Python: Solving Large Network Problems

Deelesh Mandloi
Melinda Morang
Code and slides:
http://esriurl.com/ds18pyslnp
ArcGIS Network Analyst Extension for transportation analysis

Coverage
- Service Area

Optimization
- Location-Allocation
- Vehicle Routing Problem

Point-to-point routing
- Route
- Closest Facility
- Origin-Destination Cost Matrix
What is a large problem?

- Can’t be solved in one calculation
  - Memory limits
  - Service limits
  - Unreasonable calculation time
- Large number of inputs
- Large number of outputs
  - number of origins x number of destinations
  - $1000 \times 1000 = 1,000,000$

Examples
- Calculate drive time for all patients to all medical clinics within 100 miles
- Calculate the network distance from every parcel to every other parcel
Today’s goal

Solve large OD Cost Matrix
- Any number of origins and destinations
- Using local data or a service
- With or without a time/distance limit
- Output a single feature class
Network Analyst workflow

Local network dataset
1. Make analysis layer
2. Add locations
3. Solve
4. Work with output

Remote service
1. Call service
2. Work with output
How to optimize solving a large problem

- Reduce problem size
- Chunk data
- Spatially sort data
- Solve in parallel
- Pre-calculate location fields
- Use network dataset layer
Reduce problem size

• Use a time/distance limit
• Find only the K nearest
• Use only destinations within a reasonable straight-line distance of origins
  - Watch out for the case where none are selected
Chunk data

- Break up origins and destinations into chunks of reasonable size
- Iteratively solve each chunk
- Chunk size depends on service limits or memory limits
- Consider number of origins x number of destinations

```python
# Select the origins and destinations to process
origins_where_clause = "{} >= {} And {} <= {}".format(self.origins_oid_field_name, origins_criteria[0], self.origins_oid_field_name, origins_criteria[1])
arcpy.management.MakeFeatureLayer(self.origins, self.input_origins_layer, origins_where_clause)
```
Spatially sort data

- Sort geoprocessing tool
- Sort by Shape field using Peano curve
- Requires Advanced license
Chunk of 1000 unsorted origins
Destinations within 100 miles of 1000 origins

214,449 destinations selected (98%)
Destinations within 100 miles of 1000 sorted origins

5,653 destinations selected (3%)
Solve in parallel

- concurrent.futures
  - from concurrent import futures

- Multiprocessing: Spin up multiple processes and run solves on multiple cores
  - Better choice!
  - Can only run from standalone python
  - Can’t write to same gdb from multiple processes
  - Can’t share NA layer across processes

- Multithreading: Use multiple threads in the same process
  - Not good for CPU-intensive problems
  - Does not work with arcpy
  - Only use if:
    - Writing a script tool to run in the app
    - Calling a service
Pre-calculate location fields

- Define how a point snaps to the network
- Calculate them in advance if you’re using your points more than once
- Use field mapping in Add Locations to use existing location fields
- Only works for local data. Not for services.

```python
logger.debug("Calculating network locations for %s", input_features)
result = arcpy.na.CalculateLocations(output_features, network_data_source, "20 Miles",
ODCostMatrix.get_nsd_search_criteria(network_data_source),
travel_mode=travel_mode)
```
Use network dataset layer

- Opening from catalog path is slow
- Even slower for licensed data or data on UNC path
- Open once by making a Network Dataset Layer; then use the layer.

```python
self.logger.debug("Creating network dataset layer")
arcpy.MakeNetworkDatasetLayer(self.network_data_source, self.nds_layer_name)
```
Outline of today’s code

Goal: Solve OD of any size using local data or a service, write it out to a single feature class

- Components:
  - Preprocessing
    - Sorting spatially
    - Calculate network locations
  - Solving
    - Chunking
    - Solving in parallel
  - Post-processing
    - Merging results
Let's look at some code!

```python
# Compute OD cost matrix
od_line_fcs = []
job_folders_to_delete = []
# Run on multiple processes or threads when solving large OD:
if origins_count * destinations_count > inputs["max_od_size"]:
    if ODCostMatrix.is_nfs_service(inputs["network_data_source"]):
        max_workers = os.cpu_count() // 2
        pool = Futures.ProcessPoolExecutor
    else:
        max_workers = (os.cpu_count() // 2) - 1
        pool = Futures.ProcessPoolExecutor

with pool(max_workers=max_workers) as executors:
    results = executors.map(solve_od_cost_matrix, inputs_iter, ranges)
    for result in results:
        if result["solveSucceeded"]:
            od_line_fcs.append(result["outputLines"])
            job_folders_to_delete.append(result["jobFolder"])
        else:
            logger.warning("solve failed for job id %s", result["jobId"])
            logger.debug(result["solveMessages"])
```
Wrap-up

- Reduce problem size
- Chunk data
- Spatially sort data
- Solve in parallel
- Pre-calculate location fields
- Use network dataset layer

Code and slides:
http://esriurl.com/ds18pyslnp