **do**
      $p(x) \leftarrow x$
      $Y_t \leftarrow \emptyset$
    for each point $x \in X_t$ **do**
      N = NEIGHBORS(x, eps)
      if | N | ≥ *minpts* then
        mark x as a core point
        for each point $y \in N$
          if $y \in X_t$ **then**
            if y is a core point **then**
              UNION(x,y)
            else if $y \notin$ any **then**
              mark y as member of a cluster
              UNION(x,y)
          else
            $Y_t \leftarrow Y_t \cup$ (x,y)
  for t = 1 to p in *parallel* **do**
    for each (x,y) ∈ Y_t **do**
      if y is a core point **then**
        UNIONUSINGLOCK(x,y)
      else if$y \notin$ any cluster *then*
        mark y as member of a cluster
        UNIONUSINGLOCK(x,y)

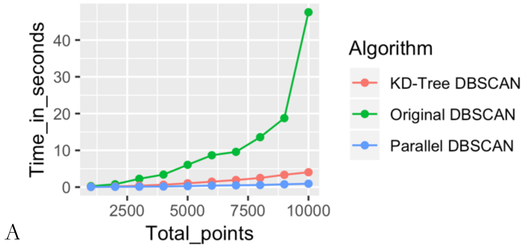
(Patawry et al., 2012) proposed using a two-step approach with initial local computation of clusters, where we split the dataset and perform the clustering separately for each partition, followed by the merging stage where points initially belonging to different partitions are checked to see if they belong to the same clusters. Pseudo-codes for the local computation, and merging stages can be seen at the left and bottom, respectively.

Algorithm Merging trees
Input: A collection of points X, *eps* distance, minpts.
Output: A vector of cluster assignments.

procedure UNIONUSINGLOCK(x, y)
  while $p(x) \neq p(y)$ **do**
    if $p(x) < p(y)$ **then**
      if $x = p(x)$ **then**
        Lock($p(x)$)
        if $x = p(x)$ **then**
          $p(x) \leftarrow p(y)$
        UNLOCK($p(x)$)
      $x = p(x)$
    else
      if $y = p(y)$ **then**
        LOCK($p(y)$)
        if $y = p(y)$**then**
          $p(y) \leftarrow p(x)$
        UNLOCK(p(y))
      $y = p(y)$

Conclusion

The performance of the Original DBSCAN, KD-tree based DBSCAN, and parallel KD-tree based DBSCAN algorithms with increase in the number of points being clustered is shown below. The algorithms were run on a randomly generated synthetic dataset using similar input parameters for each.



A both the original and KD-tree based DBSCAN algorithm implementations when the objects to be clustered continues to increase.

Future Work

- Implementing the parallel DBSCAN algorithm using SPARK for more efficient data partitioning at step 2 of the Parallel DBSCAN algorithm.
- Parallelizing ST-DBSCAN (SpatioTemporal DBSCAN) exploiting the same concurrent structure that is common amongst density based clustering algorithms.
- Coming up with better heuristics for the selection of the input parameters: *eps* and *minpts*.

References

- Patwary, Mostofa Ali, Palsetia, Diana, Agrawal, Ankit, Liao, Wei-keng, Manne, Fredrik & Choudhary, Alok (2012). A new scalable parallel DBSCAN algorithm using the disjoint-set data structure. Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis.
- Ester, Martin, Kriegel, Hans-Peter, Sander, Jörg, Xu, Xiaowei & others (1996). A density-based algorithm for discovering clusters in large spatial databases with noise.
- Karami, Amin & Johansson, Ronnie (2014). Choosing DBSCAN parameters automatically using differential evolution. International Journal of Computer Applications
- OpenMP [Online]. Available: http://openmp.org/.
- JSM 2018 Dataset[Online].Available:http://community.amstat.org/stat-computing/data-expo/data-expo-2018
- Aggregate Data[Online].Available: http://cs.joensuu.fi/sipu/datasets/