OR4NN User’s Guide

V0.1

Author: Sheng Zhou (Sheng.Zhou@os.uk)

# 1 Introduction

This guide provides a brief description on how to use the OR4NN prototype implantation to perform efficient large-scale K-NN queries in Databricks.

To perform a K-NN query such as “find the 3-nearest schools for every GB address”, three datasets will be required:

* Object set (e.g., Schools) with an ID and a geometry column.
* Query set (e.g., addresses) with an ID and a geometry column.
* Partition: Current prototype only supports partition grids generates by the prototype
  + The prototype provides several read-to-use grid partitions at 1km, 2km, 5km and 10km.
  + Utilities to generate other partition grids will also be provided.
  + A partition dataframe has an ID and a boundary geometry column.

# 2 Components

The current OR4NN prototype implementation contains a set of DataBricks Scala notebooks and a set of supporting JAR files

The Repo for the OR4NN prototype is at:

<https://ordnancesurvey@dev.azure.com/ordnancesurvey/Data%20Science%20and%20Analytics/_git/or4nn>

Scala Notebooks:

* OR4NN-PreProcess\_v0\_1: library functions for pre-processing object dataset
  + OR4NN-PreProcess-pt-to-pt\_v0\_1: pre-process example for point-to-point query
  + OR4NN-PreProcess-Plg-to-Plg\_v0\_1: pre-process example for polygon-to-polygon query
* OR4NN-Query\_v0\_1: library functions for making *k*-NN queries on a pre-processed dataset
  + OR4NN-Query-Test-Point-to-Point: point-to-point KNN query example
  + OR4NN-Query-Test-Plg-to-Plg: polygon-to-polygon KNN query example
* Radig2\_v0\_1: for Radig indexing to match queries to partitions.

JAR files:

* LargeKNN.jar (including JTS1.19): OR4NN library
* OS\_MultiMap\_1\_0.jar: MultiMap data structures
* radig2.jar: RADIG indexing

dependency:

* Sedona 1.4

# 3 Installation

In addition to Sedona, the three JAR files should be installed as libraries on to the cluster used for the query process.

The JAR files are in the repo, also available from the following paths DSAA DBFS:

OR4NN main jar: dbfs:/FileStore/jars/largeknn/LargeKNN\_v0\_1.jar

MultiMap jar: dbfs:/FileStore/jars/largeknn/OS\_MultiMap\_1\_0.jar

RADIG2 jar: dbfs:/FileStore/jars/largeknn/radig2\_v0\_1.jar

The three library notebooks should be stored in the same folder.

# 4 Process

## 4.1 Select or Generate Partition Grid

Please use GridGenerator Java class to generate partition grid.

## 4.2 Data Preprocess

The Scala function **OR4NN\_Preprocess\_Sedona** pre-processes the data, saves the result and also return the result as a dataframe. This returned dataframe may be used as input for the subsequent KNN query. Therefore, for one-off query, or\_path may be set to null so the pre-process will not be saved and the returned dataframe is used.

OR4NN\_Preprocess\_Sedona(objDf:DataFrame, obj\_id\_nm:String, obj\_geom\_nm:String, partDf:DataFrame, part\_id\_nm:String, part\_geom\_nm:String, K:Int, grid:GridGenerator, or\_path:String)

Please refer to the two example notebooks on details.

Note that pre-process function requires the geometry columns in both partition and object dataframes in Sedona Geometry udt type, not wkt or geojson.

## 4.3 Query

The Scala function **compKNN** computes K nearest neighbours for the queries in the query dataframe.

compKNN(K:Int, qryDf:DataFrame, qry\_id\_nm:String, qry\_geom\_nm:String, orDf:DataFrame, part\_id\_nm:String, part\_geom\_nm:String, objects\_nm:String, obj\_id\_nm:String, obj\_geom\_nm:String, part\_res:Double, save\_Path:String)

The object dataframe **orDf** may be loaded from saved pre-process result, or may be from the returned dataframe of pre-process function.

Please refer to the two example notebooks on details.

Note that pre-process function requires the geometry columns in both query and object-range dataframes in Sedona Geometry udt type, not wkt or geojson.