Python: Performing Transportation Network Analysis

Melinda Morang & Max Zeng

2021 ESRI
DEVELOPER SUMMIT

Code and slides:

https://arcg.is/0bnver0

Agenda

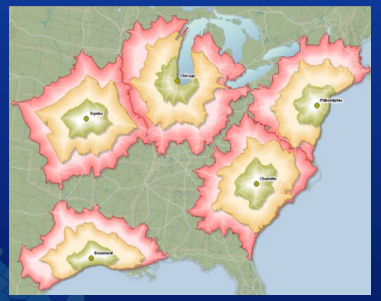
- Intro to Network Analyst
- Network analysis with Python in ArcGIS Pro using arcpy.nax
  - arcpy.nax solver classes
  - arcpy.nax.NetworkDataset class
- Network analysis using web services with the ArcGIS API for Python

Intro to Network Analyst

ArcGIS Network Analyst Extension

for transportation analysis

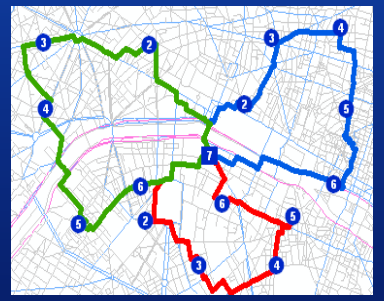
Coverage



Service Area

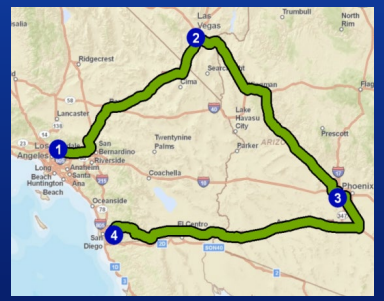
Optimization

Location-Allocation

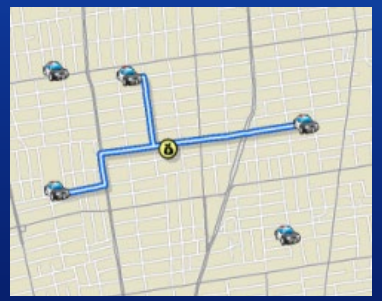


Vehicle Routing Problem

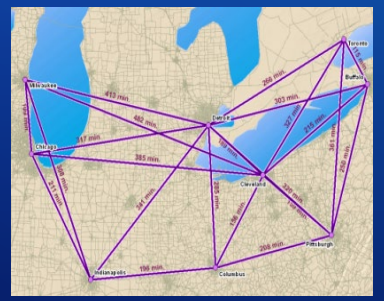
Point-to-point routing



Route

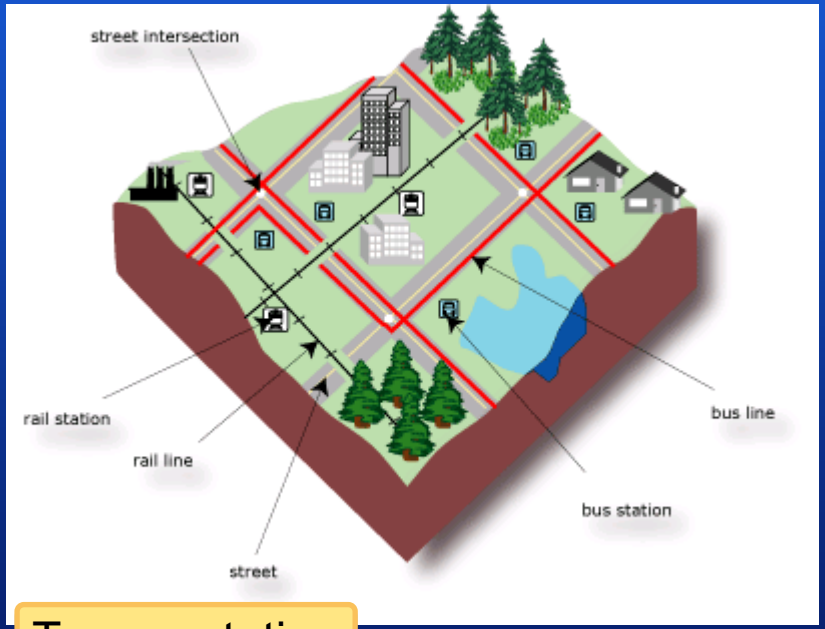


Closest Facility



Origin-Destination Cost Matrix

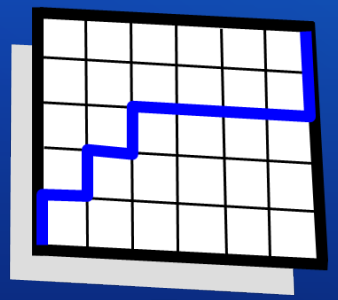
Analysis is performed on a network dataset



Transportation Network

Data Model

Network Dataset



Geodatabase

Where do I get a network dataset?

- Purchase StreetMap Premium for ArcGIS
  - High quality ready-to-use network dataset
  - Can add your own street data
- Build your own
  - Your organization's data
    - Try the ArcGIS Pro Tasks to Create a Local Government Network Dataset
  - TIGER
  - OpenStreetMap
- Use the ArcGIS Online services
  - You don't need a network. You just call the services.

Options for network analysis with python

	arcpy.nax module	ArcGIS API for Python
Use ArcGIS Online services	Yes	Yes
Use Enterprise services	Yes	Yes
Use your own local street data	Yes	No
Requires ArcGIS Pro	Yes	No
Requires Network Analyst extension license	Yes, when using your own street data. Otherwise, no.	No
Consumes service credits	Yes, if using ArcGIS Online services. Otherwise, no.	Yes, if using ArcGIS Online services. Otherwise, no.
Has analysis limits	Yes, if referencing a service with analysis limits. Otherwise, no.	Yes, if referencing a service with analysis limits. Otherwise, no*.

*To handle large outputs in the ArcGIS API for Python, you may need to adjust some of the advanced settings. The arcpy.nax module is generally better at handling large outputs.

Network analysis with Python in ArcGIS Pro

arcpy.nax

- Easy-to-use Python module
  - Perform network analysis
  - Access connectivity information from a network dataset
- Available starting at ArcGIS Pro 2.4
  - Not available in ArcMap

arcpy.nax

- Faster
- More Pythonic

Help me choose

arcpy.na

- Required for working with network analysis layers
- Required for publishing map service with network analysis capability
- Required for code that works with ArcGIS Pro versions prior to 2.4

```
(arcgispro-py3) F:\Pro\Demos\nax>python sa_arcpy_na.py
Analysis completed in 11.02 seconds

(arcgispro-py3) F:\Pro\Demos\nax>python sa_arcpy_nax.py
Analysis completed in 1.87 seconds
```

5,10,15 minutes drive time areas around a single facility and exporting results to a file geodatabase feature class

Network Analysis Workflow with arcpy.nax

1. Initialize the analysis object (based on a specific network data source)
2. Set the properties for the analysis
3. Load the inputs
4. Solve the analysis
5. Access and work with the results

Common to all the network analyses

Analysis (Solver) Classes

- Analysis class for each solver
  - Set properties
  - Load inputs
  - Solve
- Analysis class for solve results
  - Access outputs

ServiceArea

Properties

accumulateAttributeNames
allowSaveLayerFile
defaultImpedanceCutoffs
distanceUnits
excludeSourcesFromPolygonGeneration
geometryAtCutoff
geometryAtOverlap
ignoreInvalidLocations
networkDataSource
outputType
overrides
polygonBufferDistance
polygonBufferDistanceUnits
polygonDetail
searchQuery
searchTolerance
searchToleranceUnits
timeOfDay
timeUnits
timeZone
travelDirection
travelMode

Methods

addFields()
count()
fieldMappings()
fieldNames()
insertCursor()
load()
solve()

ServiceAreaResult

isPartialSolution
solveSucceeded

count()
export()
fieldNames()
saveAsLayerFile()
searchCursor()
solverMessages()

```python
# initialize route solver object

route = arcpy.nax.Route(nd_source)

# set properties for the analysis

nd_travel_modes = arcpy.nax.GetTravelModes(nd_source)
travel_mode = nd_travel_modes["Driving Time"]
route.travelMode = travel_mode

# load inputs

route.load(arcpy.nax.RouteInputDataType.Stops, input_stops)

# solve route

result = route.solve()
```

Basic arcpy.nax workflow

Analysis inputs and outputs

- Inputs
  - Load a feature class using the load() method
  - Insert inputs using the insertCursor() method
- Outputs
  - Export to a feature class using the export() method
  - Directly access outputs using the searchCursor() method
  - Save to a layer file using saveAsLayerFile() (mostly for debugging)
  - Save Route Data for use in Navigator with saveRouteData()
    - Route, VRP, & Closest Facility only

Inputs: load() and fieldMappings()

- Load inputs from an existing feature class
- Use field mappings to map fields
  - Map fields from your input data into analysis fields
  - Map pre-calculated location fields

- Learn more about input and output types and fields

VehicleRoutingProblem input data types

The input data types that can be specified when performing a vehicle routing problem (VRP) analysis are described below.

Orders

Specify one or more locations that the routes of the VRP analysis should visit. An order can represent a delivery (for example, furniture delivery), a pickup (such as an airport shuttle bus picking up a passenger), or some type of service or inspection (a tree trimming job or building inspection, for instance).

The data type supports the following fields:

Field	Description	Data type
Name	The name of the order. The name must be unique. If the name is left null, a name is automatically generated at solve time.	String
Description	The descriptive information about the order. This can contain any textual information for the order and has no restrictions for uniqueness. You may want to store a client's ID number in the Name field and the client's actual	String

```
# load input data from external system

input_orders = os.path.join(input_gdb, "Stores")
input_depots = os.path.join(input_gdb, "DistributionCenter")
input_routes = os.path.join(input_gdb, "SolveVehicleRoutingProblem_Routes")

# The order service time is in a field called "DeliveryDelay" and should default to 10 minutes
# if the field does not have a value.  Set up this field mapping:

field_mappings = vrp.fieldMappings(arcpy.nax.VehicleRoutingProblemInputDataType.Orders)
field_mappings["ServiceTime"].mappedFieldName = "DeliveryDelay"
field_mappings["ServiceTime"].defaultValue = 10

vrp.load(arcpy.nax.VehicleRoutingProblemInputDataType.Orders, input_orders, field_mappings)
vrp.load(arcpy.nax.VehicleRoutingProblemInputDataType.Depots, input_depots)
vrp.load(arcpy.nax.VehicleRoutingProblemInputDataType.Routes, input_routes)
```

Outputs: export()

- Copy outputs to a feature class
- Retains all records and fields

```
vrp_result.export(arcpy.nax.VehicleRoutingProblemOutputDataType.Stops, output_stops)
vrp_result.export(arcpy.nax.VehicleRoutingProblemOutputDataType.Routes, output_routes)
```

```python
# load input data from external system

input_orders = os.path.join(input_gdb, "Stores")
input_depots = os.path.join(input_gdb, "DistributionCenter")
input_routes = os.path.join(input_gdb, "SolveVehicleRoutingProblem_Routes")

# The order service time is in a field called "DeliveryDelay" and should default to 10 minutes
# if the field does not have a value.  Set up this field mapping:

field_mappings = vrp.fieldMappings(arcpy.nax.VehicleRoutingProblemInputDataType.Orders)
field_mappings["ServiceTime"].mappedFieldName = "DeliveryDelay"
field_mappings["ServiceTime"].defaultValue = 10

vrp.load(arcpy.nax.VehicleRoutingProblemInputDataType.Orders, input_orders, field_mappings)
vrp.load(arcpy.nax.VehicleRoutingProblemInputDataType.Depots, input_depots)
vrp.load(arcpy.nax.VehicleRoutingProblemInputDataType.Routes, input_routes)
```

load() with field mappings

Inputs: insertCursor()

- Directly insert inputs
- Similar to working with arcpy.da.InsertCursor()
- Can be faster since you do not need to first convert your data into a feature class or a table

```
# load data using an insert cursor

with route.insertCursor(arcpy.nax.RouteInputDataType.Stops, ["Name", "SHAPE@XY",]) as stops_insert_cursor:
    stops_insert_cursor.insertRow(["Home", (-122.4378792, 37.7955582)])
    stops_insert_cursor.insertRow(["Work", (-122.3979990, 37.7936440)])
```

Outputs: searchCursor()

- Directly access outputs
- Similar to working with arcpy.da.SearchCursor()
- Can be faster in certain workflows since you do not need to first export the outputs to a feature class or a table.

```
# examine result with a search cursor

for row in result.searchCursor(arcpy.nax.RouteOutputDataType.Directions, ["ArriveTime", "Text"]):
    print(f'{row[0]:%H:%M:%S} {row[1]}')
```

```python
# load data using an insert cursor

with route.insertCursor(arcpy.nax.RouteInputDataType.Stops, ["Name", "SHAPE@XY",]) as stops_insert_cursor:
    stops_insert_cursor.insertRow(["Home", (-122.4378792, 37.7955582)])
    stops_insert_cursor.insertRow(["Work", (-122.3979990, 37.7936440)])

# solve route

result = route.solve()

if not result.solveSucceeded:
    print("Solved failed")
    print(result.solverMessages(arcpy.nax.MessageSeverity.All))
    sys.exit(0)

# examine result with a search cursor

for row in result.searchCursor(arcpy.nax.RouteOutputDataType.Directions, ["ArriveTime", "Text"]):
    print(f'{row[0]:%H:%M:%S} {row[1]}')
```

Inputs and outputs with cursors

Tips and Tricks: Units for the analysis

- As part of the analysis settings, you can set the timeUnits and distanceUnits properties
- These can be different than the units of the cost attributes on your network dataset
- The travel costs in the output are always returned in the units you specify.

```
# set properties for the analysis

od.timeUnits = arcpy.nax.TimeUnits.Seconds
od.distanceUnits = arcpy.nax.DistanceUnits.Feet
```

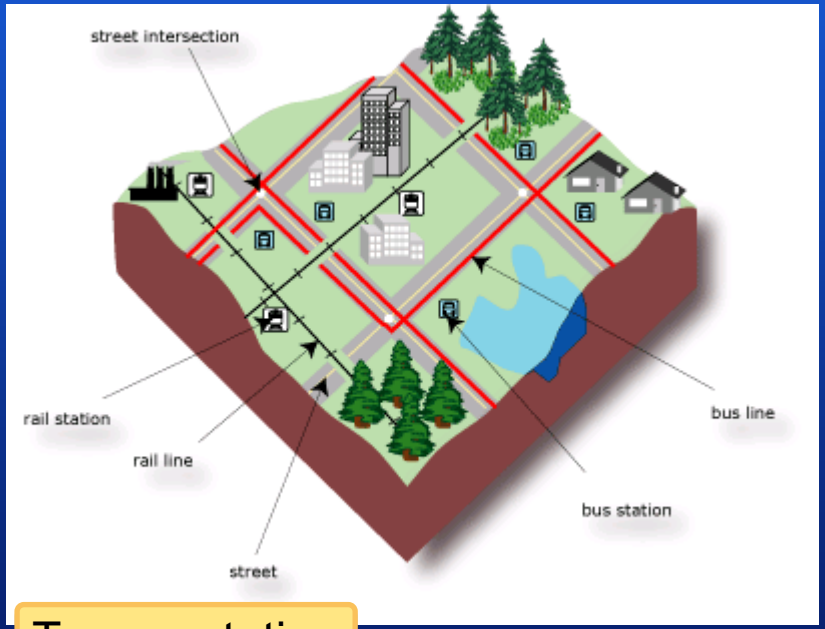
Tips and Tricks: Use the network dataset layer

- Opening a network dataset 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 name (not the layer object)

```
nd_path = os.path.join(cwd, "SanFrancisco.gdb", "Transportation", "Streets_ND")
nd_layer_name = "SanFrancisco"

arcpy.nax.MakeNetworkDatasetLayer(nd_path, nd_layer_name)

# initialize route solver

route = arcpy.nax.Route(nd_layer_name)
```

NetworkDataset class

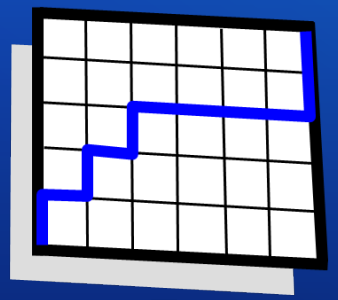
Analysis is performed on a network dataset



Transportation Network

Data Model

Network Dataset



Geodatabase

Accessing network dataset properties and components with Python

- arcpy.nax.NetworkDataset class
- Access basic network dataset properties
  - Is the network dataset built?
  - Describe properties (same as arcpy.Describe())
- Retrieve network edges, junctions, and turns and their properties
  - Attribute values
  - Connectivity
  - Optionally specify a time of day or a travel mode

Accessing network dataset properties and components with Python

Cursors to iterate over network elements

NetworkDataset
buildTimestamp isBuilt travelModes
checkIntersectingFeatures() describe() edges() getDataSourceFromSourceID() junctions() turns()

Edge properties

EID
SOURCEID
SOURCEOID
FROMPOSITION
TOPOSITION
DIRECTION
FROMJUNCTION
TOJUNCTION
ISRESTRICTED
COST
TIME
DISTANCE

Attribute values

such as restrictions and impedance

Using Network Dataset with other software packages

- Export the network dataset for use with 3rd Party libraries, such as networkx
- Use algorithms not available in Network Analyst but available in other libraries
- Example: Download the Find Network Subgraphs tool

```
# Initialize networkx graph
# Use a non-directed multigraph so we can have parallel edges and self-loops, but directio
graph = networkx.MultiGraph()

# Add all network edges to networkx graph
arcpy.AddMessage("Creating graph from network edges...")
props = [fld_sourceid, fld_sourceoid, "DIRECTION", "FROMJUNCTION", "TOJUNCTION"]
for edge in ND.edges(props):
    if not edge[2]:
```

```
    edge_at[illegible] = {fld_sourceid: source_id, fld_sourceoid: source_oid}
    # Add the edge to the graph
    graph.add_edge(from_junc, to_junc, attr_dict=edge_attrs)
```

```python
# Initialize networkx graph
# Use a non-directed multigraph so we can have parallel edges and self-loops, but directic
graph = networkx.MultiGraph()

# Add all network edges to networkx graph
arcpy.AddMessage("Creating graph from network edges...")
props = [fld_sourceid, fld_sourceoid, "DIRECTION", "FROMJUNCTION", "TOJUNCTION"]
for edge in ND.edges(props):
    if not edge[2]:
        # This is a non-directed graph, so skip all the edges in the backwards direction.
        continue
    from_junc = edge[3]
    to_junc = edge[4]
    source_id = edge[0]
    source_oid = edge[1]
    # Create list of all source ids for later use
    if source_id not in source_ids:
        source_ids.append(source_id)
    # Store sourceid and sourceoid for each edge
    edge_attrs = {fld_sourceid: source_id, fld_sourceoid: source_oid}
    # Add the edge to the graph
    graph.add_edge(from_junc, to_junc, attr_dict=edge_attrs)
```

Accessing network dataset properties with the NetworkDataset class

Network Analyst with web services using the ArcGIS API for Python

ArcGIS API for Python

- A pythonic API to work with ArcGIS Online or ArcGIS Enterprise services
- Don’t need ArcGIS Pro
- Easy to use with Jupyter notebooks
- Included in ArcGIS Notebooks in Online, Enterprise or Pro.

Accessing network analysis services

Solvers	network.analysis	features.analysis
Route	find_routes	connect_origins_to_destinations
Service Area	generate_service_areas	create_drive_time_areas
Closest Facility	find_closest_facilities	find_nearest
Location Allocation	solve_location_allocation	choose_best_facilities
Vehicle Routing Problem	solve_vehicle_routing_problem	plan_routes
Origin Destination Cost Matrix	generate_origin_destination_cost_matrix	Not available

- **Access to all solvers and analysis options**
- **Faster**
- **Feature collection output only**

- **Workflow driven**
- **Feature service or feature collection output**

ArcGIS API for Python: Learn more

- Do the tutorials
- Check out another DevSummit session
- Download our code sample

Use connect origins to destinations tool to find walking distance between student and assigned school.

Because the school field values in the students table correspond to the name field values in the schools table, you can use these two fields as the special ID fields that indicate which origin should connect to which destination. Set the ID field in origins parameter to school and the Matching ID field in destinations parameter to name. Set route shape to StraightLine so it is easy to visualize which student goes to which school. The calculation is based on actual routing streets even though the output route shape is StraightLine.

In [ ]:
```
import datetime
result = arcgis.features.use_proximity.connect_origins_to_destinations(
    origins_layer=students_fc,
    destinations_layer=schools_fc,
    measurement_type="Walking Distance",
    origins_layer_route_id_field="school",
    destinations_layer_route_id_field="name",
    route_shape="StraightLine")
```

Network analysis with web services using arcgis.features.analysis

Wrap-up

Summary

- arcpy.nax solver objects
  - Perform network analysis in ArcGIS Pro
  - Use local data or a service
  - Write scripts and script tools
- arcpy.nax.NetworkDataset class
  - Access network graphs
  - Extend network analysis capabilities
- ArcGIS API for Python
  - Perform network analysis using web services without ArcGIS Pro
  - arcgis.network.analysis or arcgis.features.analysis

Resources

- arcpy.nax documentation
- Slides and code:

https://arcg.is/0bnver0

esri®

THE SCIENCE OF WHERE®

Copyright © 2021 Esri. All rights reserved.

Please provide your feedback for this session by clicking on the session survey link directly below the video.