



# ArcPy: Solving large transportation analysis problems

Melinda Morang & Jeff Wickstrom

2022 ESRI DEVELOPER SUMMIT

Welcome to “ArcPy: Solving large transportation analysis problems”.

My name is Jeff. My co-presenter Melinda and I are both on the Network Analyst Team.

# Agenda

- Intro/background
- Techniques for solving large problems
- Python script walk-through
- Working with services

Code and slides:

**<https://github.com/Esri/large-network-analysis-tools>**

All the code as well as these slides are available for download at this address. Go ahead and take a moment to make a note of it.

We'll show it again at the end in case you miss it.

# Introduction

# 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

The ArcGIS Network Analyst is an ArcGIS Extension for solving transportation analysis problems.

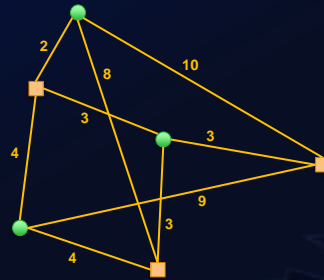
It has tools for

- assessing the coverage of different facilities
- optimizing things like site location and fleets of vehicles
- and for point-to-point routing, such as “How do I get from point A to point B?”, “What’s the Closest Facility”, and for solving origin-destination cost matrices.

## OD Cost Matrix

Calculates the travel time or distance between a set of origins and a set of destinations.

- Can use a time/distance limit
- Can limit the number of destinations to find



In today's presentation we're going to be talking about the origin-destination, or OD, cost matrix. Although we'll be focusing on this tool, a lot of the techniques we'll be discussing apply to the other Network Analyst tools as well.

OD Cost Matrix is used for finding travel time or distances between sets of origins and sets of destinations.

- This tool has the ability to use a time or distance limit, which is important for large problems.
- You can also limit the number of destinations you want to find. So instead of saying "What's the travel time from every origin to every destination?", you can say "What the travel time from every origin to the closest five destinations?".

Of all the Network Analyst solvers, this is the one that's most optimized for efficiency in performing its calculations and delivering results.

# What is a large problem?

- Can't be solved in one calculation
  - Unreasonable calculation time
  - Memory limits
  - Service limits
- 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

So what is a large problem? A large problem is one that can't be solved in one calculation in the usual ways.

- maybe because it would take way too long, or
- maybe you hit some memory limits on your machine.
- If you're using a service, maybe you run into some limits of what the service is capable of providing.

Usually large problems involve a large number of inputs, and they may also have large output.

The OD Cost Matrix result comes back as a table that includes the number of origins times the number of destinations.

For example, if you have 1000 origins and 1000 destinations, your output has a million records in it. The output can sometimes be the large part.

Some examples of a large problem include...

- 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 choices

- Single feature class
- Set of CSV files
- Set of Apache Arrow files



Today's goal is to create a script that we can use to solve a large OD Cost Matrix with

- any number of origins and destinations,
- local data or a service for the network data source,
- and with or without a time or distance limit.

We'll write the output to either a single feature class, a set of CSV files, or a set of Apache Arrow files.



## What is a local solve and a service solve?

- Local solve
  - Solving a network analysis on your computer using a network dataset stored on your disk
  - Solve uses computing resources of your computer
- Service solve
  - Solving a network analysis using a GIS web service
  - Solve uses computing resources of GIS Server

What do we mean by a local solve vs a service solve?

A local solve means I'm solving the network analysis problem on my own computer using my own network dataset stored on my own disk. I'm using the computing resources of my own machine to solve the problem.

When I'm referring to a service, what I mean is solving the network analysis problem using some kind of GIS web service. The data is sent to the service, and the computing resources of the server are used to calculate the result, and the result is sent back to me.

Both of these are a valid way to solve the problem, and it really depends on your situation. We'll go into this in more detail later.

## 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. Work with the results

*Common to all the network analyses*

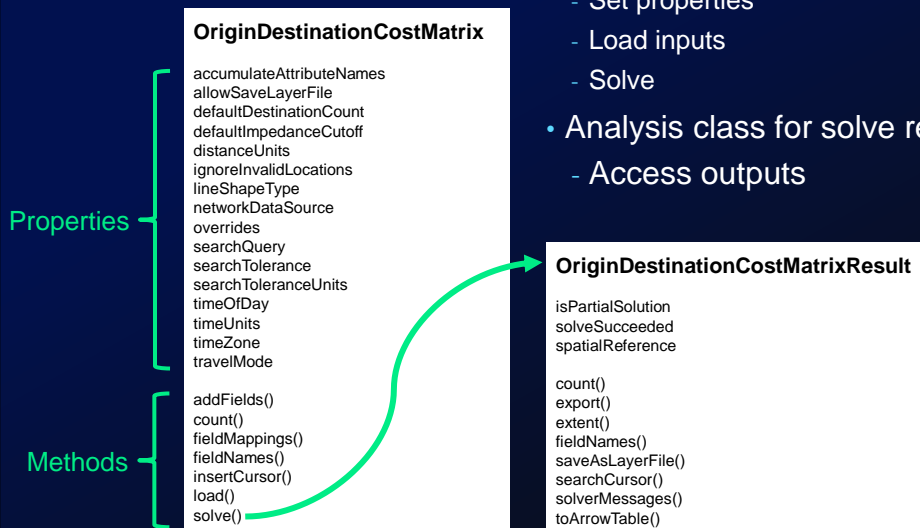
ArcGIS Pro has a Python module for performing network analysis, `arcpy.nax`. This module features easy-to-use classes and methods for performing network analysis using either local data or a service.

This is a summary of the workflow you'll typically use for performing a network analysis, such as an OD Cost Matrix. The sample code we'll be showing later uses this workflow.

(Read steps from slide)

## arcpy.nax Analysis (Solver) Classes

- Easy-to-use python objects for network analysis
- Analysis class for each solver
  - Set properties
  - Load inputs
  - Solve
- Analysis class for solve results
  - Access outputs



In the arcpy.nax module, each type of network analysis, or solver, has its own python class.


When you instantiate the solver object, you set the network data source. This can be either a network dataset or a URL to a service.

Once you instantiate the solver object class, you can set the various analysis properties. For example, the travel mode to use for the analysis, the defaultDestinationCount and the defaultImpedanceCutoff.

The class has methods for loading data and solving as well as some other helper methods.

When you solve the analysis using the solve() method, it creates an instance of a result object for the particular solver. You can access the analysis results from that result object in several different ways.

The scripts we're going to show today will make use of the OD Cost Matrix solver object.

The background of the slide is a deep blue. It features several abstract elements: a cluster of small, light blue squares in the upper left; a series of larger, semi-transparent blue squares arranged in a descending staircase pattern on the right; a network of thin, light blue lines forming a web-like structure; and a large, curved area in the bottom right corner filled with many closely spaced, concentric blue lines.

## Techniques for solving large problems

Before we jump into our code, we'll talk about some techniques for solving large problems.

# How to optimize solving a large problem

- Reduce problem size
- Eliminate irrelevant data
- Chunk data
- Spatially sort data
- Solve in parallel
- Pre-calculate network location fields
- Use network dataset layer
- \*\*Update to ArcGIS Pro 2.9 or higher

There's a lot you can do to optimize the problem you're going to solve before you actually try to solve it. Some techniques include...

- Reducing problem size
- Eliminating irrelevant data
- Chunking the data
- Spatially sorting the data
- Solving chunks in parallel
- Pre-calculating network location fields
- Using a network dataset layer

We'll go through each one of these in more detail.

Finally, note that we made some substantial performance improvements to the OD solver objects at the ArcGIS Pro 2.9 release. You should definitely update your software for the best results.

## Reduce problem size

- Find only the K nearest
  - defaultDestinationCount property
- Use a time/distance limit
  - defaultImpedanceCutoff property

<b>defaultDestinationCount</b> (Read and Write)	<p>The maximum number of destinations to find per origin. The default is <i>None</i>, which means to find all destinations.</p> <p>The value set in this property can be overridden on a per-origin basis using the <i>TargetDestinationCount</i> field in the input origins.</p>	Integer
<b>defaultImpedanceCutoff</b> (Read and Write)	<p>The impedance value at which to stop searching for destinations from a given origin.</p> <p>If the travel mode used in the analysis uses a time-based impedance attribute, the <i>defaultImpedanceCutoff</i> is interpreted in the units specified in the <i>timeInits</i> property. If the travel mode used in the analysis uses a distance-based impedance attribute, the <i>defaultImpedanceCutoff</i> is interpreted in the units specified in the <i>distanceInits</i> property. If the travel mode's impedance attribute is neither time-based nor distance-based, the <i>defaultImpedanceCutoff</i> value is interpreted in the units of the impedance attribute. The default is <i>None</i>, which means that no cutoff is applied.</p> <p>The <i>defaultImpedanceCutoff</i> can be overridden on a per-origin basis using the <i>Cutoff</i> field in the input origins.</p>	Double

The first thing to do is to reduce your problem size. You might find out after you've done this that you don't actually have a large problem after all.

One thing you can do to reduce the problem size is to find only the K nearest destinations. Do you really need to find the travel time between all origins and all destination, or can you find only the 10 closest destinations?

For example, do you need to calculate the driving time from a patient to all doctor's offices, or can you just return the 10 closest options? You can use the *defaultDestinationCount* property to limit the number of destinations to find.

You can use a time or distance cutoff limit so you find only destinations that fall within this limit. Do you care about results that are more than 100 miles away?

For example, if your patient is located in California, you probably don't need to give them doctor's offices in Maine as options. You can use the *defaultImpedanceCutoff* property for this.

## Eliminate irrelevant data

- Remove destinations that aren't within a reasonable straight-line distance of origins prior to running the OD Cost Matrix analysis
- Applies only if you're using a time/distance limit
- Use Select Layer By Location
- Watch out for the case where none are selected

```
# Use SelectLayerByLocation to select those within a straight-line distance
self.logger.debug(
    # "Eliminating destinations outside of distance threshold {cutoff_dist} {self.distance_units.name}..."
    self.input_destinations_layer_obj = run_gp_tool( arcpy.management.SelectLayerByLocation, [
        self.input_destinations_layer,
        "WITHIN_A_DISTANCE_GEODESIC",
        self.input_origins_layer,
        # "{cutoff_dist} {self.distance_units.name}",
    ], log_to_use=self.logger).getOutput(0)
```



Another thing to do is to eliminate irrelevant data. This helps reduce the overall problem size. It applies primarily to the case where you are using a time or distance limit.

If you only care about finding destinations within 100 miles *network distance* of each origin, you can exclude from the analysis any destinations that are more than 100 miles *straight line distance* away because you know that those destinations will never be included in the solution. This drastically reduces the number of inputs you have to send to the OD Cost Matrix analysis in the first place.

Now, you don't know the driving distance from the origins to the destinations prior to running the OD Cost Matrix. However, the network distance from an origin to a destination will always be greater than or equal to the straight-line distance and some small additional buffer just to be on the safe side.

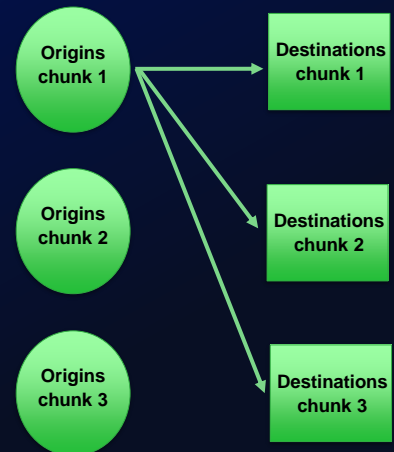
You can do this using the Select Layer By Location tool and a straight-line distance calculation to eliminate all destinations outside of your cutoff.

One thing to watch out for is the case where no destinations are found within this cutoff distance. The Select Layer By Location tool, rather than returning an empty set, just returns the entire dataset, so you have to explicitly handle this case.

# 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

```
# Select the origins with ObjectIDs in this range
self.logger.debug("Selecting origins for this chunk...")
origins_where_clause = (
    f"{self.origins_oid_field_name} >= {origins_criteria[0]} "
    f"And {self.origins_oid_field_name} <= {origins_criteria[1]}"
)
self.logger.debug(f"Origins where clause: {origins_where_clause}")
self.input_origins_layer_obj = run_gp_tool(
    arcpy.management.MakeFeatureLayer,
    [self.origins, self.input_origins_layer, origins_where_clause],
    log_to_use=self.logger
).getOutput(0)
num_origins = int(arcpy.management.GetCount(self.input_origins_layer_obj).getOutput(0))
self.logger.debug(f"Number of origins selected: {num_origins}")
```



Once you've reduced the problem as far as you can, you'll probably want to chunk your data and then solve each chunk iteratively, perhaps in parallel.

So you'll create chunks of origins and chunks of destinations.

For each chunk of origins, you'll calculate the travel time or distance to the destinations in each chunk of destinations. Then, you'll do the same for each additional chunk of origins.

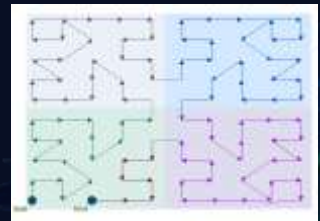
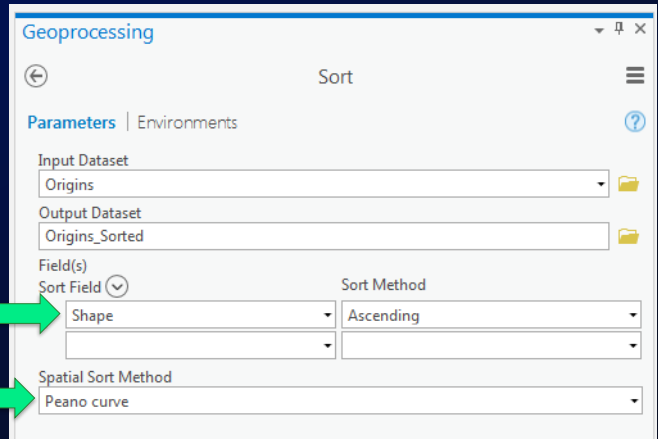
The chunk size really depends on the memory resources of the machine you're using to solve the problem or the limits of the service you're using.

Bear in mind the size of the output for each chunk is the number of origins times the number of destinations.



# Spatially sort data

- Sort geoprocessing tool
- Sort by Shape field using Peano curve
- Requires Advanced license



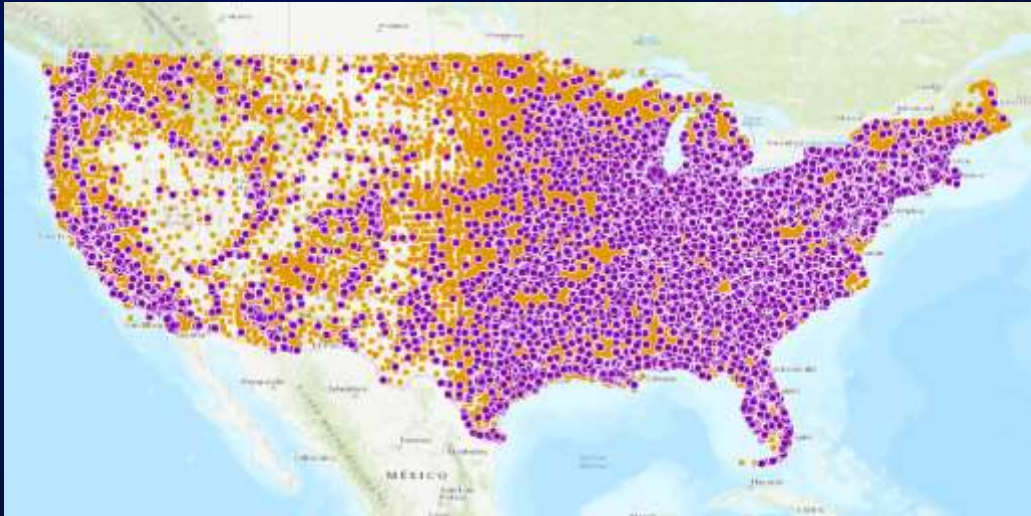
```
arcpy.management.Sort(temp_inputs, input_features, [[desc.shapeFieldName, "ASCENDING"]], "PEANO")
```

Something you can do to improve your chunking is to spatially sort your data. This allows you to make smarter chunks that will overall reduce the number of iterations you need.

The Sort geoprocessing tool can be used to spatially sort your data so that the points are all nice and clustered as it goes through the dataset.

Note that the spatial sort option requires the Advanced license.

## All origins and all destinations



26 million origins ■, 220k destinations ■

Now we'll walk through an example to help make this clearer.

Let's say I want to solve a problem where I have 26 million origins (the purple dots) and 200K destinations (the orange squares),

and I want to calculate the travel time from each origin to each destination - if the destination is within 100 miles of the origin – so we have a distance cutoff.

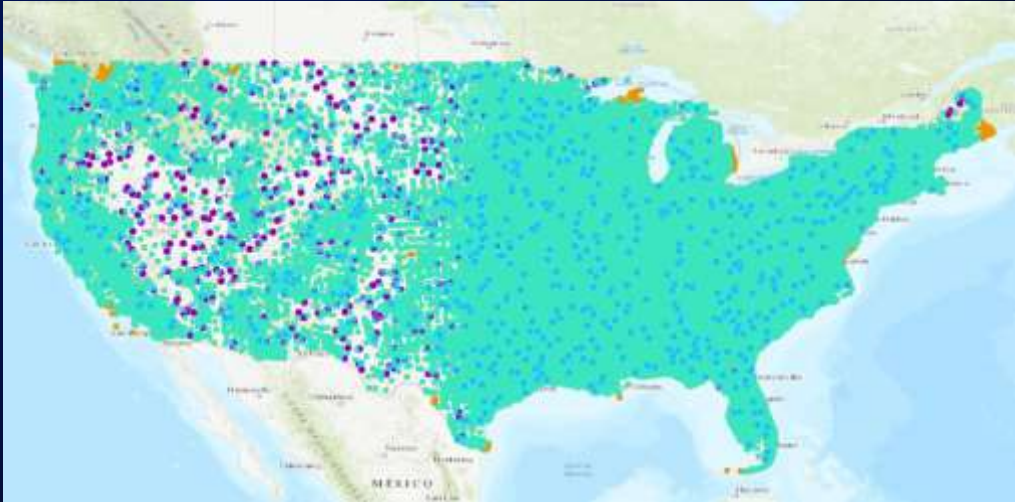
## Chunk of 1000 unsorted origins



Let's say my chunk size is 1000, and I just randomly select 1000 of my origins.

They're scattered all over the country. Now, if we find the number of destinations relevant to this chunk of origins – by eliminating any that are over 100 miles straight-line distance from any origin in this chunk....

## Destinations within 100 miles of 1000 origins



214,449 destinations selected (98%)

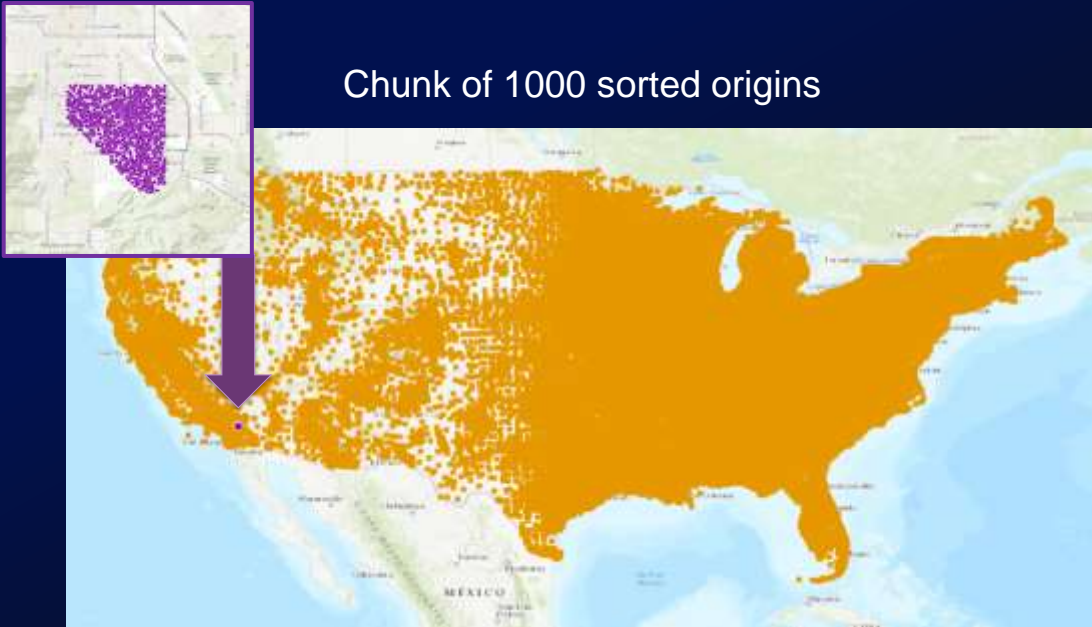
We see these selected destinations. These are all the ones relevant for this chunk of origins. They're within 100 miles of at least one of my origins in the chunk.

Because the origins are scattered all over the country, we select about 98% of the destinations. We haven't really successfully reduced the number of destinations to be searched for this chunk of origins.

We end up needing to run through 215 chunks of destinations, and it's going to take a while.

Doing the same for each chunk of origins leads to way more solve operations that if we spatially sort the origins.

## Chunk of 1000 sorted origins



If the origins are first spatially sorted, a chunk of 1000 origins are all right here in the Palm Springs area.

That little tiny purple dot.

## Destinations within 100 miles of 1000 sorted origins



5,653 destinations selected (3%)

Now if I select destinations that are within 100 miles of these sorted origins, there's fewer than 6000, which is about 3% of my destinations, as opposed to the 98% that I got when I didn't sort my origins.

Now, for this chunk of origins, I'll only need 6 chunks of destinations, and I can be confident that all the other destinations are completely irrelevant for these origins.

So we have substantially reduced the number of chunks I need to process overall.



## Solve in parallel

- concurrent.futures `from concurrent import futures`
- Multiprocessing: Spin up multiple processes and run solves on multiple cores – USE THIS ONE

```
# Use the concurrent.futures ProcessPoolExecutor to spin up parallel processes that solve the OD cost matrices
with futures.ProcessPoolExecutor(max_workers=self.max_processes) as executor:
    # Each parallel process calls the solve_od_cost_matrix() function with the od_inputs dictionary for the
    # given origin and destination O/D ranges.
    jobs = {executor.submit(solve_od_cost_matrix, self.od_inputs, range): range for range in self.ranges}
```

- Warnings:

- If running as a geoprocessing script tool, need to use subprocess module
- Can't write to same gdb from multiple processes

- Multithreading: Use multiple threads in the same process – DO NOT USE
  - Not good for CPU-intensive problems in python
  - Does not work with arcpy

Assuming I've done my chunking, I now need to solve my OD Cost Matrix in parallel. So instead of doing each chunk one at a time, I can solve several chunks at once, and that will ultimately return my results faster. And I will use all of my CPUs.

Python makes this easy with the concurrent.futures module.

Concurrent.futures gives you two options for doing things in parallel: multiprocessing and multithreading.

Multithreading is not too great for CPU-intensive problems in python, and it doesn't work very well with arcpy, so we don't recommend using that.

We recommend instead using multiprocessing to spin up multiple solve processes on multiple cores. It works with arcpy and it's really easy to run from standalone python.

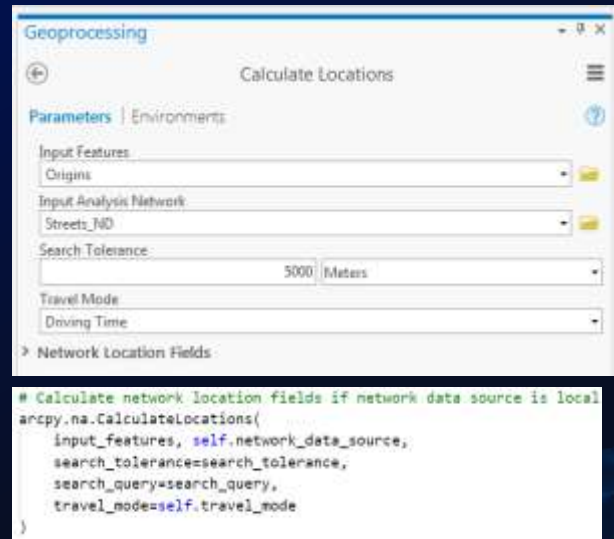
A couple of things to watch out for.

- If you're trying to write a script tool that runs within the ArcGIS Pro UI, it won't work with multiprocessing. You'll need to use the subprocess module to spin up a separate processes. The subprocess can then spin up the concurrent parallel processes.
- You also can't write to the same file geodatabase from multiple processes at the same time, so you have to be careful about how you're storing your output,

This is how the sample script does it.

# 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 when loading inputs to use existing location fields
- Only works for local data (not for services)



SourceID, SourceOID, PosAlong, SideOfEdge

If you're using your origins and destinations more than once (like in chunks that get used more than once), you can speed up the calculation by pre-calculating your network location fields.

What are network location fields? These describe where each point snaps to the network. A point is usually just somewhere floating in space, and the network location fields describe the closest point on the network dataset.

Every time you do a network analysis, the solver has to determine what these network location fields are. If you're going to be using the same origin or destination multiple times, it doesn't make sense to waste time re-calculating the network location fields each time you solve a chunk - the network location fields for a specific point aren't going to change. It's more efficient to pre-calculate the network location fields and just re-use those.

You can do this with the Calculate Locations geoprocessing tool. The fields get stored in your feature class.

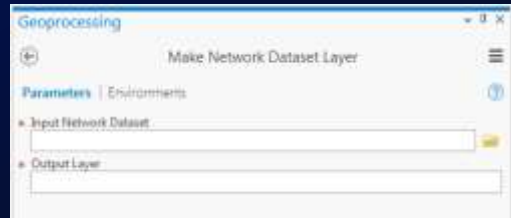
Note that if you modify your network dataset or decide to use a different network dataset or travel mode, you need to re-calculate your network location fields, since they are specific to the network dataset and the travel mode settings.

The Calculate Locations tool doesn't work with services.



# 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 name (not the layer object)



```
run_gp_tool(  
    arcpy.na.MakeNetworkDatasetLayer,  
    [self.network_data_source, nds_layer_name],  
    log_to_use=self.logger  
)
```

Finally, you can get a small additional performance improvement by using a network dataset layer instead of a catalog path to a network dataset in your script.

Using a network dataset layer by name saves having to re-open the network dataset multiple times.



## Sample script for a large OD Cost Matrix problem

[Switch presenters] Now Melinda will show you this script we've been talking about.

# Outline of today's code

Goal: Solve OD cost matrix of any size using local data or a service

- Components:

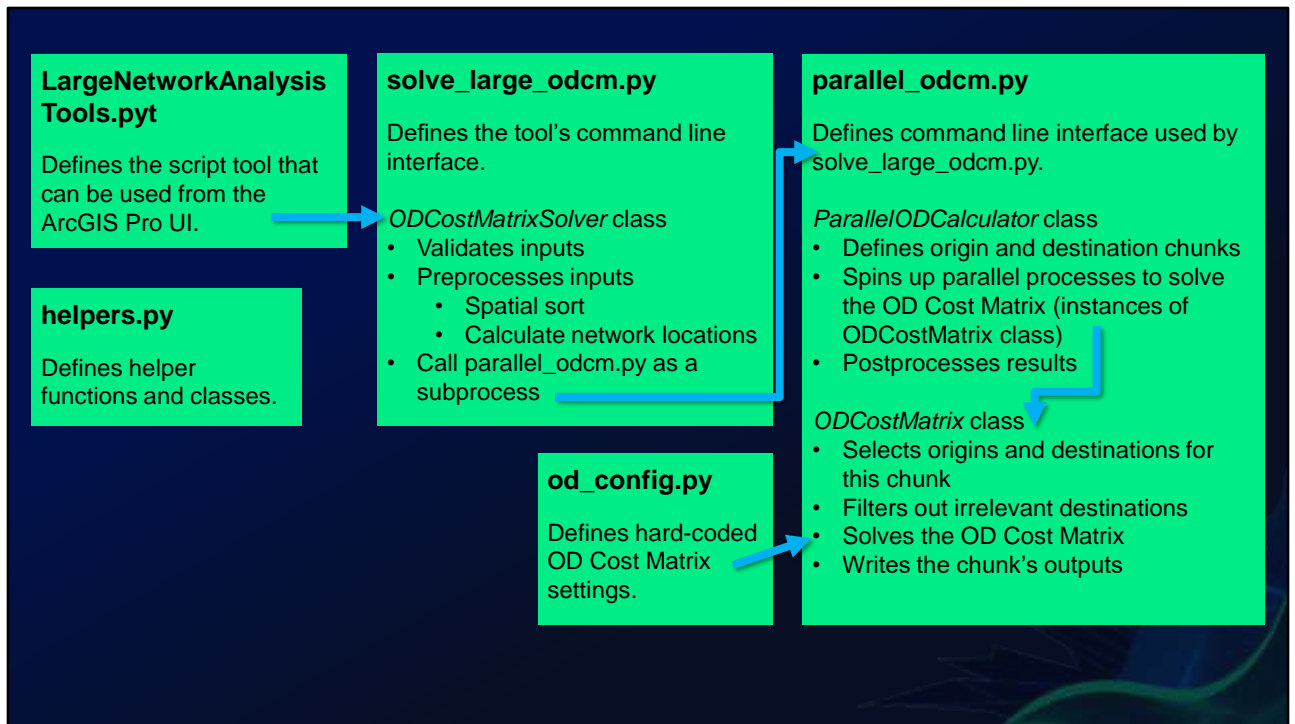
- Preprocessing
  - Sorting spatially
  - Calculate network locations
- Solving
  - Chunking
  - Solving in parallel
- Post-processing
  - Merging results

**<https://github.com/Esri/large-network-analysis-tools>**

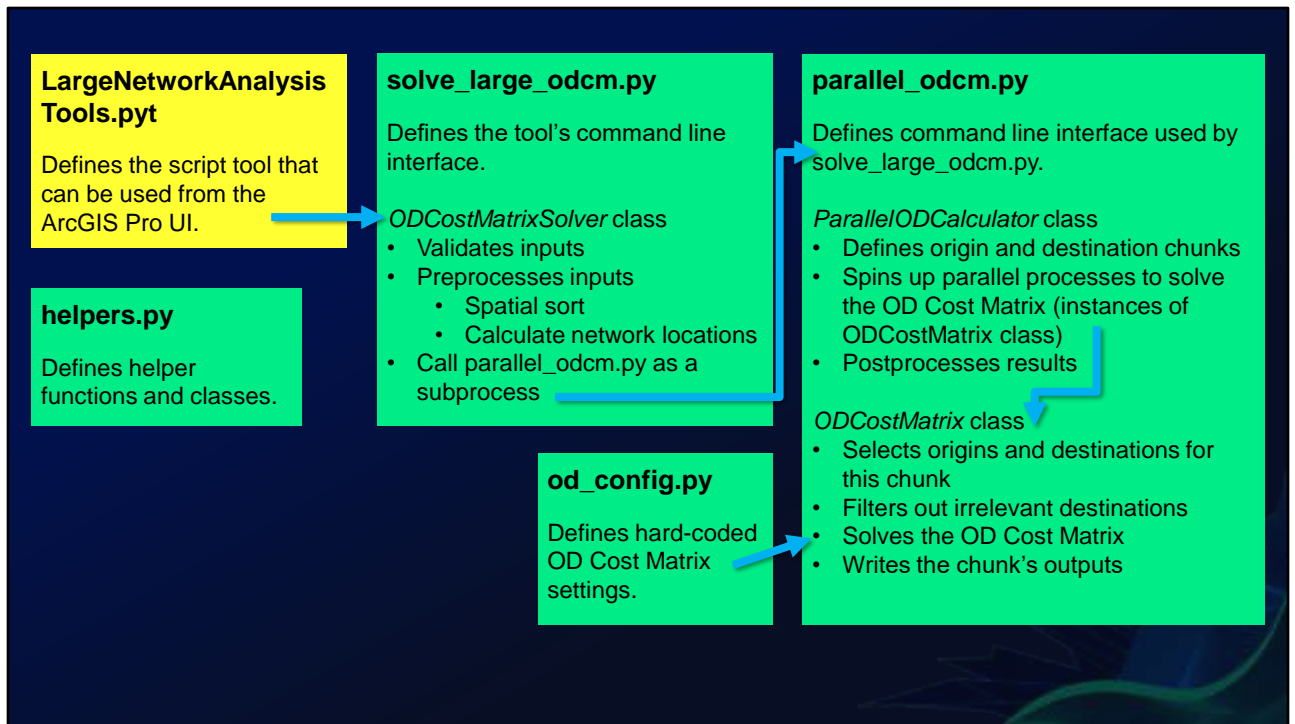
I'm going to show you some code in a moment, but first let me review the broad outline. The goal is to solve an OD Cost Matrix of any size using either local data or a service. The script itself is a bit complicated, but it contains a couple of key parts:

- There is a preprocessing part where the data is spatially sorted and the network locations are calculated.
- There is a solving section where the data is chunked into pieces and solved in parallel
- Finally, there is a post-processing section where the results are merged and certain special situations are handled.

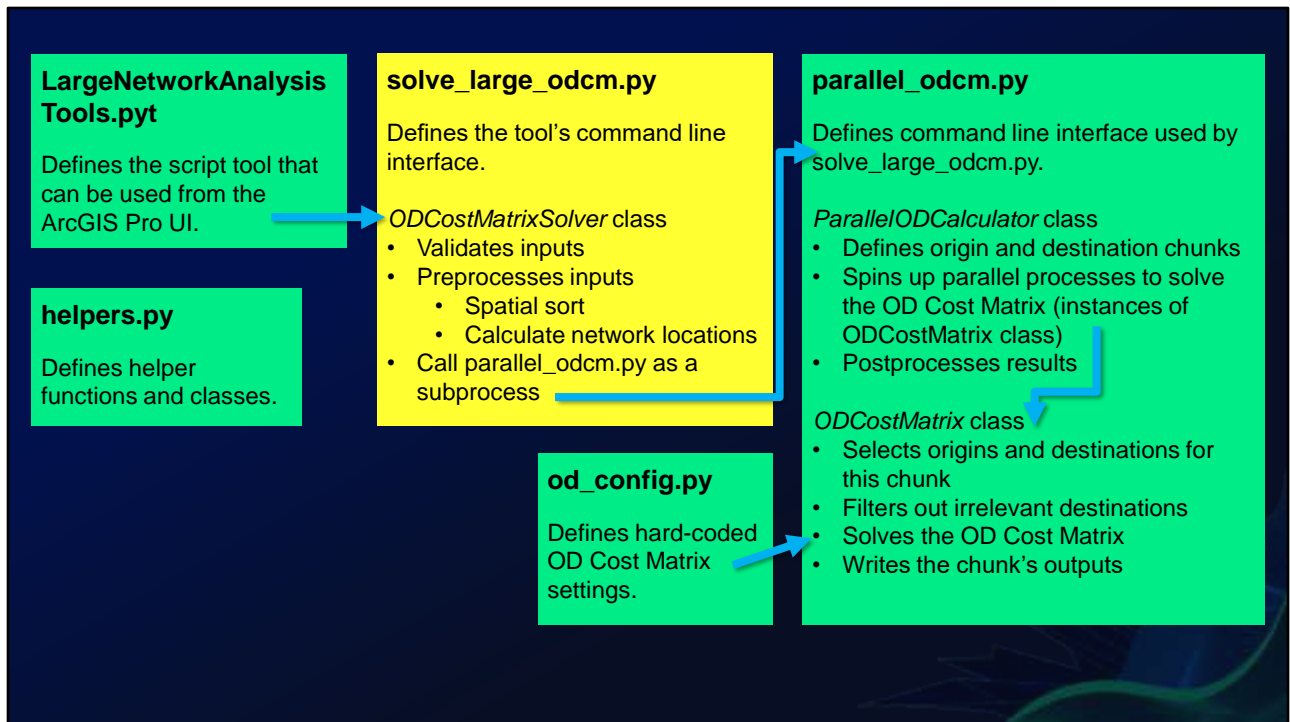
We know it's super hard to follow along with a code walk-through, so you can download the code from this link about look at it later. We've been careful to comment it thoroughly, so hopefully it should be pretty understandable if you open it up and try to modify it for your needs.



Here's a diagram showing how the code is structured.

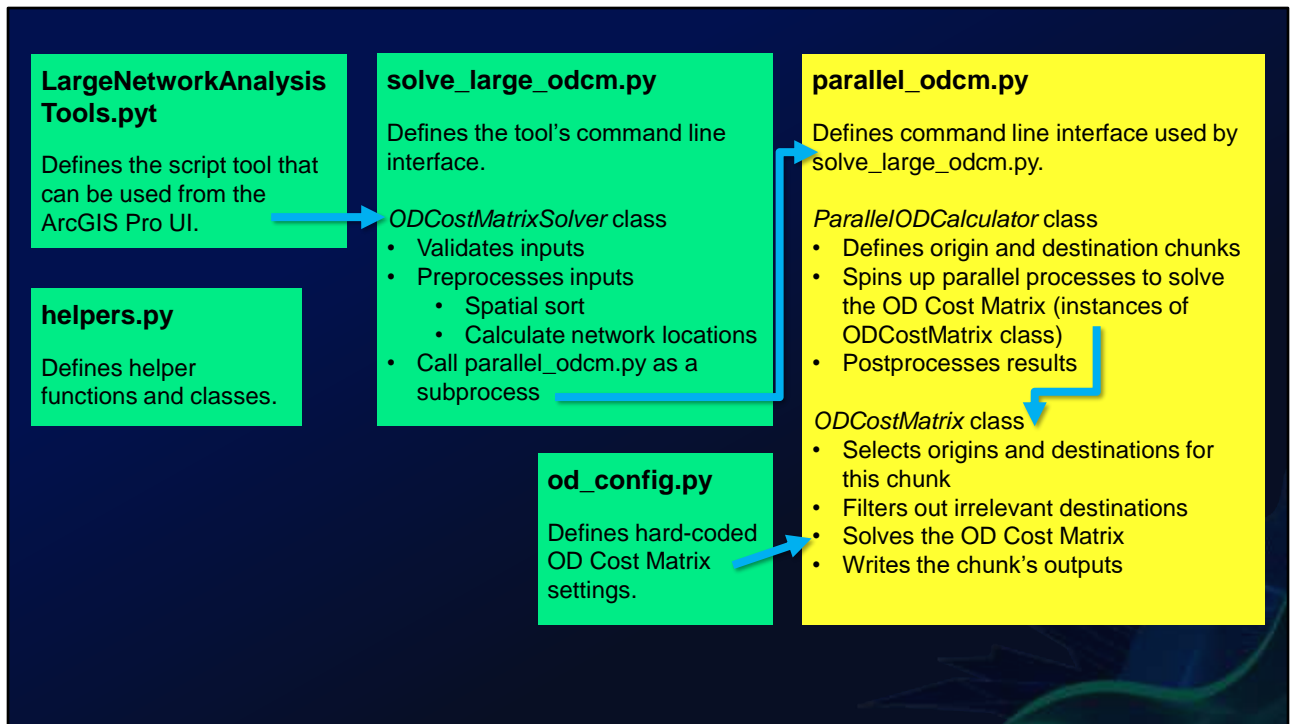


LargeNetworkAnalysisTools.pyt defines the script tool that can be used from the ArcGIS Pro UI.

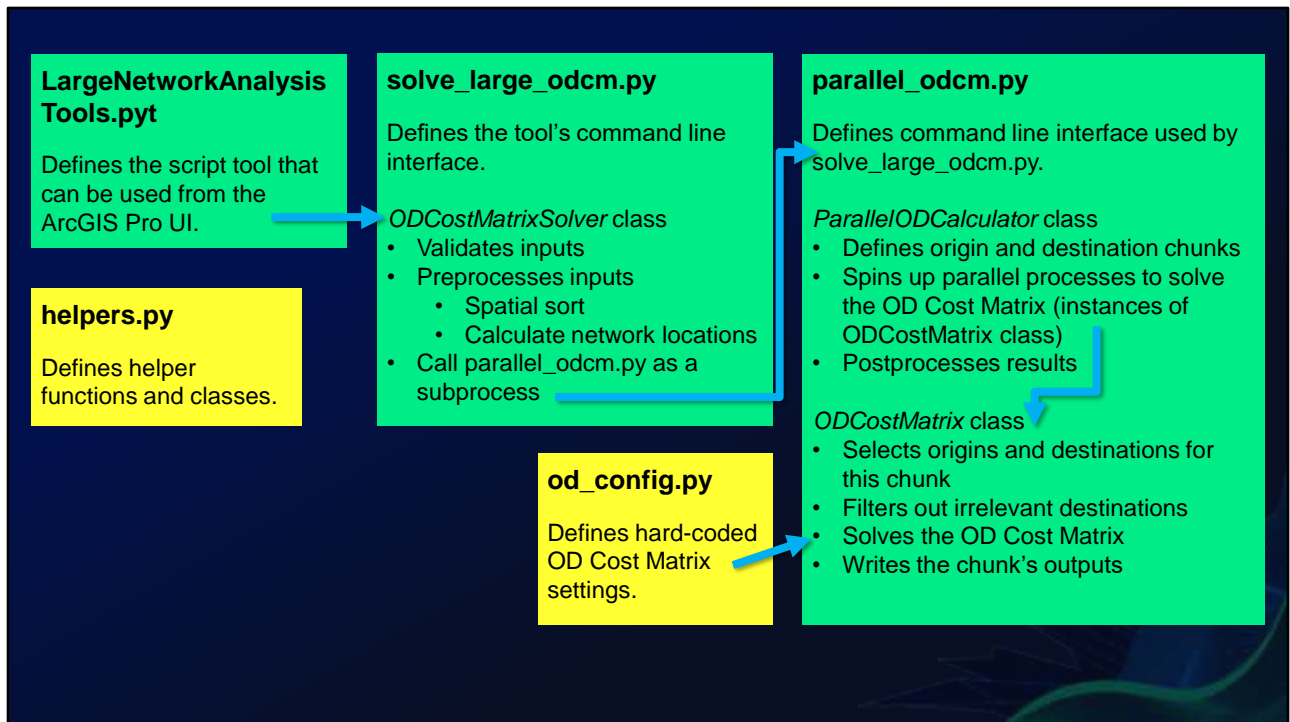


The script tool calls `solve_large_odcm.py`, which also has a command line interface if you prefer to interact with the tool from the command line. This script has a class called `ODCostMatrixSolver` that validates the inputs, preprocesses the inputs (spatial sorting, calculating network locations), and then calls the `parallel_odcm.py` script as a subprocess.

The reason it's using a subprocess instead of calling the module directly has to do with the script tool interface in ArcGIS Pro. We're going to use `concurrent.futures` to spin up parallel processes, and a script tool process can't do that directly. Basically it gets confused and tries to spin up parallel instances of ArcGIS Pro instead of python, so we get around this limitation by calling a subprocess and having the subprocess do the parallelization with `concurrent.futures`.



**parallel\_odcm.py** is where the real work is done. The **ParallelODCalculator** class defines the origin and destination chunks, spins up the parallel processes, and then collects and postprocesses the results. The **ODCostMatrix** class calculates the OD Cost Matrix for each chunk. This class selects the origins and destinations for the chunk, filters out irrelevant destinations, solves the OD Cost Matrix, and writes out the results for the chunk.



There is a file called `od_config.py` that holds hard-coded OD Cost Matrix settings. It's done this way so it's easy for you to change without digging through the code.

There's also a `helpers.py` file with some helper functions.



```

class ParallelOCCalculator():
    """Solves a large QD Cost Matrix by chunking the problem, solving in parallel, and combining results."""

    def __init__(self): # public: class-level-only locals, for non-arguments

    def _validate_settings(self):

    @staticmethod
    def _get_nid_ranges_for_inputs(nnodes, n, ncd, chunk_size):

    def solve_nid_in_parallel(self):
        """Solve the QD Cost Matrix in chunks and post-process the results."""
        # validate QD Cost Matrix settings. Essentially, create a dummy QDMatrix class instance and set up the
        # solver object to ensure this at least works. On this up front before spinning up a bunch of parallel processes
        # the optimized that one guarantees to all fail, while we're doing this, check our state the field name that
        # will represent costs in the output QD lines table. We'll use this in post processing.
        self.optimized_cost_field = self._validate_settings()

        # compute QD cost matrix in parallel
        completed_jobs = 0 # track the number of jobs completed so far to see if logging
        # use the concurrent.futures.ProcessPoolExecutor to spin up parallel processes that solve the QD cost matrices
        with futures.ProcessPoolExecutor(max_workers=self.n_processors) as executor:
            # Each parallel process calls the solve_nid_cost_matrix() function with the nid_inputs dictionary for the
            # given origin and destination NID ranges
            jobs = [executor.submit(solve_nid_cost_matrix, self.nid_inputs, range(r) for range in self.ranges)]
            # as each job is completed, add some logging information and store the results to post-process later
            for future in futures.as_completed(jobs):
                completed_jobs += 1
                logger.info(
                    f"Finished QD Cost Matrix calculation (completed_jobs) of {self.total_jobs}."
                )

```

Let's look at  
some code!

Okay, let's take a look at all this in the code. I will go through this fairly quickly, hitting only the major highlights. Again, this code is pretty complex, but it's available for you to download and use or customize for your own needs.

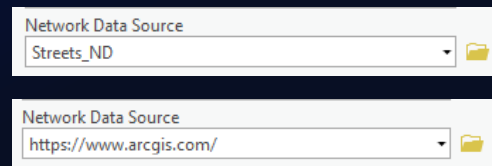
The background of the slide is a deep blue. It features several abstract elements: a series of small, light blue squares scattered in the upper left; a cluster of larger, semi-transparent blue squares in the upper right; a network of thin, light blue lines forming a web-like pattern in the middle right; and a large, wavy, light blue line that curves from the bottom right towards the center. The text "Working with services" is written in a white, sans-serif font on the left side.

## Working with services

[Switch presenters] Before we wrap up today, let's talk about a few additional considerations when working with services.

## Using a service as the network data source

- The code is the same!
- Use the portal URL as the Network Data Source
- Limit parallel processes to 4 for ArcGIS Online
- Cannot precalculate network locations



From a code perspective, solving your OD Cost Matrix analysis using a service is essentially the same as solving it using a local network dataset.

You just use the portal URL as the network data source in your solver object.

The script tool still runs on your local machine – preprocessing, postprocessing and solving parallel. Just that the OD solve itself is calculated by a service instead of your local machine.

When using ArcGIS Online services to solve, you need to limit the number of parallel processes to 4. To not overload the services and block other users from using them.

The service limits determine your chunking rules – how many origins and destinations you can solve at one time.

Recall, you also can't pre-calculate network locations.

## Which do I pick?

	Local network dataset	ArcGIS Online service	ArcGIS Enterprise service
Requires your own street data	Yes	No	Yes
Requires Network Analyst extension license	Yes	No	Yes (for Desktop and Server)
Requires service credits	No	Yes	No
Requires ArcGIS Enterprise	No	No	Yes
Number of concurrent processes	Number of CPU cores on your machine	Up to 4	
Analysis limits	Depends on your machine's memory	1000 origins x 1000 destinations	Configurable depending on memory resources of the server

So which approach do you use, a local network dataset or a service?

Service solves actually have two sub-options: ArcGIS Online or an ArcGIS Enterprise running on your own servers. Let's take a look at this table, which compares these three options.

If you don't have any street data, or you don't have the Network Analyst Extension license for ArcGIS Pro or ArcGIS Enterprise, using the ArcGIS Online services as your network data source is the only option.

It has good quality street data from around the world, and you don't need the Network Analyst extension to use it.

But ArcGIS Online service imposes limits on the size of the problem you can solve: 1000 origins and 1000 destinations in a single request.

This isn't necessarily a problem, since our sample script can chunk up the data within these limits. However, if you have a very large problem, you might need to use a large number of chunks. And you will use service credits which can add up for large problems. More on that in a moment.

If you *do* have your own street data and the Network Analyst Extension license, you can either

- solve using a local network dataset on a single machine in ArcGIS Pro
- or you can host your own routing service in ArcGIS Enterprise

If you have ArcGIS Enterprise, you can speed the problem up by running the process on a bank of servers.

## All about credits and service limits!

- ArcGIS Online services require service credits
  - Credit cost for each service
  - OD Cost matrix is 0.0005 credits **per input origin-destination pair**
    - 1000 origins x 1000 destinations = 1 million origin-destination pairs
    - 1 million pairs x 0.0005 credits = 500 credits
  - Cost is based on inputs, not outputs. **Reduce your problem size and sort your data first!**
- ArcGIS Online imposes problem size limits
  - ArcGIS Online is limited to 1000 origins and 1000 destinations per request
  - Use `arcpy.nax.GetWebToolInfo()` to retrieve tool limits in a script
  - Current service limits for ArcGIS Online: [OD Cost Matrix](#), [Route](#), [Closest Facility](#), [Service Area](#), [Location-Allocation](#), [Vehicle Routing Problem](#)

Limit Description	Limit Value
Maximum number of origins	1000
Maximum number of destinations	1000
Maximum number of barriers	1000

If you're working with ArcGIS Online, you have to worry about two things: credits and limits.

The ArcGIS Online services charge credits for each solve. For OD Cost Matrix, it costs 0.0005 credits *per input origin-destination pair*. 1000 x 1000 = 1 million pairs = 500 credits

So if you send in 1000 origins and 1000 destinations but you only want to find the one closest destination for each origin, it charges you for 1 million pairs, not 1000.

This is why it's especially important to reduce your problem size using as we have been describing here.

Make sure to estimate the cost of your analysis in advance to determine how many credits you'll need and determine if it's more cost effective to do this or to invest in your own street data.

Also the you need to consider service limits. The ArcGIS Online services limit for OD is 1000 origins and 1000 destinations. These limits determine the maximum chunk size you can use when breaking up your large problem.

The `arcpy.nax.GetWebToolInfo()` function can be used to retrieve tool limits from a service dynamically in a script.

# How to publish your own routing services to ArcGIS Enterprise

- Use the [Publish Routing Services](#) utility
- Special considerations for large problems:
  - Change minimum and maximum service instances to be equal to number of physical CPU cores on your server
  - Set a high value for service usage timeout
  - Set a high value for maximum records returned by server

If you're using ArcGIS Enterprise, you need to publish the OD Cost Matrix service. You can do that using the Publish Routing Services utility, which ships with ArcGIS Enterprise. You can learn more at the link in the slides. This tool takes in a network dataset and publishes the service in your ArcGIS Enterprise.

The tool publishes the service with default parameters, but if you're using the service to solve large problems, there are some settings you should check and configure:

- Number of service instances:
  - This depends on the number of physical cores you have on your server and how many servers you have. If your servers have four physical cores, you should change the number to 4. This makes sure you have 4 service instances running on the server simultaneously.
- Execution time for a request:
  - By default, the setting is set to 10 minutes, so if a request takes longer than that, it will time out and return an error. If you're solving a large OD, the request can take longer than that, so I recommend you set this to a higher number, like an hour or three hours.
- Number of features returned from the service:
  - Consider the number of output records likely to be produced by each chunk and set this number accordingly. By default, it's set to 1 million, and the service will return an error if it tries to return more than 1 million records.

The background of the slide is a deep blue. On the left, there are several small, faint, light blue squares scattered across the space. On the right side, there is a large, curved, wavy pattern made of many thin, parallel blue lines. In the center-right area, there are several overlapping, semi-transparent blue squares of different sizes. One of these squares contains a faint, glowing, yellowish-green pattern that looks like a network or a map. Another square below it shows a similar pattern with more detail, including some darker areas.

## Wrap up

Okay, everyone. That concludes our presentation today.

## Wrap-up

- Reduce problem size
- Eliminate irrelevant data
- Chunk data
- Spatially sort data
- Solve in parallel
- Pre-calculate network location fields
- Use network dataset layer
- \*\*Update to ArcGIS Pro 2.9 or higher

Code and slides:

**<https://github.com/Esri/large-network-analysis-tools>**

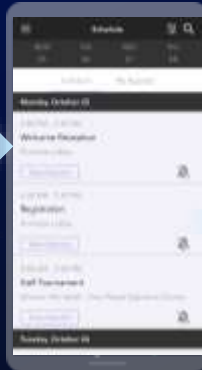


## Please Share Your Feedback in the App

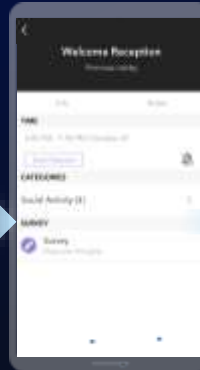
Download the Esri Events app and find your event



Select the session you attended



Scroll down to "Survey"



Log in to access the survey

