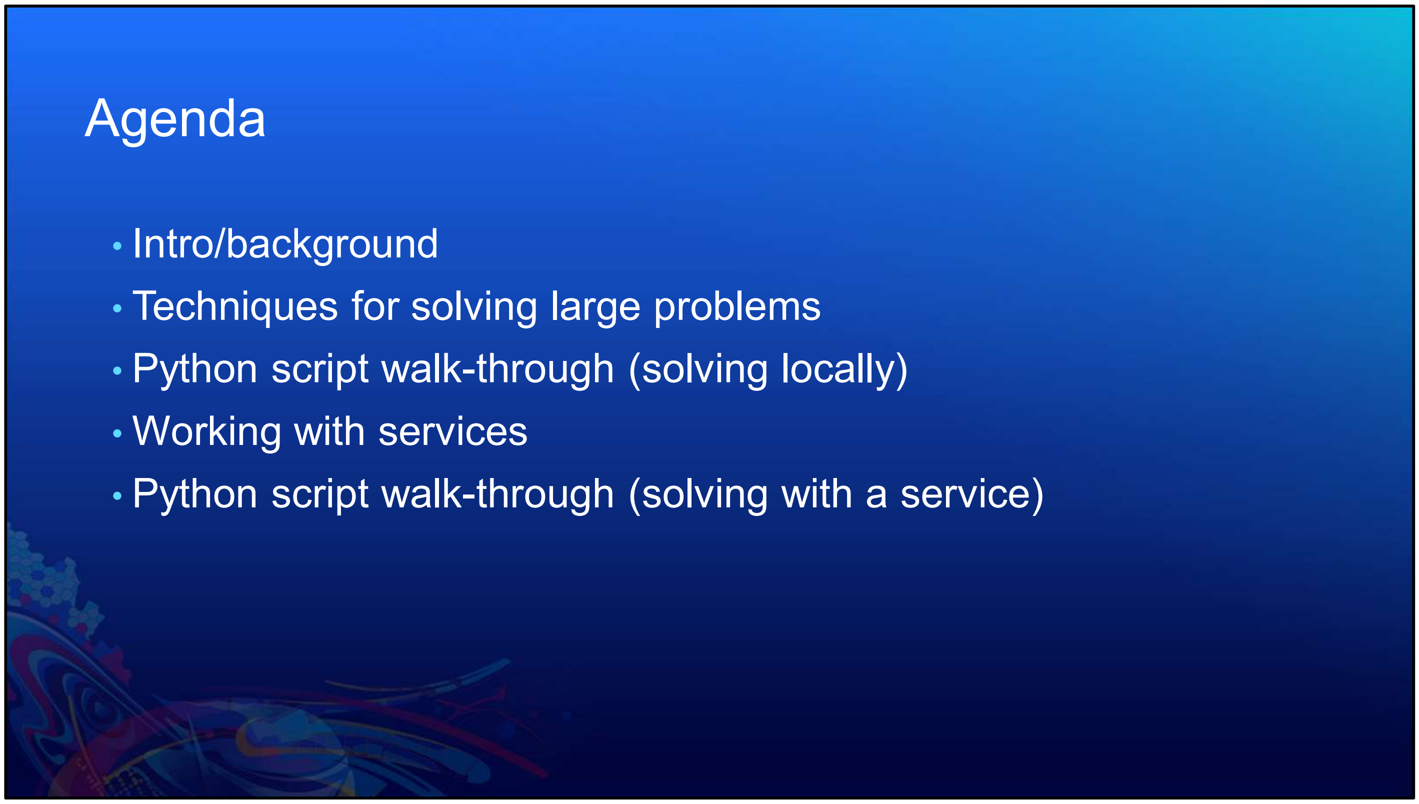
esri

ArcPy: Solving large transportation analysis problems

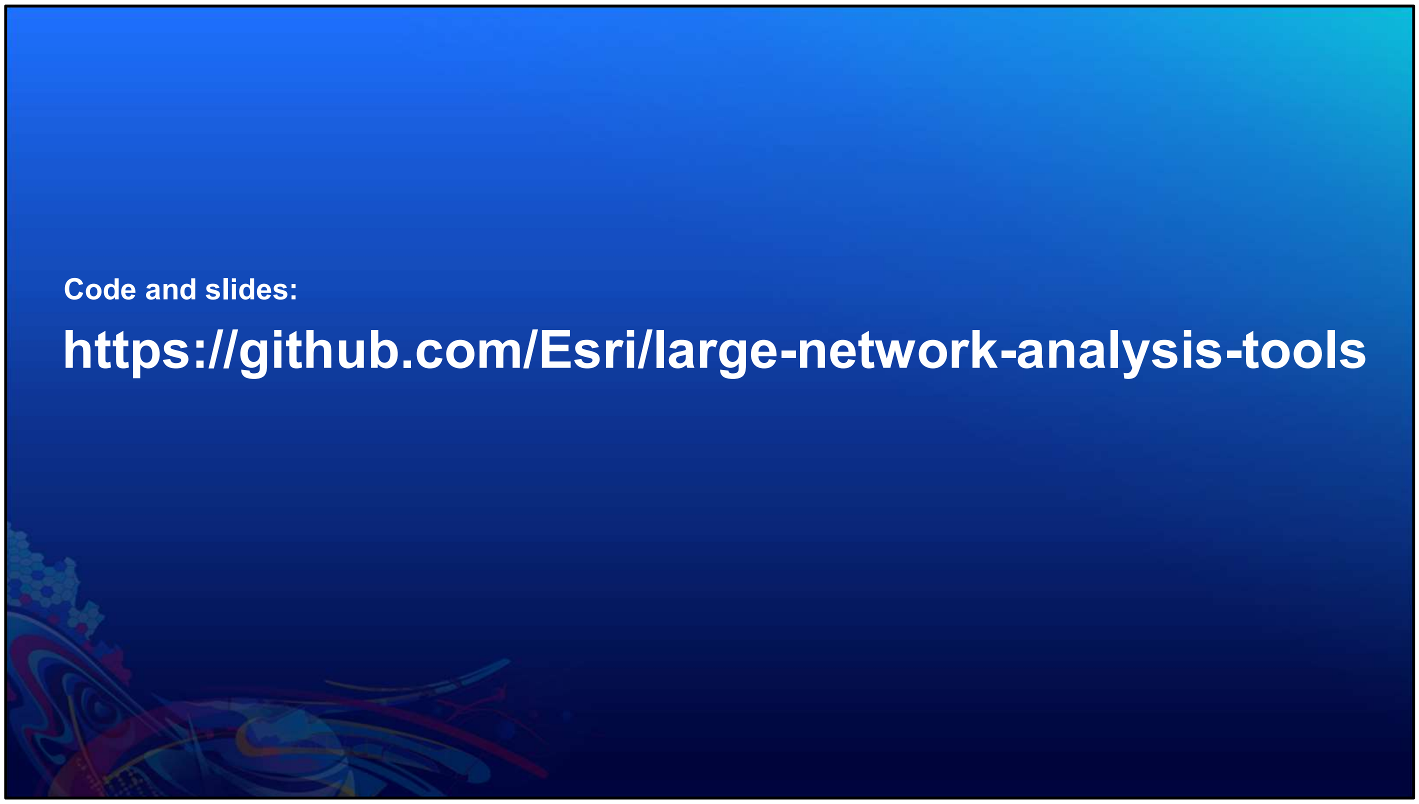
Deelesh Mandloi & Melinda Morang

2021 ESRI DEVELOPER SUMMIT



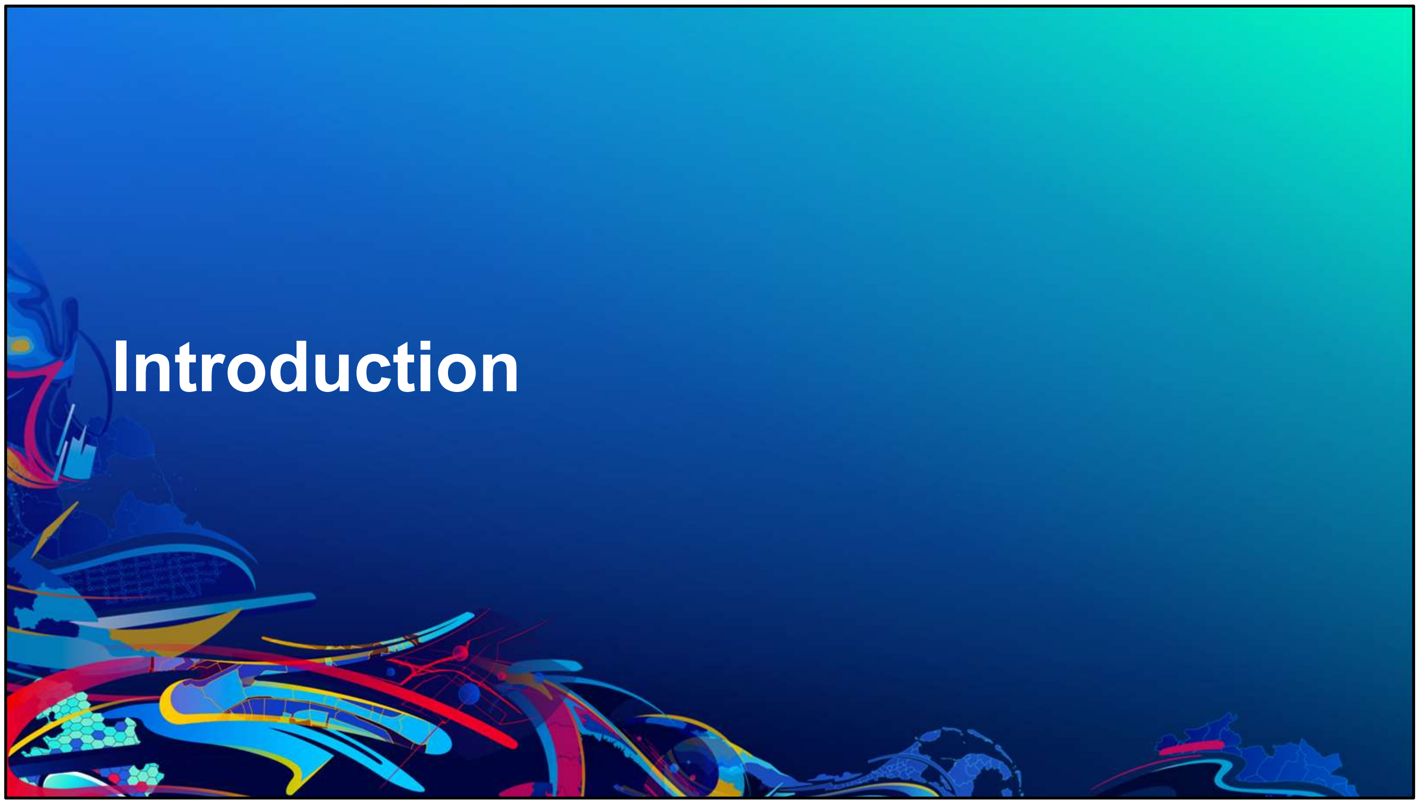
Agenda

- Intro/background
- Techniques for solving large problems
- Python script walk-through (solving locally)
- Working with services
- Python script walk-through (solving with a service)

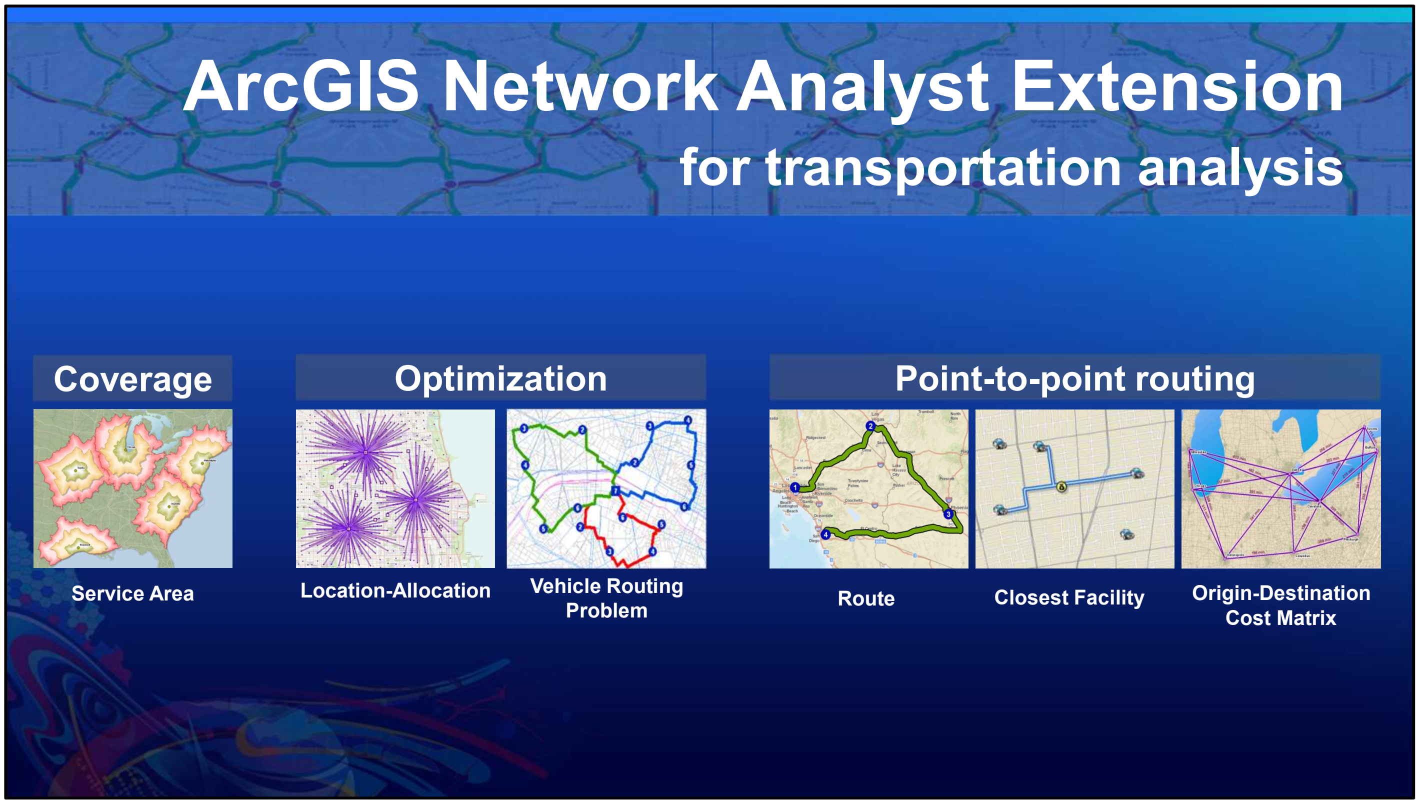

Code and slides:

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

We realize that it can be quite difficult to follow along with code in a presentation like this, so 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

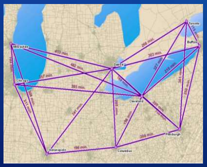
Location-Allocation

Vehicle Routing Problem

Point-to-point routing

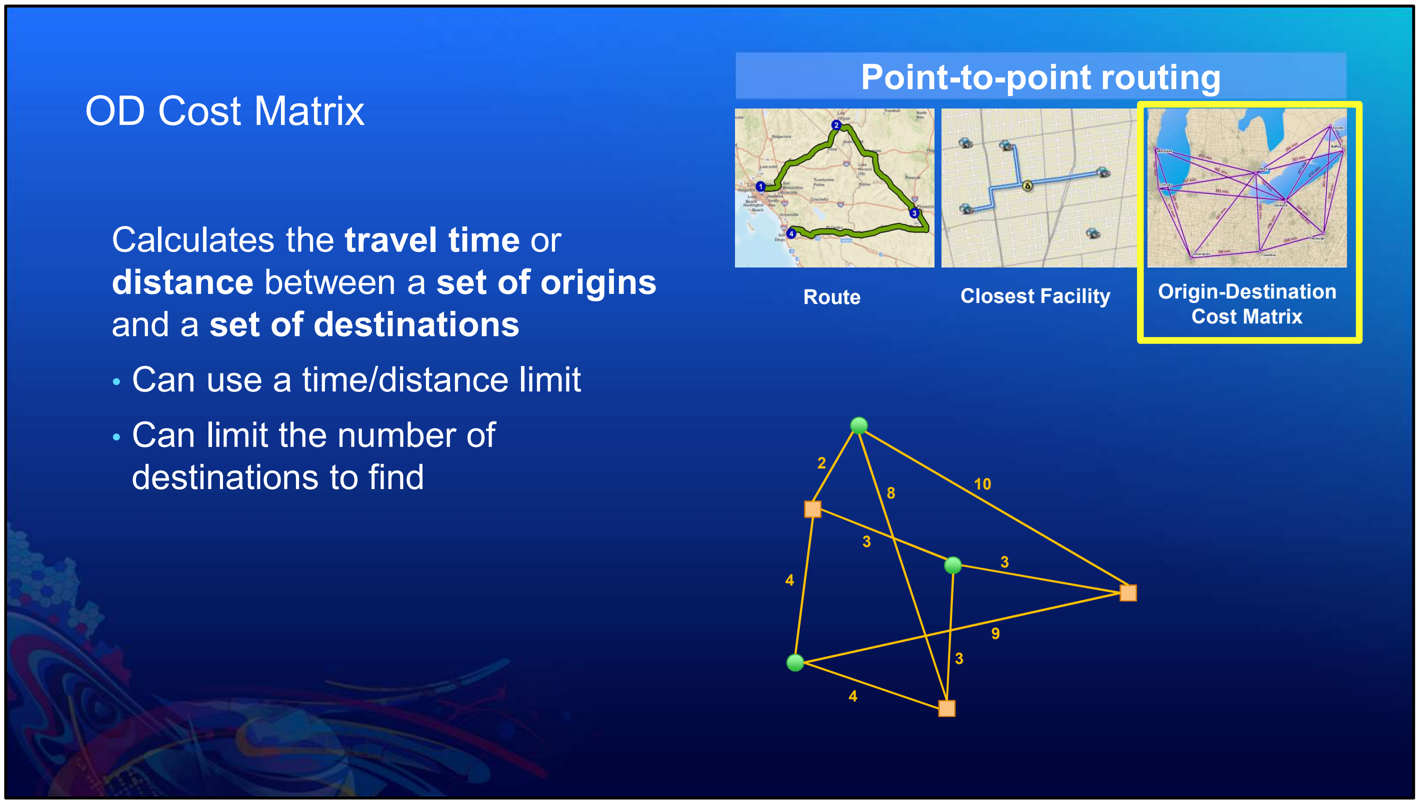
Route

Closest Facility



Origin-Destination Cost Matrix

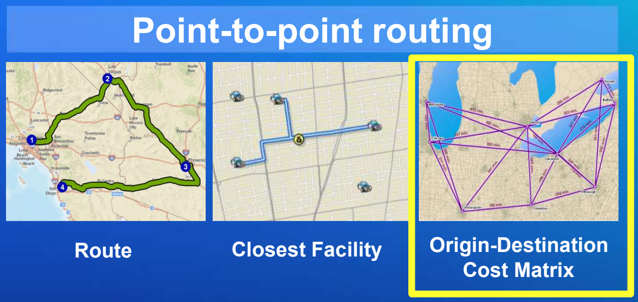
The ArcGIS Network Analyst Extension is the part of the software that’s for solving transportation analysis problems. It has tools for assessing the coverage of different facilities, for optimizing things like site location and fleets of vehicles, and for point-to-point routing, such as “How do I get from point A to point B?”, “What’s the Closest Facility”, and for solving origin-destination cost matrices.

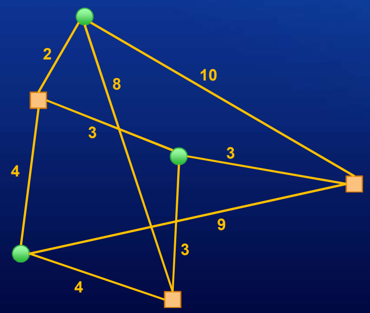


OD Cost Matrix

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

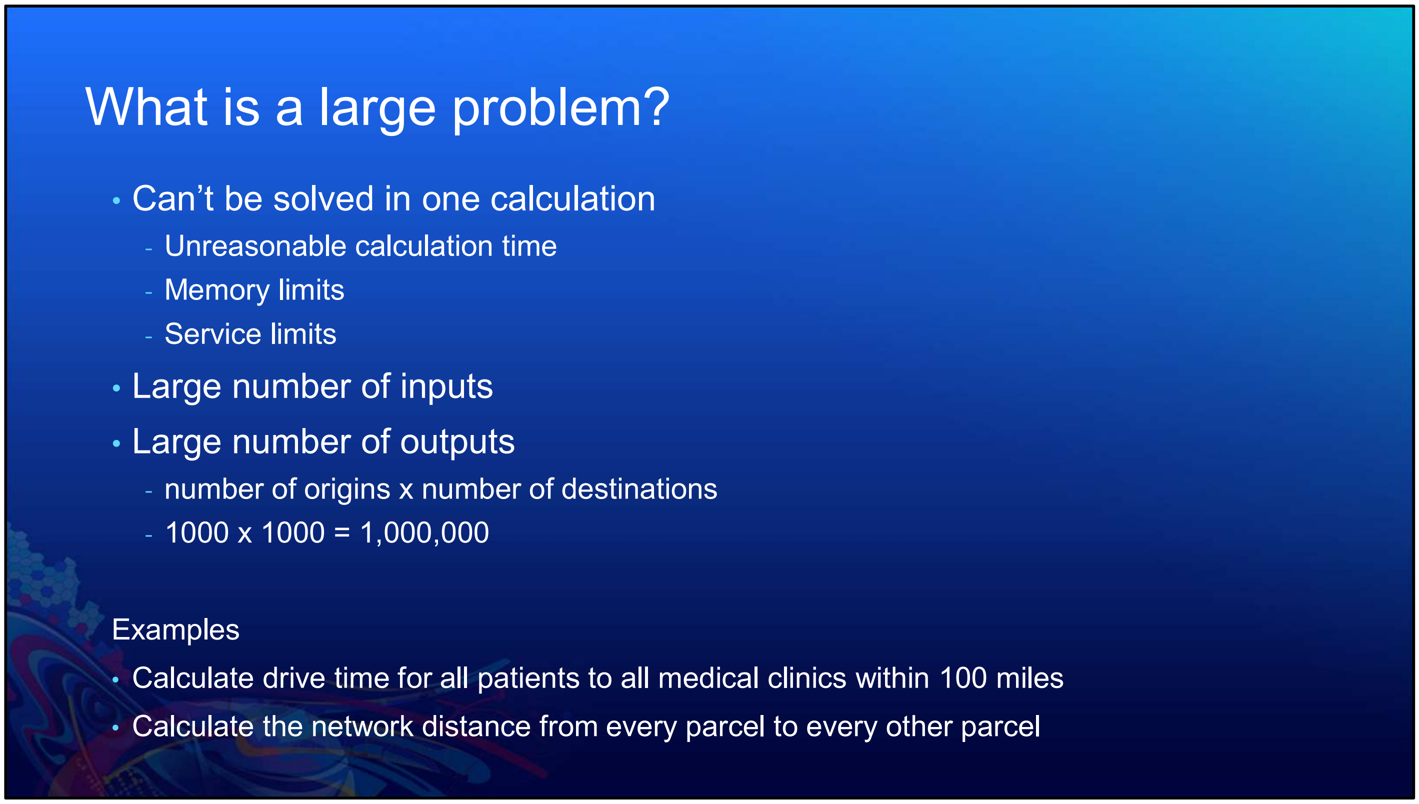
- Can use a time/distance limit
- Can limit the number of destinations to find





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?”, for example.



What is a large problem?

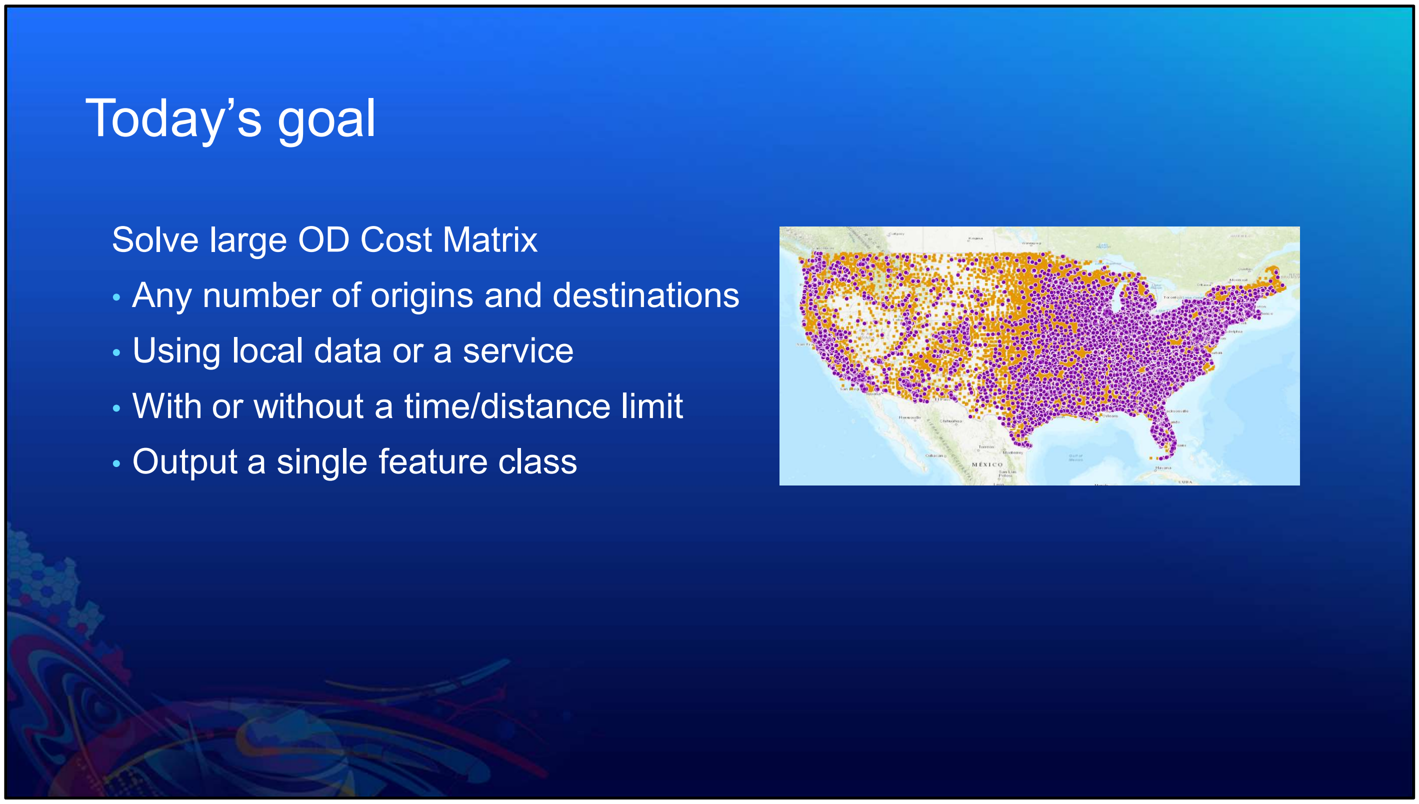
- Can't be solved in one calculation
  - Unreasonable calculation time
  - Memory limits
  - Service limits
- Large number of inputs
- Large number of outputs
  - number of origins x number of destinations
  - 1000 x 1000 = 1,000,000

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, maybe because it would involve an unreasonably large calculation time, 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 be large because the output is large. For example, with the OD Cost Matrix, the result comes back as a table that includes the number of origins times the number of destinations. If you have 1000 origins and 1000 destinations, your output has a million records in it, so the output can sometimes be the large part.

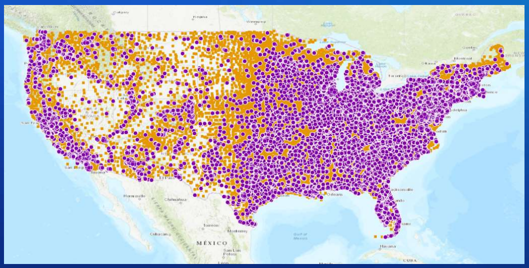
Some examples of a large problem include…



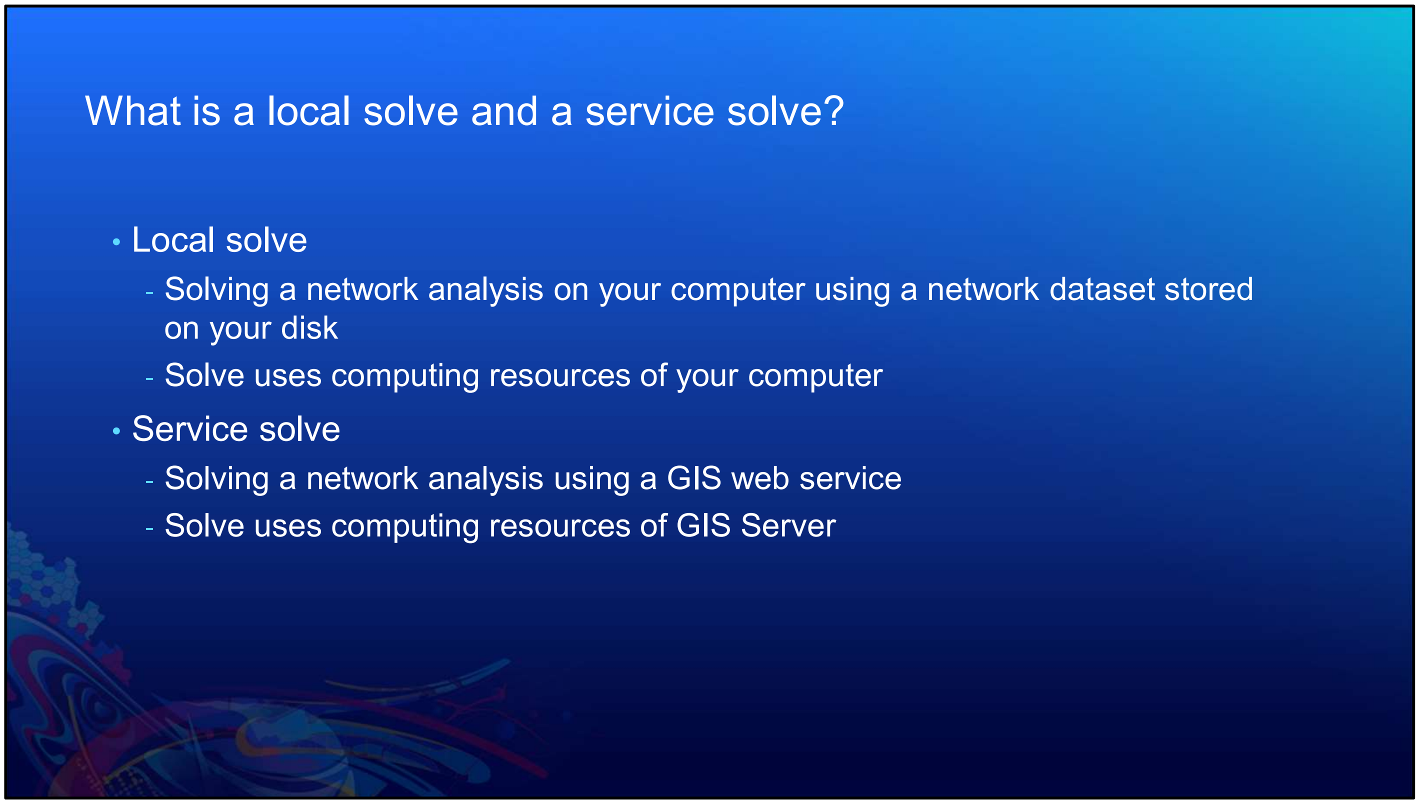
Today's goal

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



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



What is a local solve and a service solve?

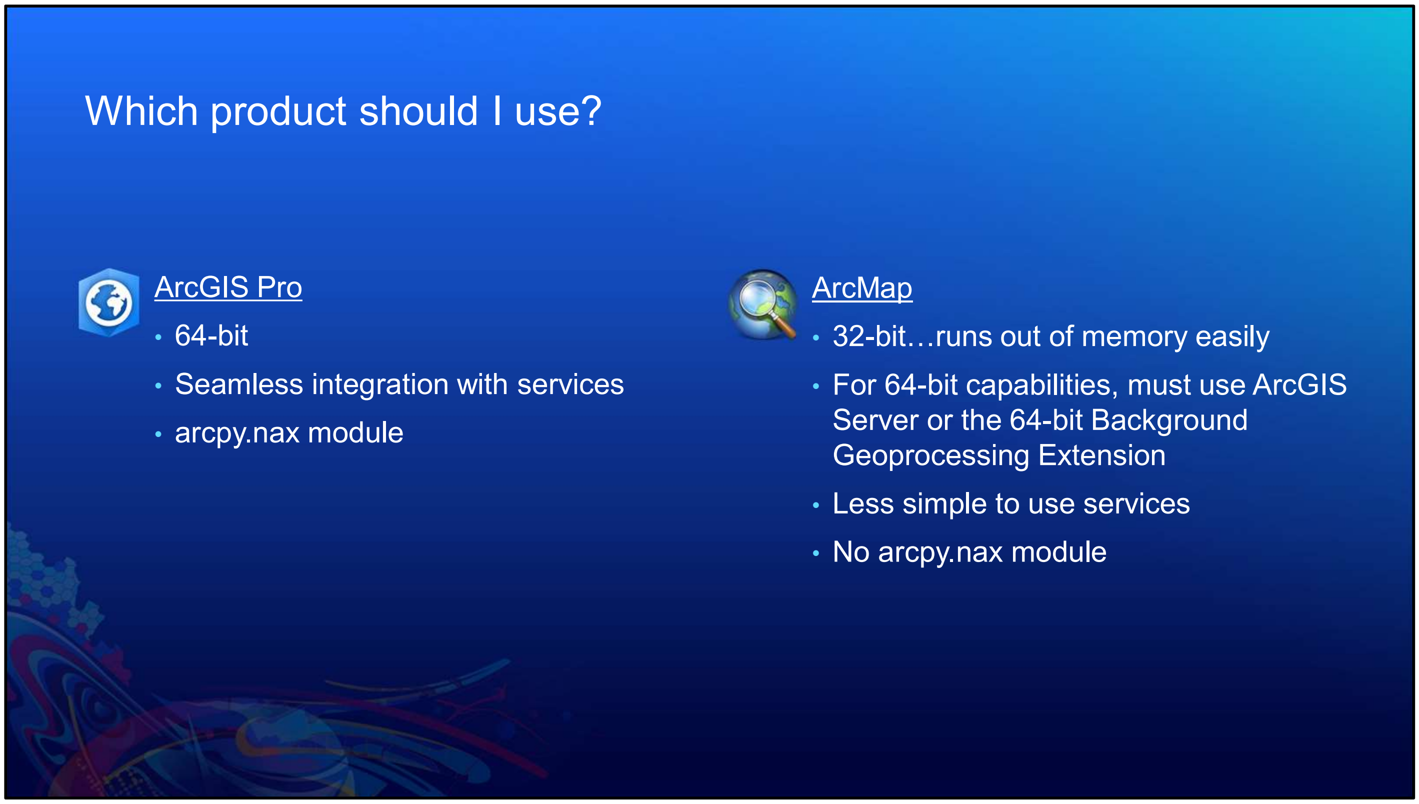
- Local solve
  - Solving a network analysis on your computer using a network dataset stored on your disk
  - Solve uses computing resources of your computer
- Service solve
  - Solving a network analysis using a GIS web service
  - Solve uses computing resources of GIS Server

I've mentioned local solves vs services a couple of times. What am I talking about?

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 a bit more detail later.



Which product should I use?

ArcGIS Pro
- 64-bit
- Seamless integration with services
- arcpy.nax module

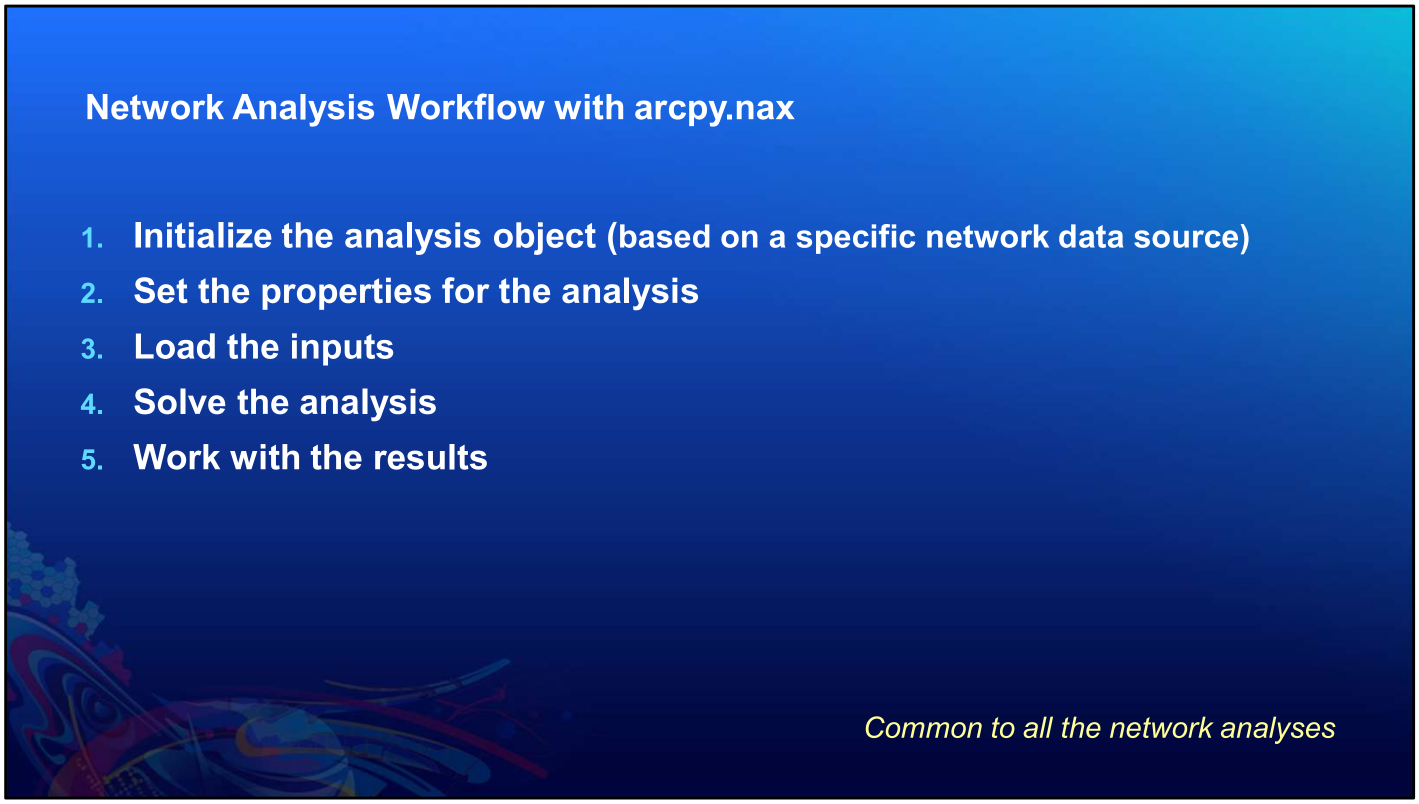
ArcMap
- 32-bit…runs out of memory easily
- For 64-bit capabilities, must use ArcGIS Server or the 64-bit Background Geoprocessing Extension
- Less simple to use services
- No arcpy.nax module

Which product should I be using? ArcGIS Pro or ArcMap? We recommend ArcGIS Pro for solving large problems.

Pro: It's a 64-bit application, so it can use the full memory resources of your own machine. You can solve problems in Pro that might have caused memory issues in ArcMap. It also has a much more seamless integration with services. ArcGIS Pro also has the arcpy.nax module, which provides a very nice clean and fast set of classes and methods for doing network analysis that is easier and less clunky than using all the network analysis layers and geoprocessing tools that you need in ArcMap.

ArcMap, on the other hand, is a 32-bit application, so it tends to run out of memory. It can't use the full memory resources of your own machine. For 64-bit capabilities, you have to install the 64-bit Background Geoprocessing Extension or use ArcGIS Server. It's much less simple to use with services and doesn't have the new arcpy.nax module.

For the rest of this presentation, we'll be showing code designed to work with ArcGIS Pro.



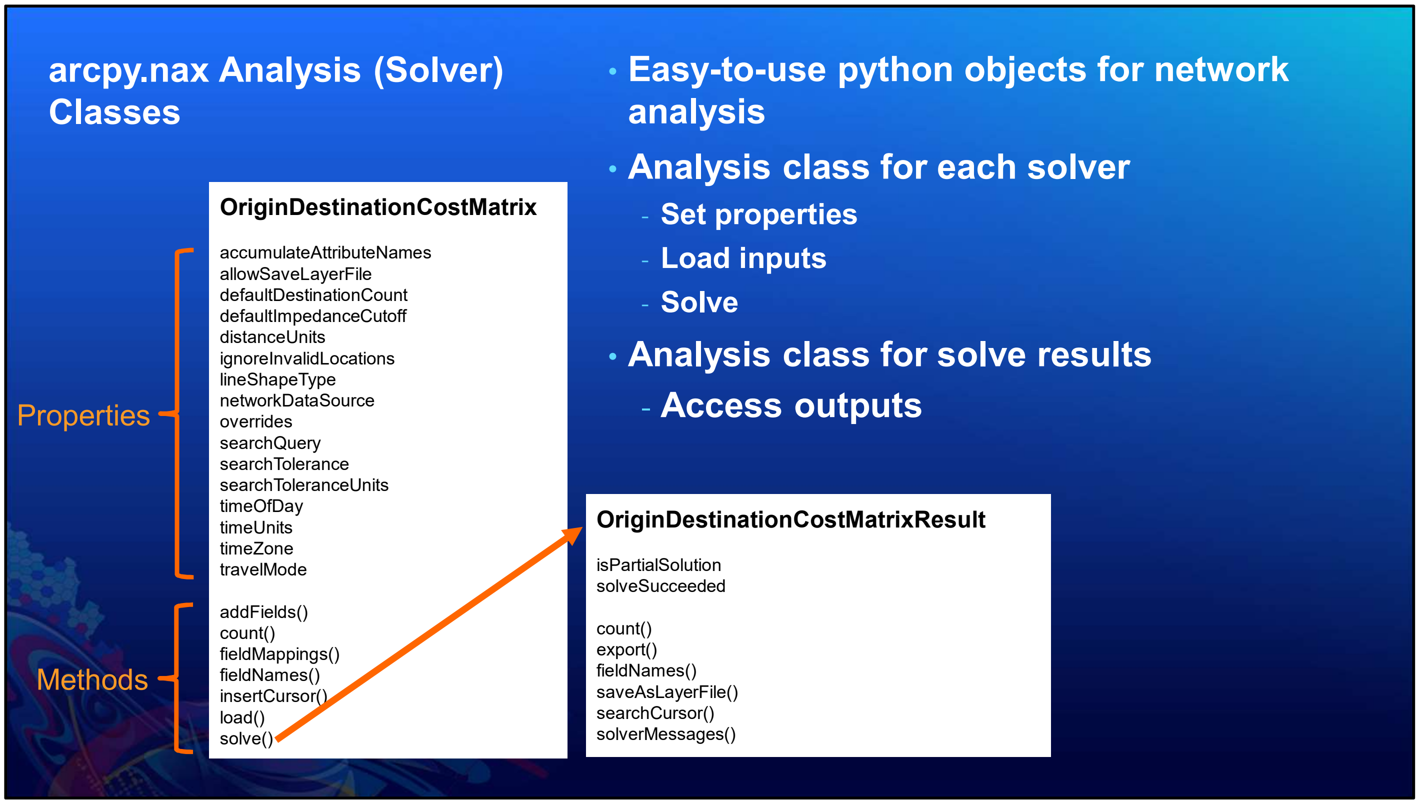
Network Analysis Workflow with arcpy.nax

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

As I just mentioned, 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 want to use for performing a network analysis, such as calculating an OD Cost Matrix.



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

OriginDestinationCostMatrix

accumulateAttributeNames
allowSaveLayerFile
defaultDestinationCount
defaultImpedanceCutoff
distanceUnits
ignoreInvalidLocations
lineShapeType
networkDataSource
overrides
searchQuery
searchTolerance
searchToleranceUnits
timeOfDay
timeUnits
timeZone
travelMode

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

OriginDestinationCostMatrixResult

isPartialSolution
solveSucceeded

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

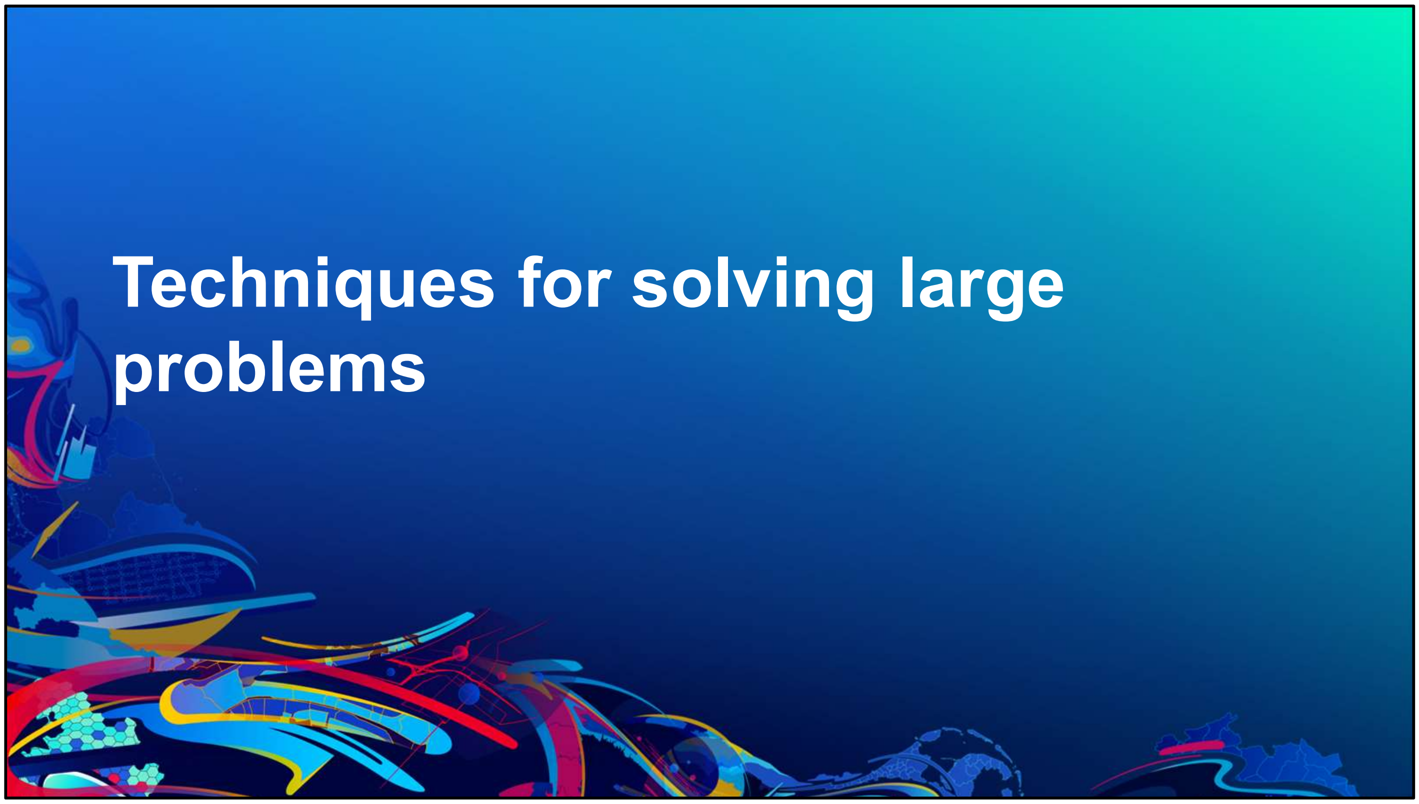
Each type of network analysis, or solver, has its own python class.

When you instantiate the solver object, you need to set the network data source. This can be either a network dataset or a URL to a service, such as ArcGIS Online or a service running on your own portal.

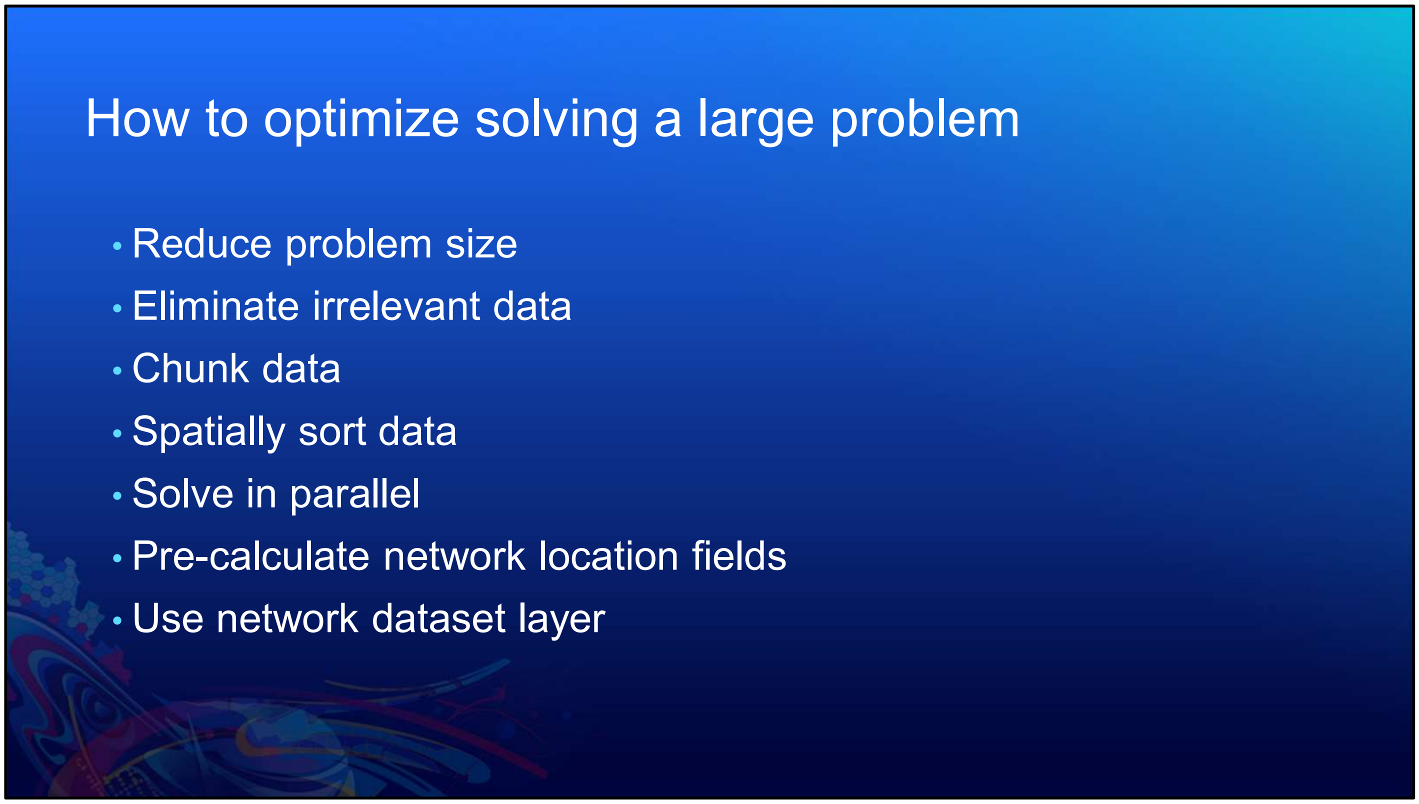
Once you instantiate the solver object class, you can set the various analysis properties. You can see the properties for the OD Cost Matrix object here, including the travel mode to use for the analysis, and the defaultDestinationCount and defaultImpedanceCutoff.

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.

The scripts we're going to show today will make use of these solver objects.



Techniques for solving large problems

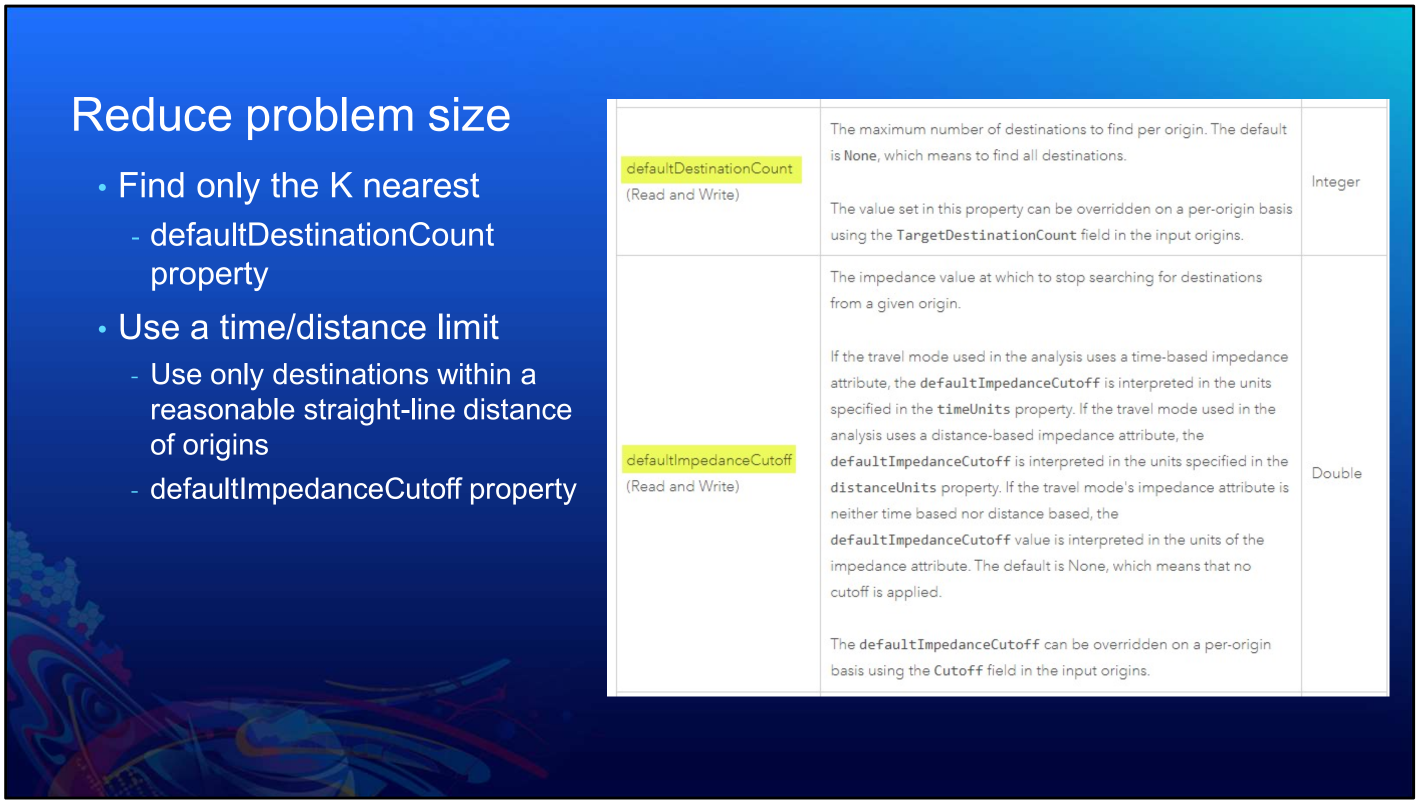


How to optimize solving a large problem

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

Some techniques include…

I'm going to go through each one of these in more detail.



Reduce problem size

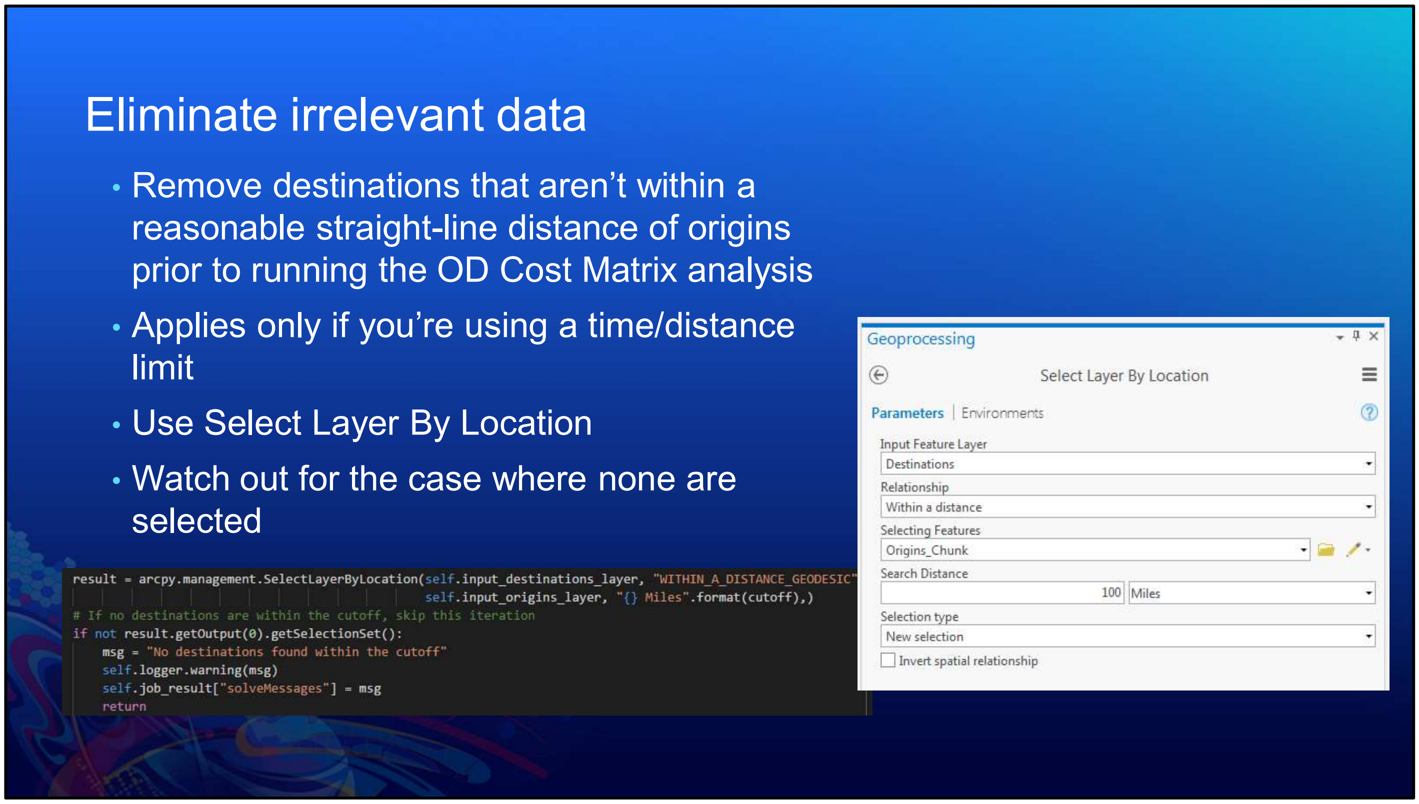
- Find only the K nearest
  - defaultDestinationCount property
- Use a time/distance limit
  - Use only destinations within a reasonable straight-line distance of origins
  - defaultImpedanceCutoff property

defaultDestinationCount (Read and Write)	The maximum number of destinations to find per origin. The default is None, which means to find all destinations. The value set in this property can be overridden on a per-origin basis using the `TargetDestinationCount` field in the input origins.	Integer
defaultImpedanceCutoff (Read and Write)	The impedance value at which to stop searching for destinations from a given origin. If the travel mode used in the analysis uses a time-based impedance attribute, the `defaultImpedanceCutoff` is interpreted in the units specified in the `timeUnits` property. If the travel mode used in the analysis uses a distance-based impedance attribute, the `defaultImpedanceCutoff` is interpreted in the units specified in the `distanceUnits` property. If the travel mode's impedance attribute is neither time based nor distance based, the `defaultImpedanceCutoff` value is interpreted in the units of the impedance attribute. The default is None, which means that no cutoff is applied. The `defaultImpedanceCutoff` can be overridden on a per-origin basis using the `Cutoff` field in the input origins.	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? The OD Cost Matrix tool has the ability to limit the number of destinations to find. You can use the defaultDestinationCount property on the OD Cost Matrix solver object.

You can also 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 of the OD Cost Matrix solver object to set a limit like 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

```
result = arcpy.management.SelectLayerByLocation(self.input_destinations_layer, "WITHIN_A_DISTANCE_GEODESIC"
                                                 self.input_origins_layer, "{} Miles".format(cutoff),)
# If no destinations are within the cutoff, skip this iteration
if not result.getOutput(0).getSelectionSet():
    msg = "No destinations found within the cutoff"
    self.logger.warning(msg)
    self.job_result["solveMessages"] = msg
    return
```

Geoprocessing

Select Layer By Location

Parameters | Environments

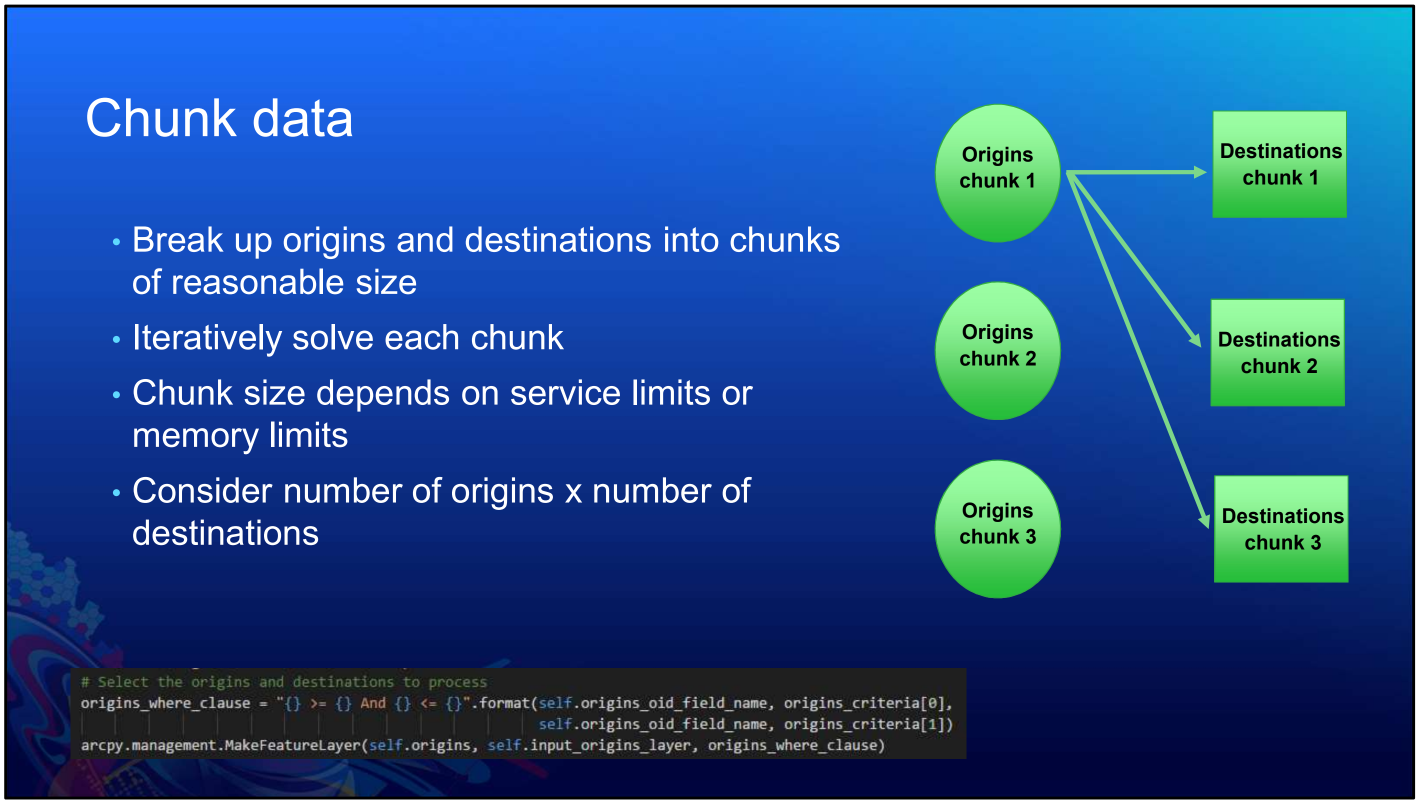
Input Feature Layer: Destinations
Relationship: Within a distance
Selecting Features: Origins_Chunk
Search Distance: 100 Miles
Selection type: New selection
☐ Invert spatial relationship

Another thing to do is to eliminate irrelevant data. This is related to reducing the overall problem size, and this 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. Consequently, you can first do a quick straight-line distance calculation to eliminate all destinations outside of your cutoff, maybe using some small additional buffer just to be on the safe side.

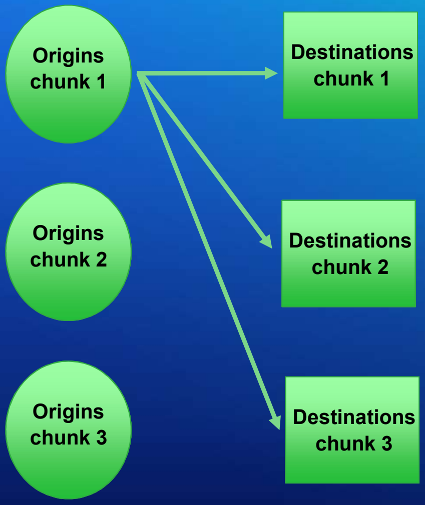
You can do this using the Select Layer By Location tool, which selects only the destinations within the specified straight-line distance of any origin.

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

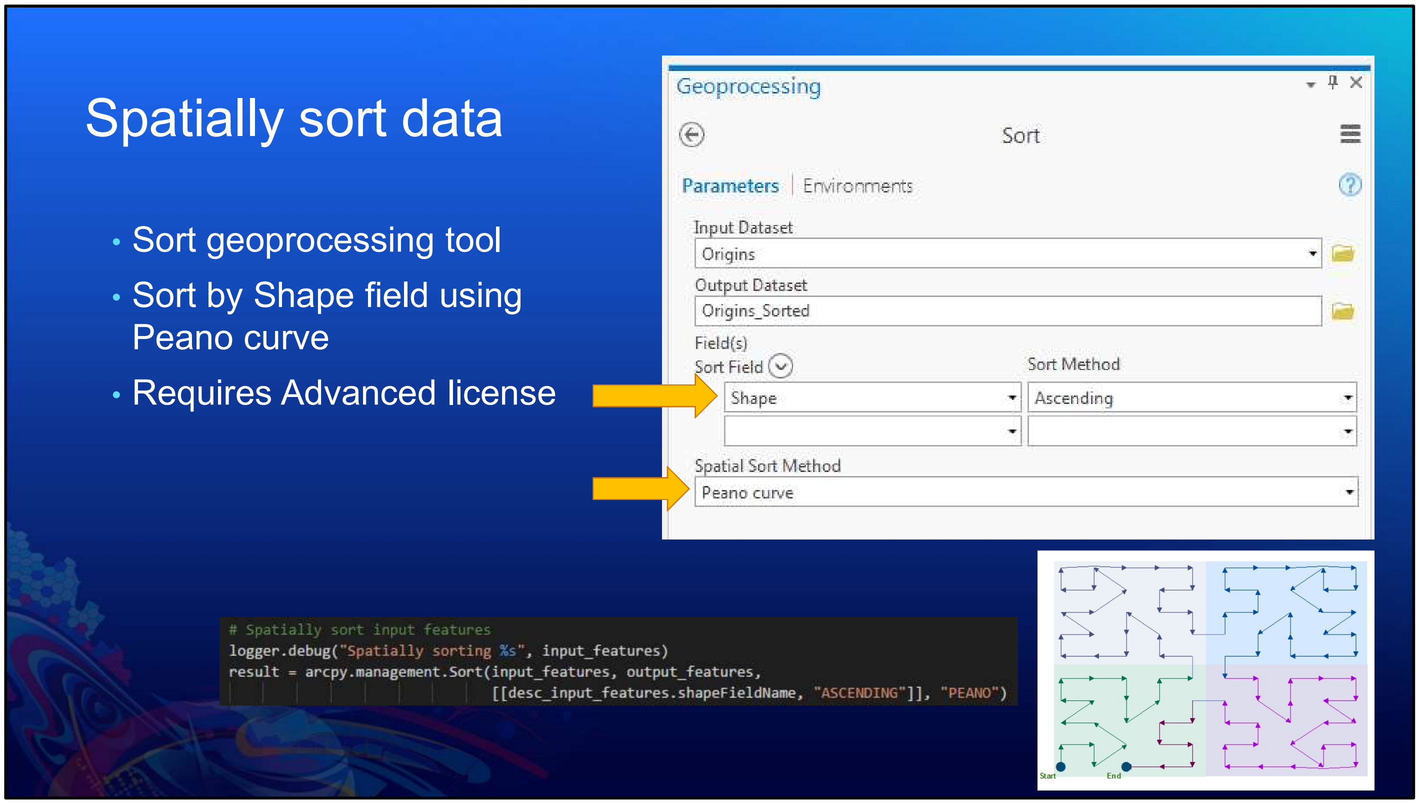


```
# Select the origins and destinations to process
origins_where_clause = "{} >= {} And {} <= {}".format(self.origins_oid_field_name, origins_criteria[0],
                                                      self.origins_oid_field_name, origins_criteria[1])
arcpy.management.MakeFeatureLayer(self.origins,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

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 want to 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 move on to the next chunk of origins and run it against each chunk of destinations.

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. Some services, like ArcGIS Online, have limits to the number of inputs you can pass in.

Don't forget to consider the number of origins times the number of destinations, or more generally consider the size of the output that will be returned by these chunks.



Spatially sort data

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

Geoprocessing

Sort

Parameters | Environments

Input Dataset: Origins

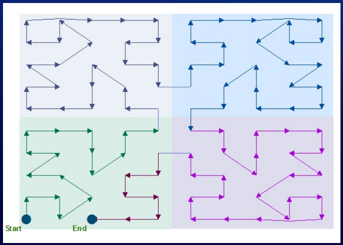
Output Dataset: Origins_Sorted

Field(s)

Sort Field	Sort Method
Shape	Ascending

Spatial Sort Method: Peano curve

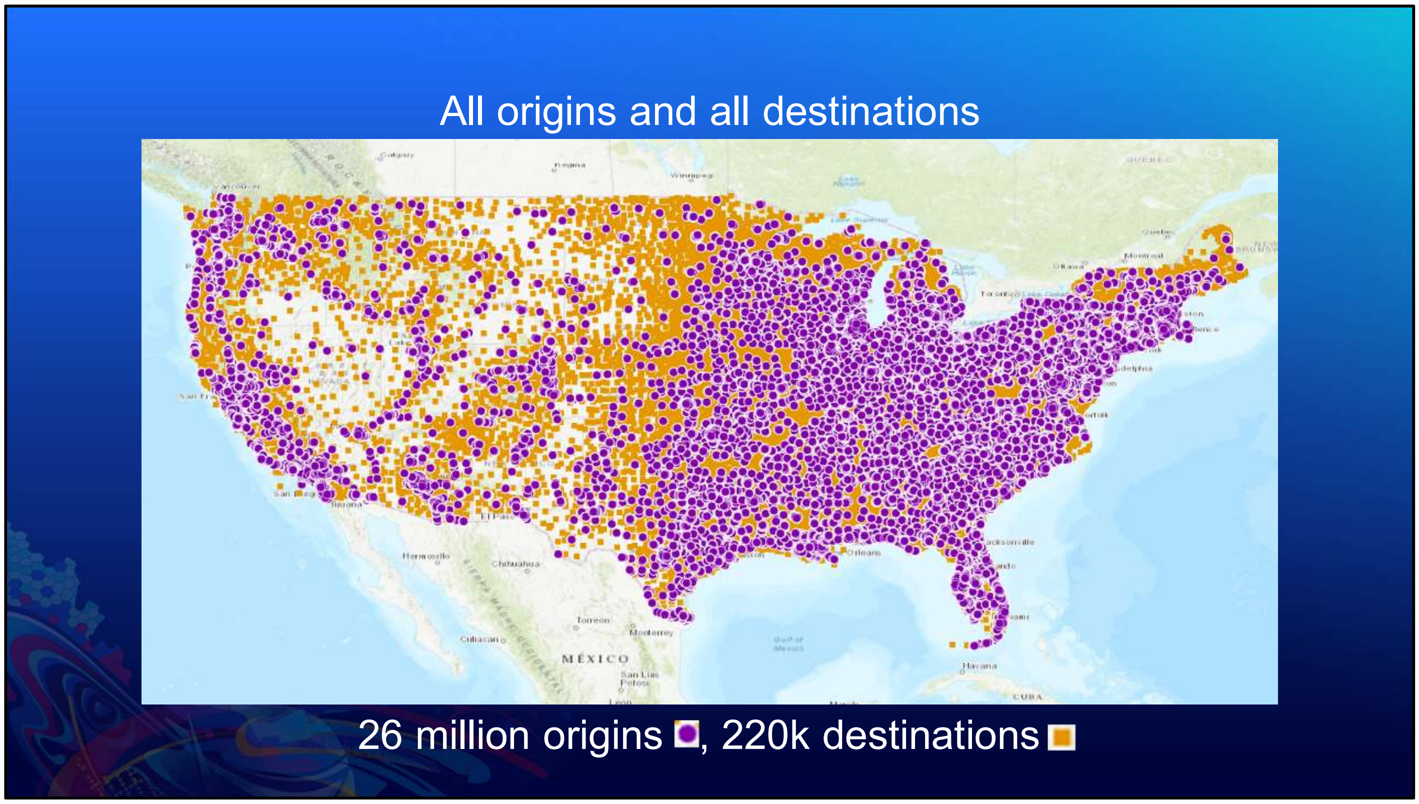
```
# Spatially sort input features
logger.debug("Spatially sorting %s", input_features)
result = arcpy.management.Sort(input_features, output_features,
                               [[desc_input_features.shapeFieldName, "ASCENDING"]], "PEANO")
```



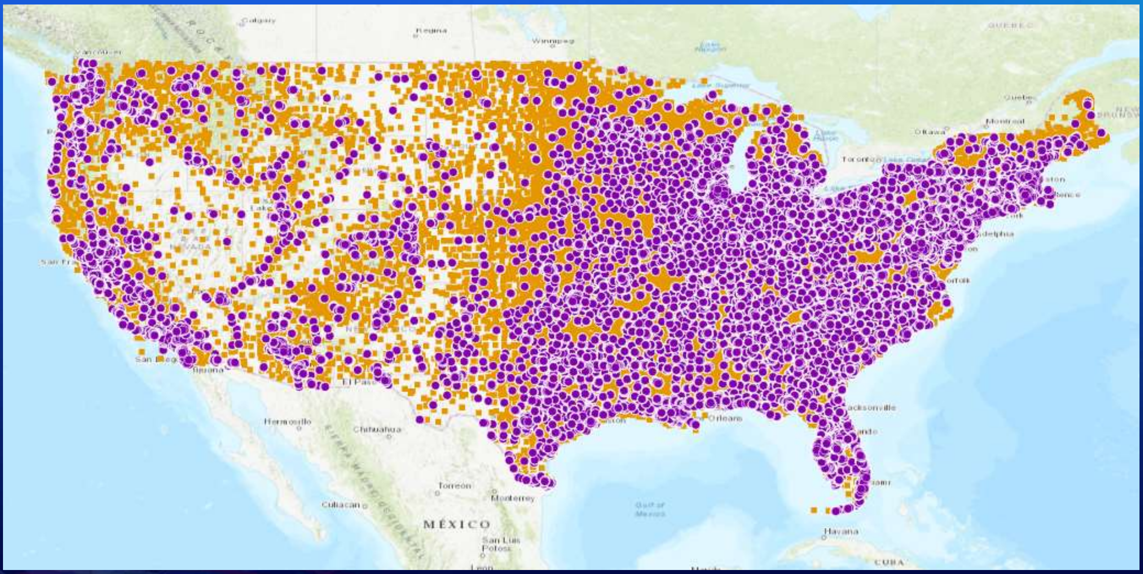
Something you can do to improve your chunking is to spatially sort your data. This allows you to make smarter chunks that will overall reduce the number of iterations you need.

You can use the Sort geoprocessing tool to spatially sort data by selecting the Shape field as the Sort Field and using the Peano curve method. This method makes the points all nice and clustered as it goes through the dataset.

One thing to note is that this spatial sort option requires the Advanced license.



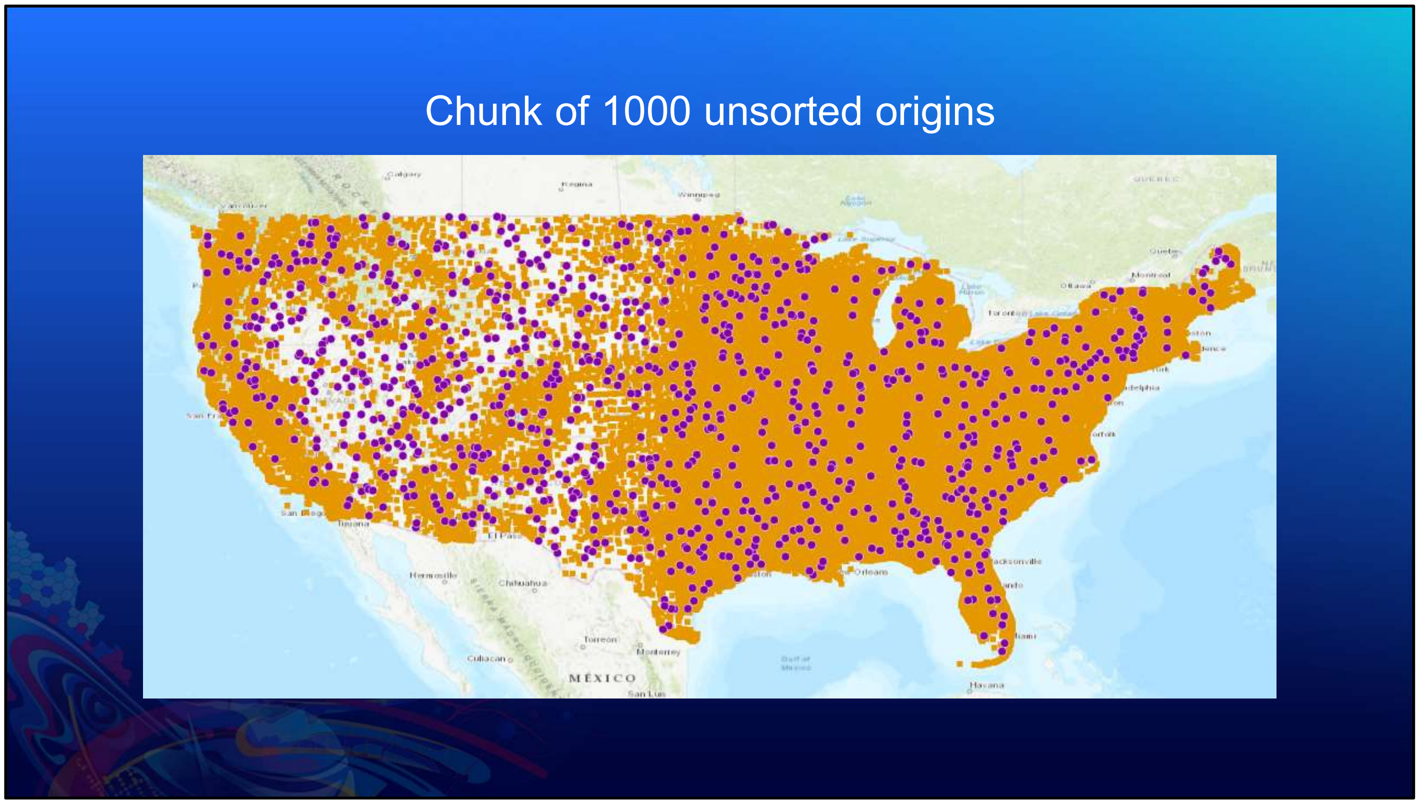
All origins and all destinations



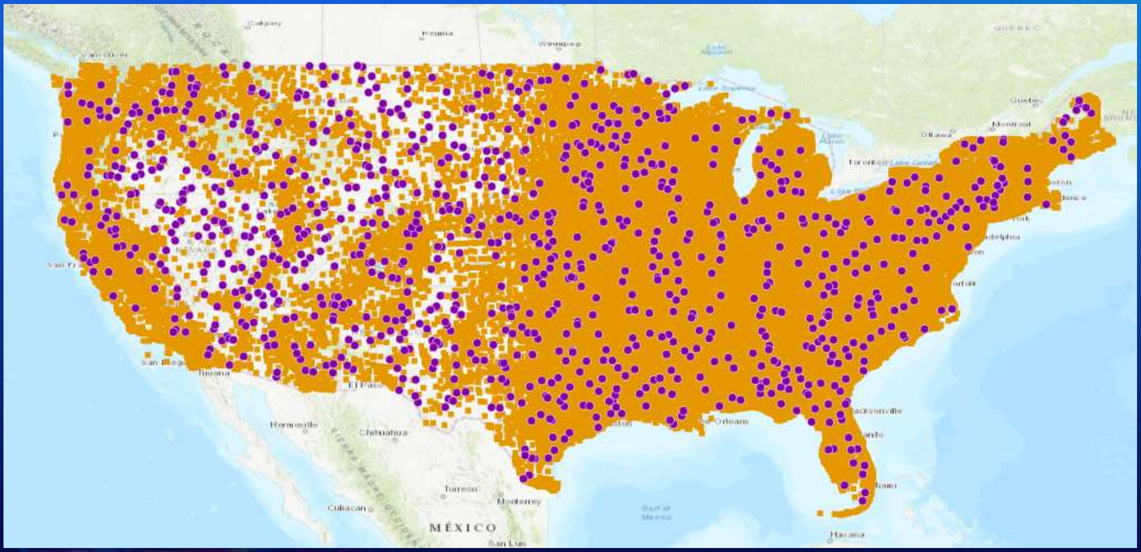
26 million origins ●, 220k destinations 🟧

The last couple things I explained were a little confusing, so I'm going to walk you through an example to make it clearer.

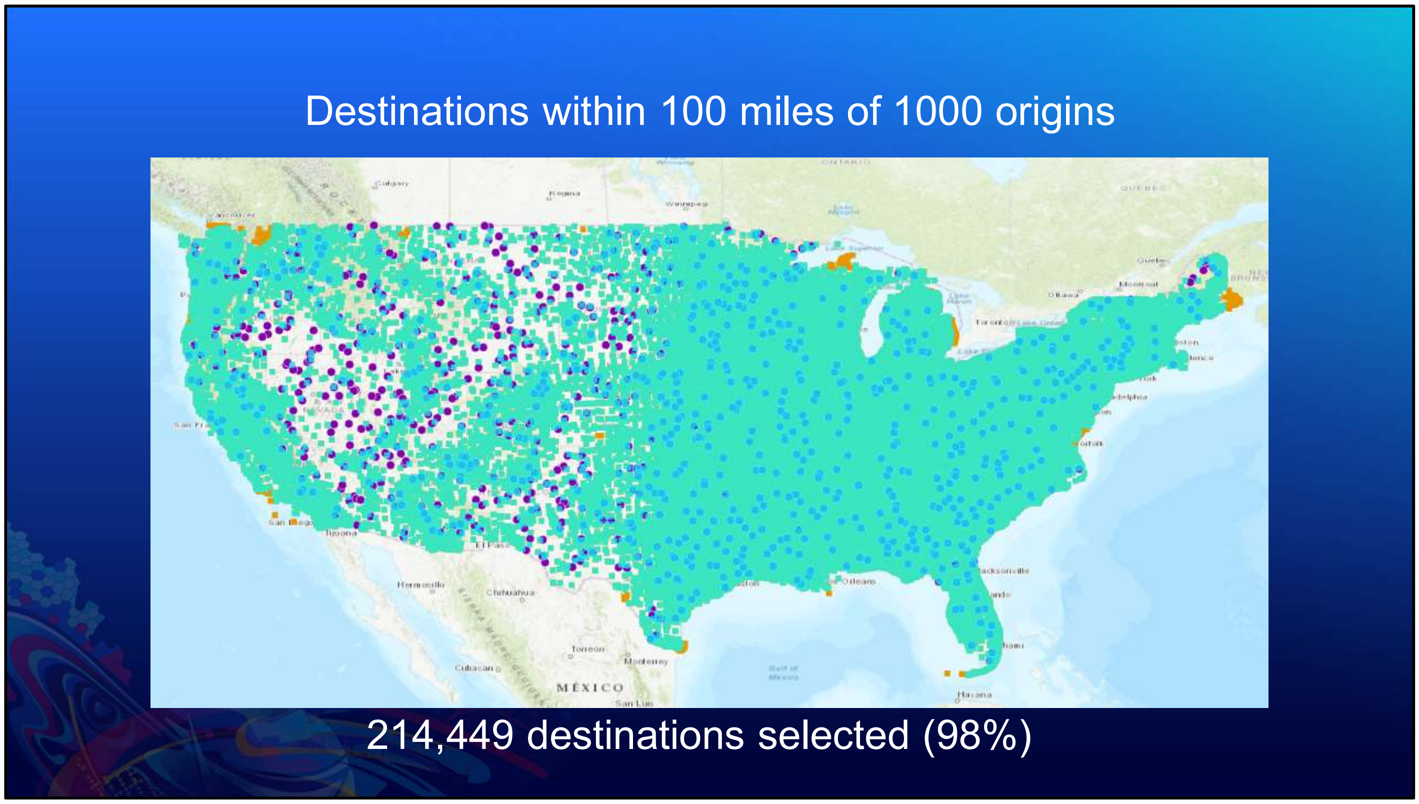
Let's say I want to solve a problem where I have 26 million origins (the purple dots) and 200K destinations (the orange squares), and I want to calculate the travel time from each origin to each destination if the destination is within 100 miles of the origin.



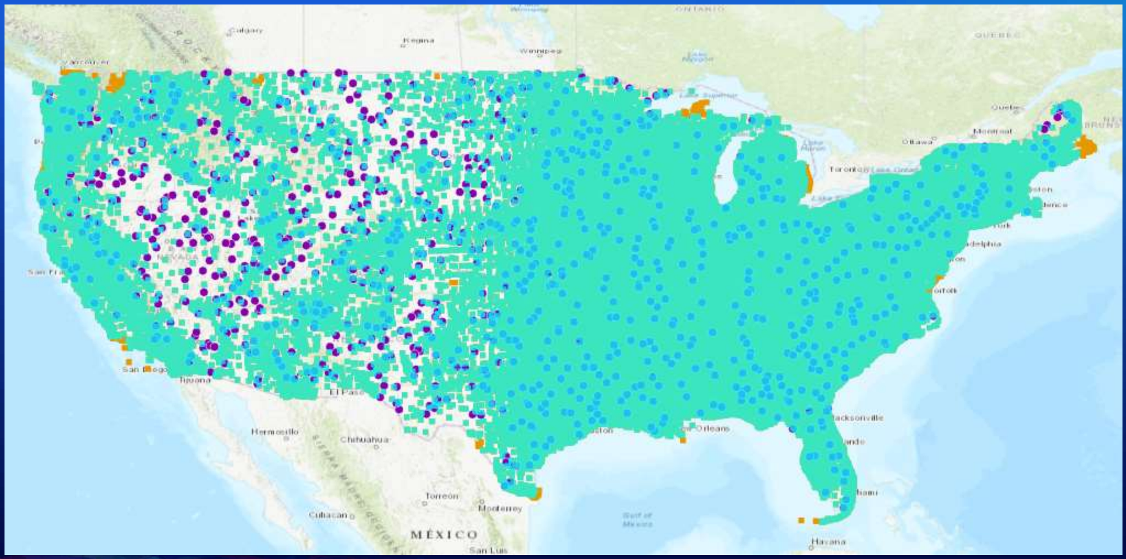
Chunk of 1000 unsorted origins



Let’s say my chunk size is 1000, and I just randomly select 1000 of my origins. They’re scattered all over the country here. Then, let’s try to reduce 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. I’ll use the Select Layer By Location tool.

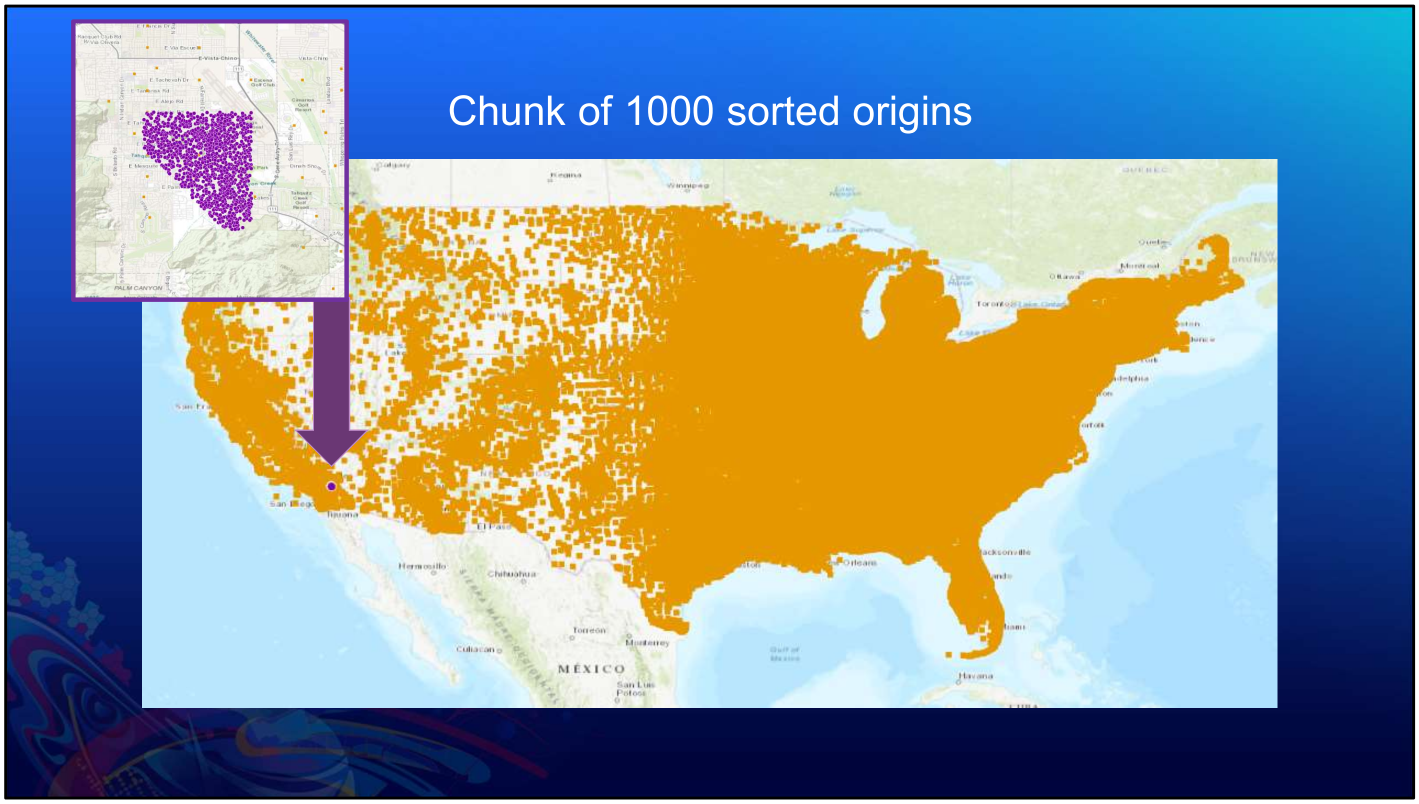


Destinations within 100 miles of 1000 origins

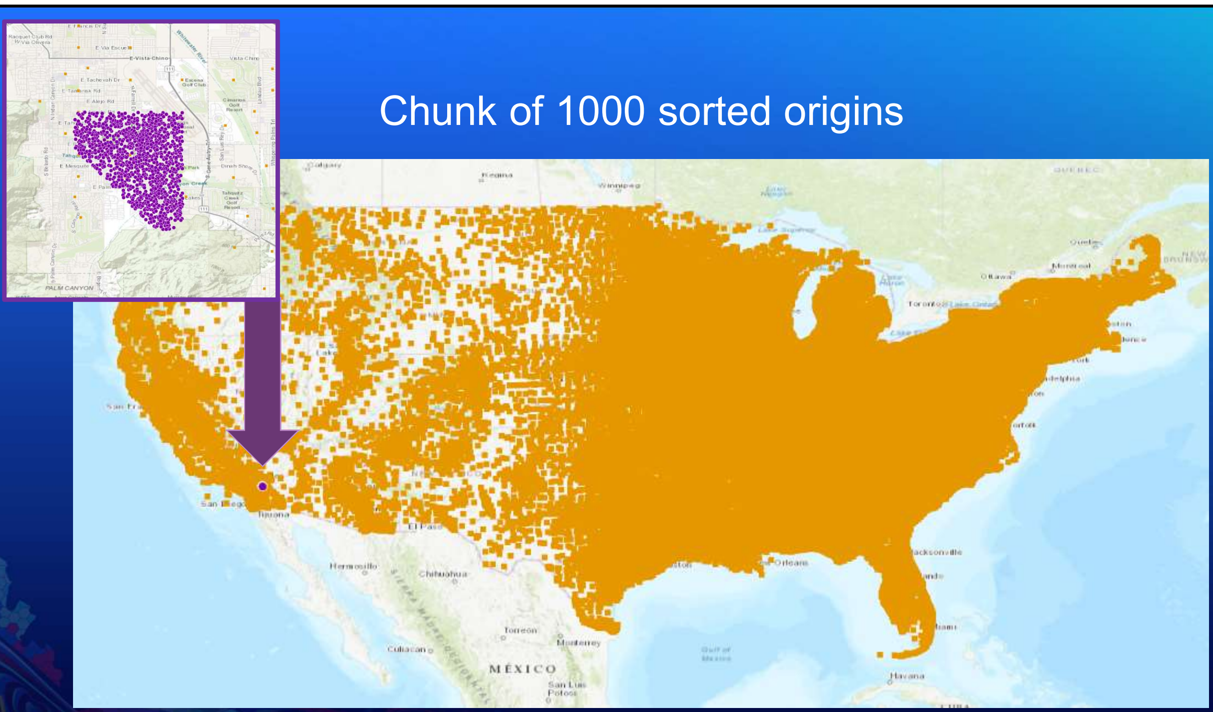


214,449 destinations selected (98%)

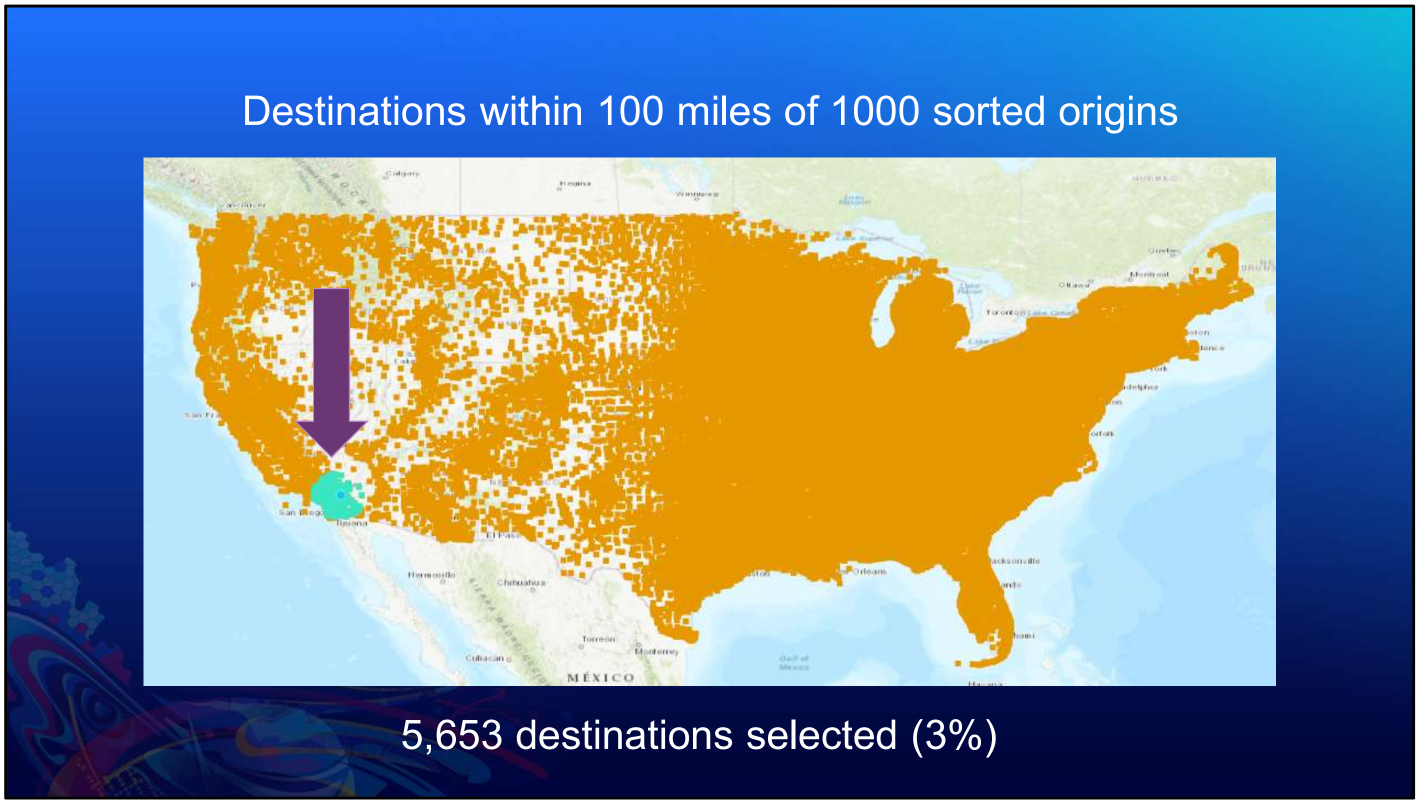
The selected destinations are all the ones that are still relevant for this chunk of origins. They're within 100 miles of at least one of my origins in the chunk. Because my origins are scattered all over the country, I end up selecting about 98% of the destinations that were there in the first place, so I haven't really successfully reduced the number of destinations that I have to handle for this chunk of origins. So if I'm using a chunk size of 1000, for this chunk of 1000 origins, I'm going to have to run through 215 chunks of destinations, and it's going to take a while.



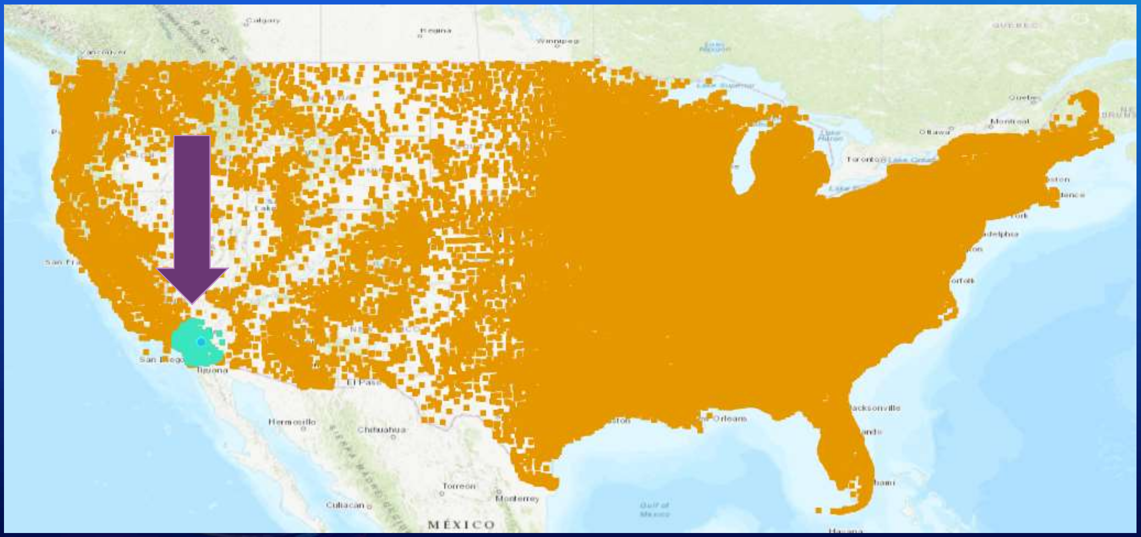
Chunk of 1000 sorted origins



Let’s say instead that I first spatially sort my origins. Now, let’s say I select a chunk of 1000. All of them are right here in the Palm Springs area. That little tiny purple dot is 1000 origins.

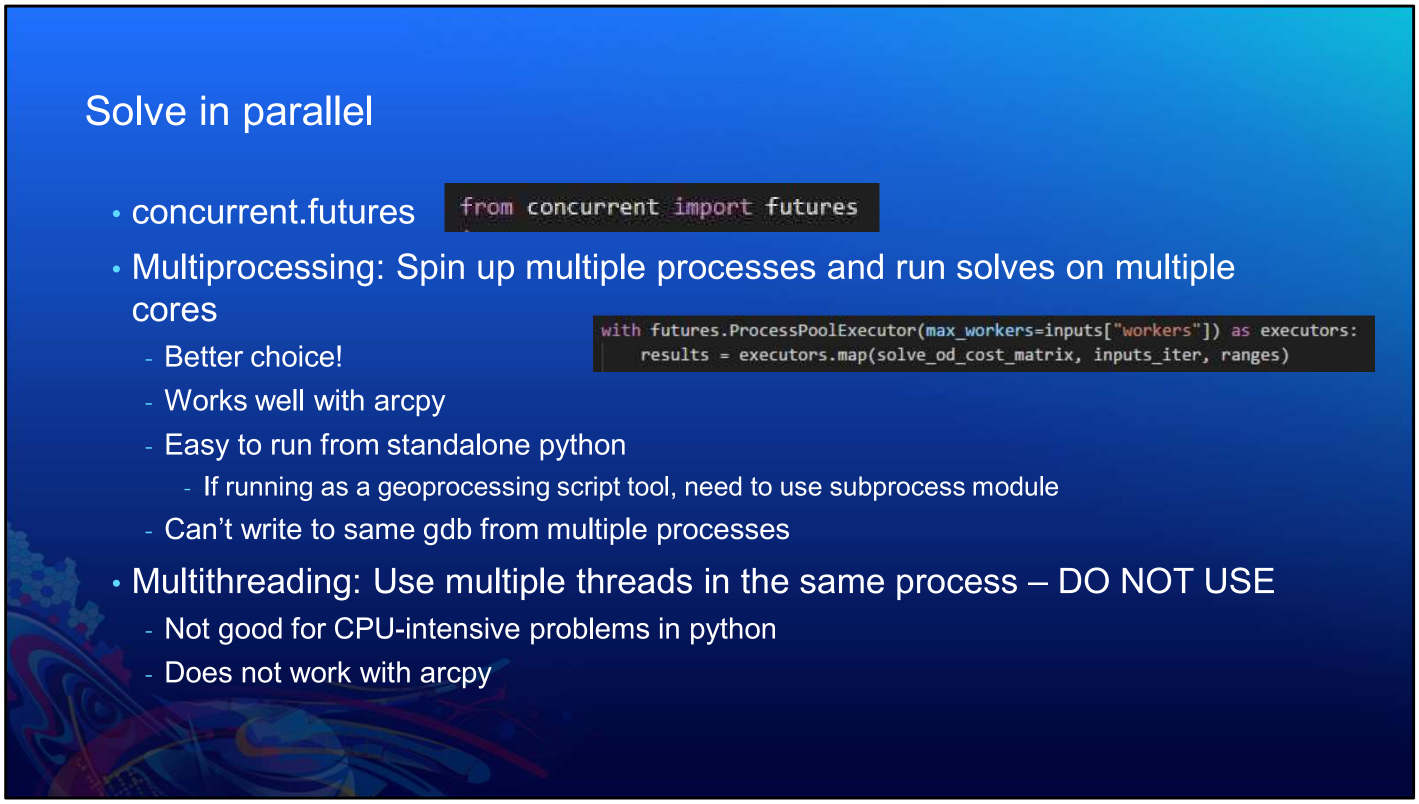


Destinations within 100 miles of 1000 sorted origins



5,653 destinations selected (3%)

Then, 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. So 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. Thus, I 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
  - Better choice!
  - Works well with arcpy
  - Easy to run from standalone python
    - If running as a geoprocessing script tool, need to use subprocess module
  - Can't write to same gdb from multiple processes

```
with futures.ProcessPoolExecutor(max_workers=inputs["workers"]) as executors:
    results = executors.map(solve_od_cost_matrix, inputs_iter, ranges)
```

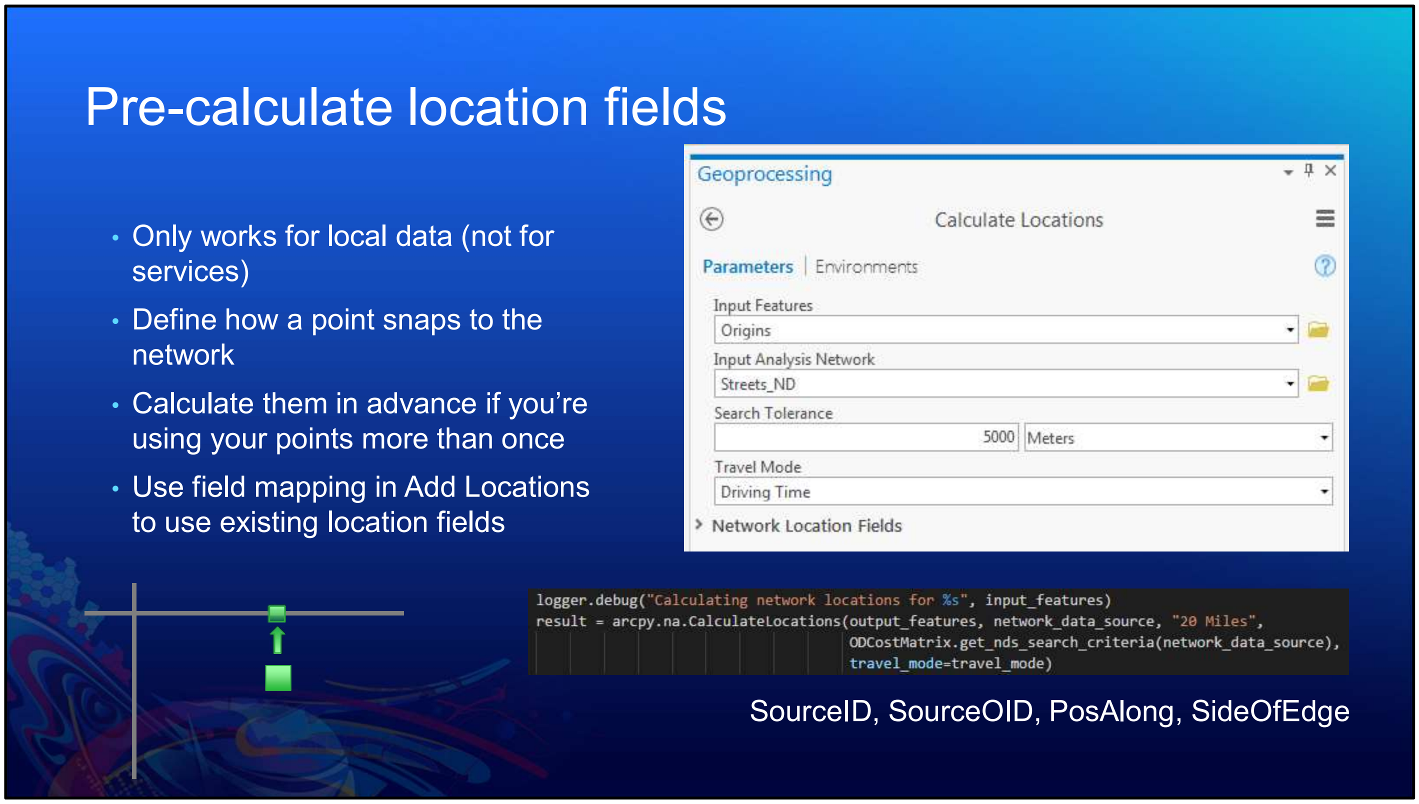
- Multithreading: Use multiple threads in the same process – DO NOT USE
  - Not good for CPU-intensive problems in python
  - Does not work with arcpy

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.

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. This allows you to spin up multiple processes and run your solves against multiple cores. It works nicely 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 directly. You'll have to spin up your process as a subprocess using the subprocess module. You also can't write to the same file geodatabse from multiple processes at the same time, so you have to be careful about how you're storing your output.

Deelesh will demonstrate how to handle these challenges in his sample script later.



Pre-calculate location fields

- Only works for local data (not for services)
- 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

Geoprocessing

Calculate Locations

Parameters | Environments

Input Features: Origins

Input Analysis Network: Streets_ND

Search Tolerance: 5000 Meters

Travel Mode: Driving Time

Network Location Fields

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

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 that's used as the actual beginning and endpoint of the network analysis. 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, since 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.

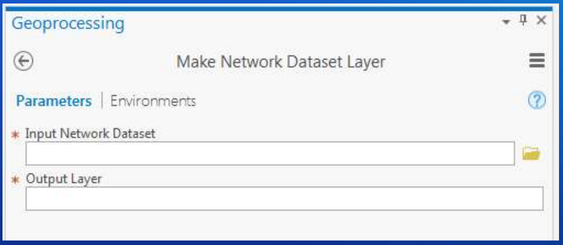
You can do this with the Calculate Locations geoprocessing tool. The fields get stored in your tables. When you load your origins and destinations, you can use field mappings to map the existing location fields, and the solver will just use those.

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. You can only calculate network locations with a local network dataset. Technically you could pre-calculate network locations if you're going to solve using an ArcGIS Enterprise service by running the Calculate Locations tool with the same network dataset that the service is published with, but that’s beyond the scope of this presentation, and our sample script tool doesn’t do that.

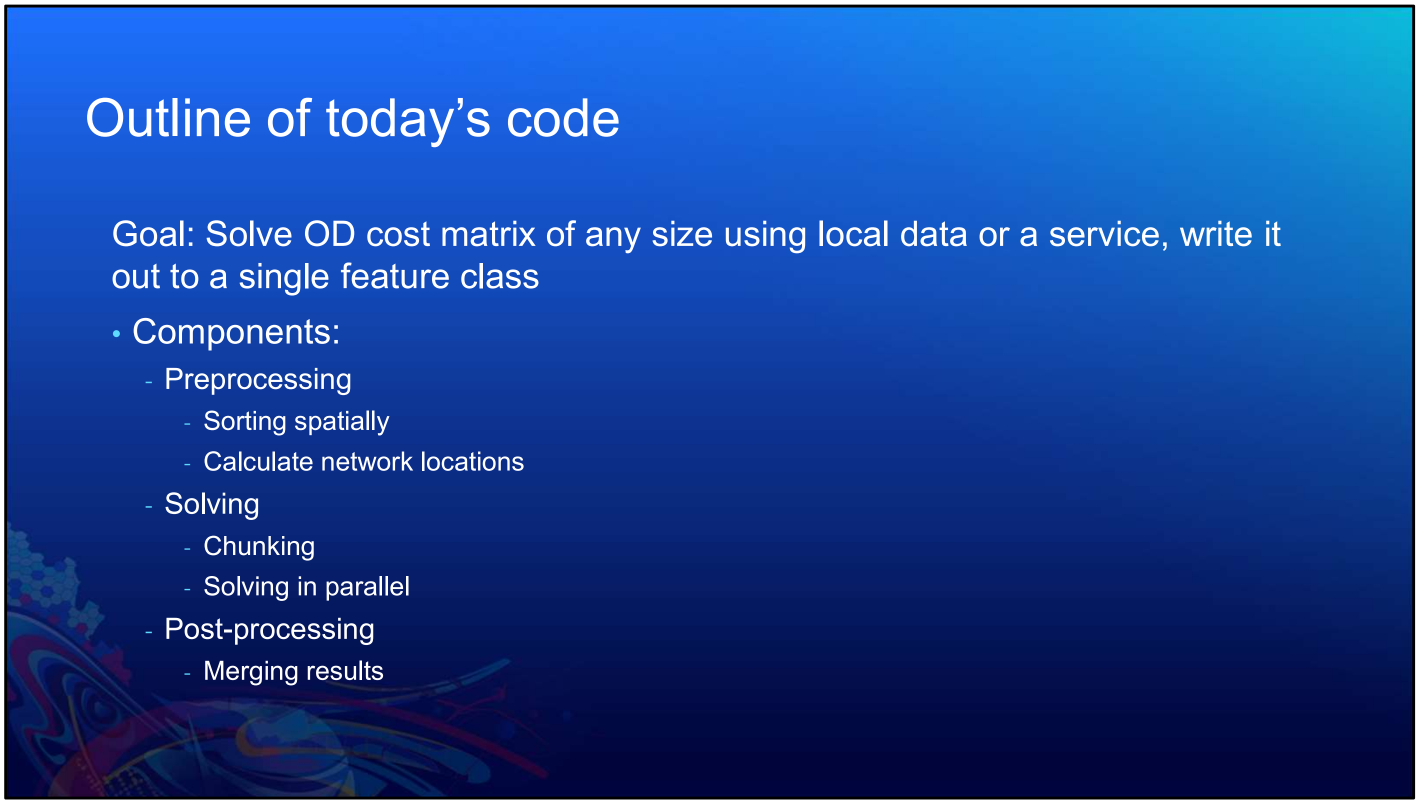
Use network dataset layer

- 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)



```
self.logger.debug("Creating network dataset layer")
arcpy.na.MakeNetworkDatasetLayer(self.network_data_source, self.nds_layer_name)
```

Finally, assuming you've followed all the other suggestions so far, 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. If you use a catalog path, the script has to open the network dataset behind the scenes every time you reference the catalog path, and this can be especially slow if you're using licensed data or data on a UNC path. If you instead use the Make Network Dataset layer tool to open the network once and then pass in that layer by referring to the layer name in other tools that use a network dataset, that can give you a bit of a performance improvement.

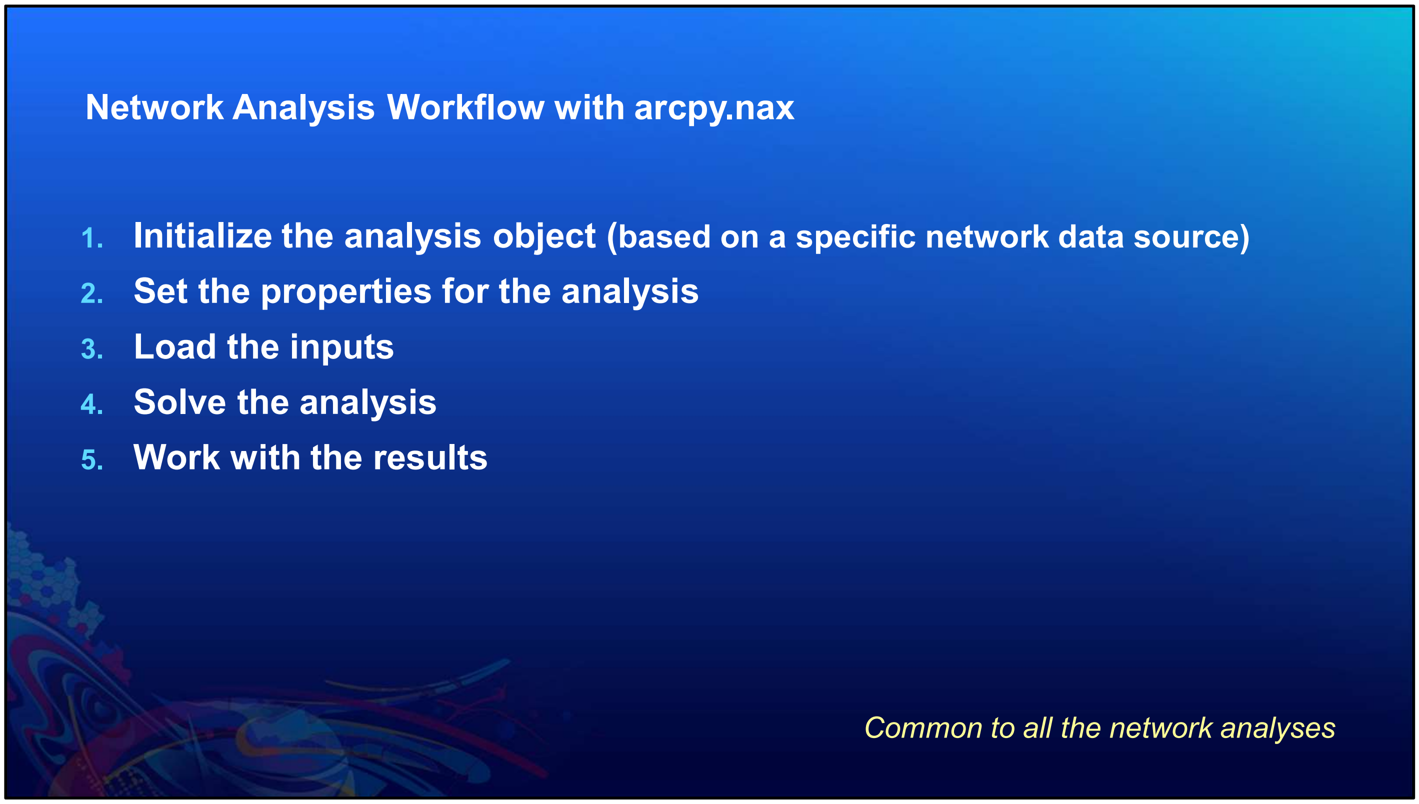


Outline of today’s code

Goal: Solve OD cost matrix 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

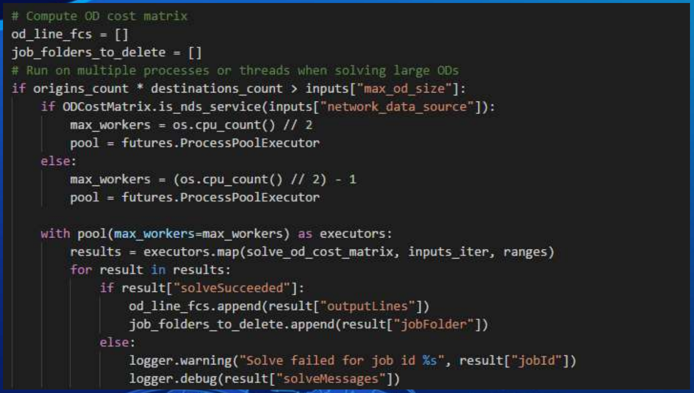
I’m going to hand it over to Deelesh to show you some code, but first, let me give you an outline of what the code is going to show.



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

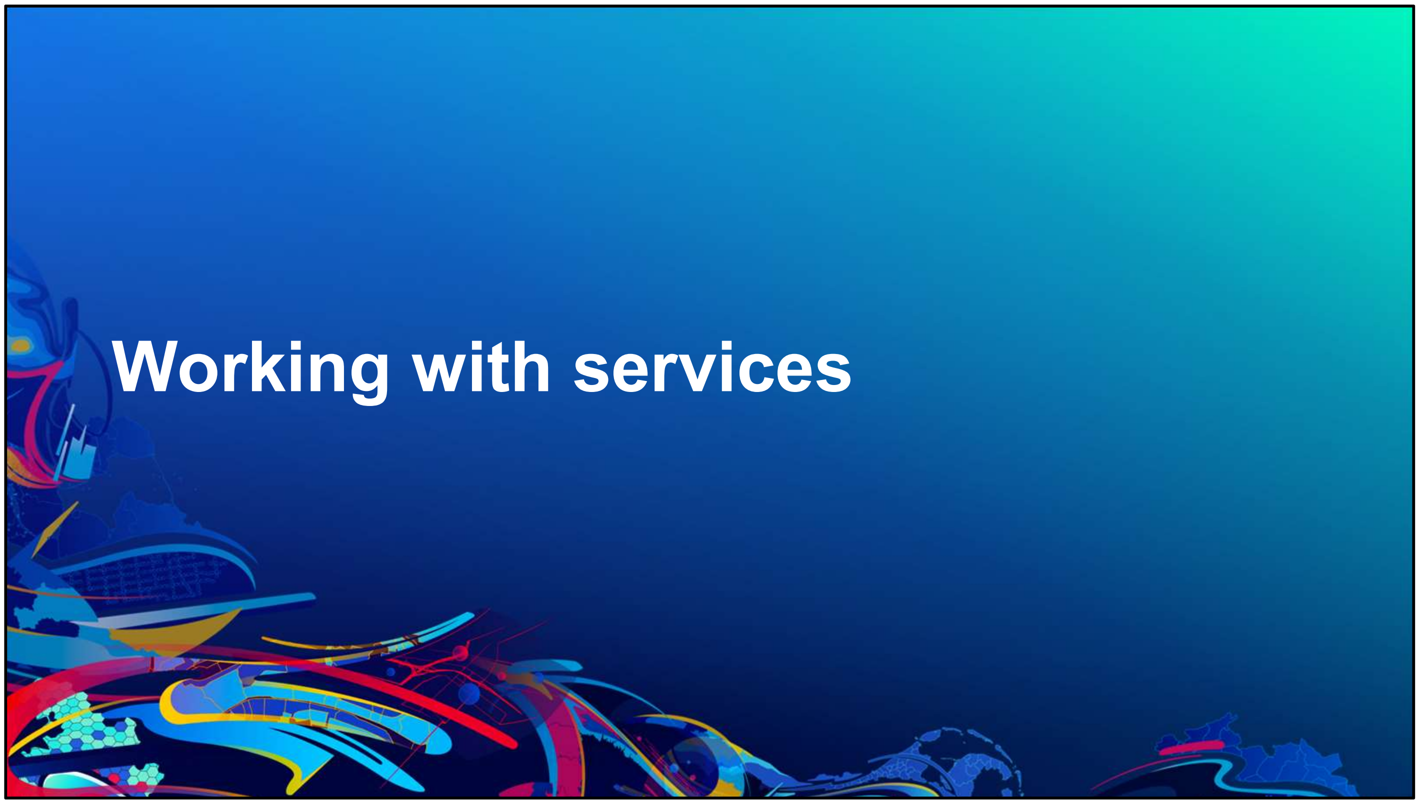


```python
# Compute OD cost matrix
od_line_fcs = []
job_folders_to_delete = []
# Run on multiple processes or threads when solving large ODs
if origins_count * destinations_count > inputs["max_od_size"]:
    if ODCostMatrix.is_nds_service(inputs["network_data_source"]):
        max_workers = os.cpu_count() // 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"])
```

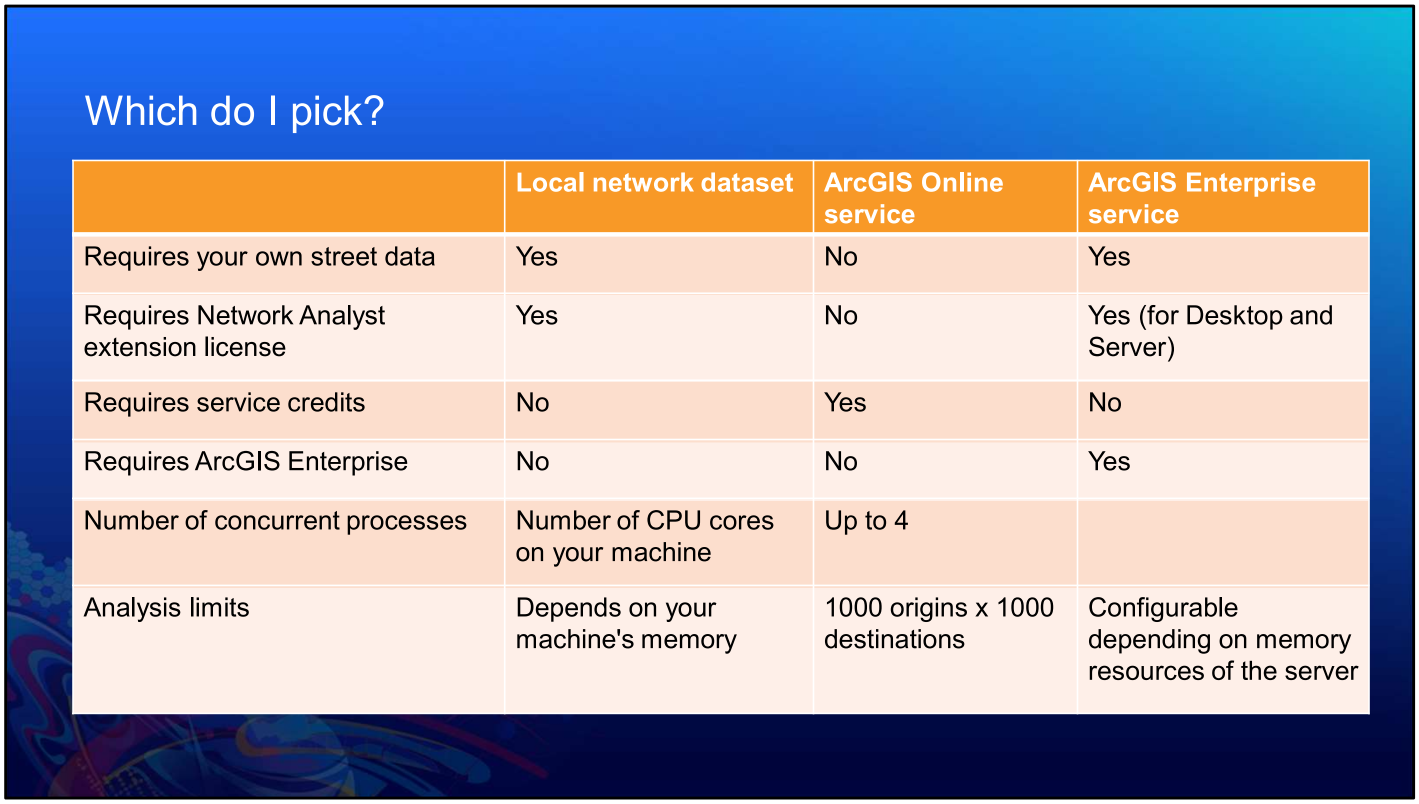
Let’s look at some code!

(Local solve)



Working with services

Melinda talked about local solves, and I showed you in the code how you can do local solves. Now let's see how you can solve this same problem if you're working with services.



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

As we said, you have two options: local solve and service solve. Service solves actually have two sub-options: ArcGIS Online or your own ArcGIS Enterprise service running on your own servers. Which do you pick?

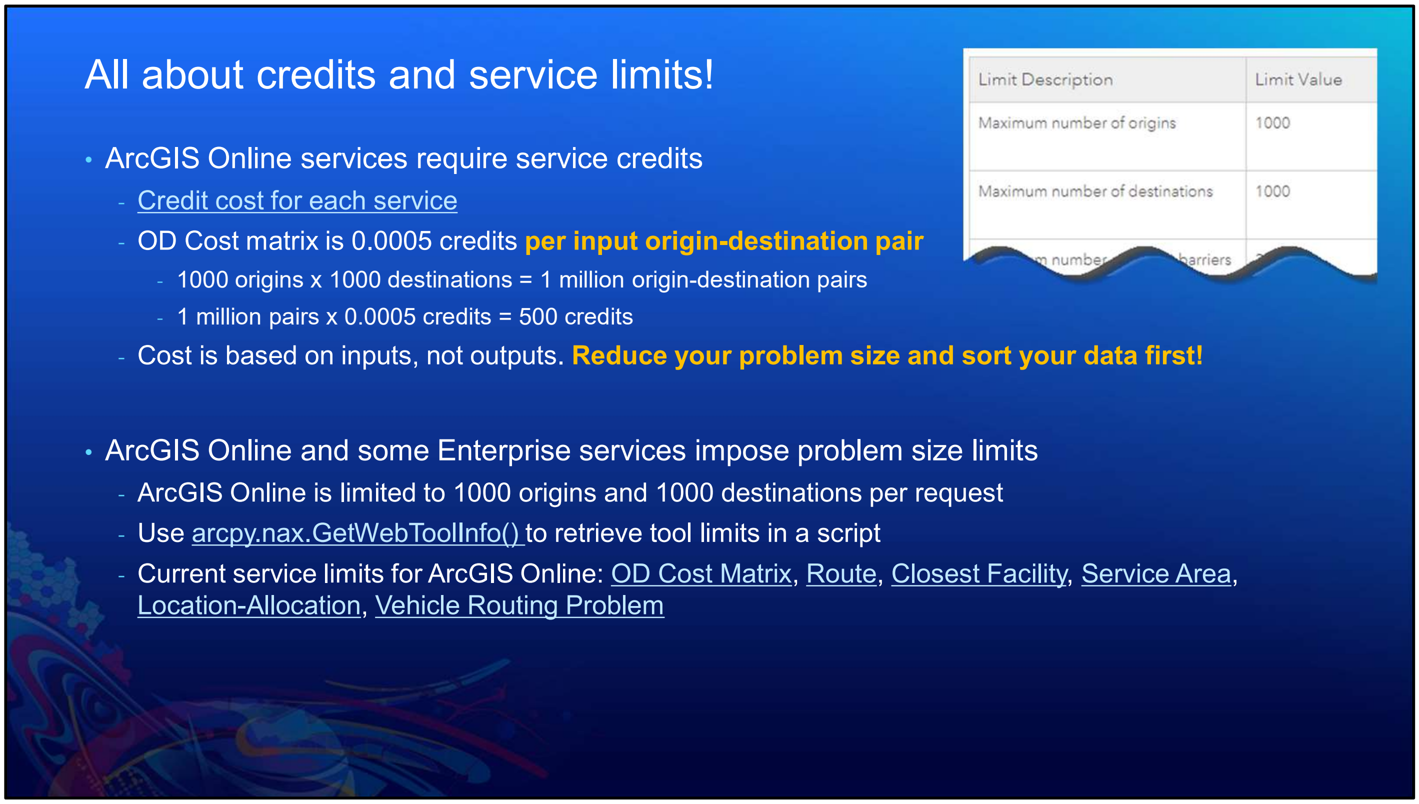
If you don't have any street data, the only option is ArcGIS Online because the street data is provided there.

If you don't have a Network Analyst Extension license for ArcGIS Pro, or ArcGIS Enterprise, then ArcGIS Online is your only option. You don't need the Network Analyst Extension, just service credits.

However, the ArcGIS Online service imposes limits on the size of the problem you can solve: 1000 origins and 1000 destinations in a single request. If you have a very large problem, you might need to use a large number of chunks. Also, the number of credits required to solve your problem might be unreasonably large.

You can solve locally on your own machine if you have your own street data and the Network Analyst License. You should do a local solve if you just have one machine with ArcGIS Pro running on it.

If you have ArcGIS Enterprise, you can speed the problem up by running the process on a bank of servers. Each one can process chunks for you in parallel. You can have as many servers as you want, and you don't have the same analysis input limits that you do with ArcGIS Online.



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 and some Enterprise services impose 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
m number barriers	

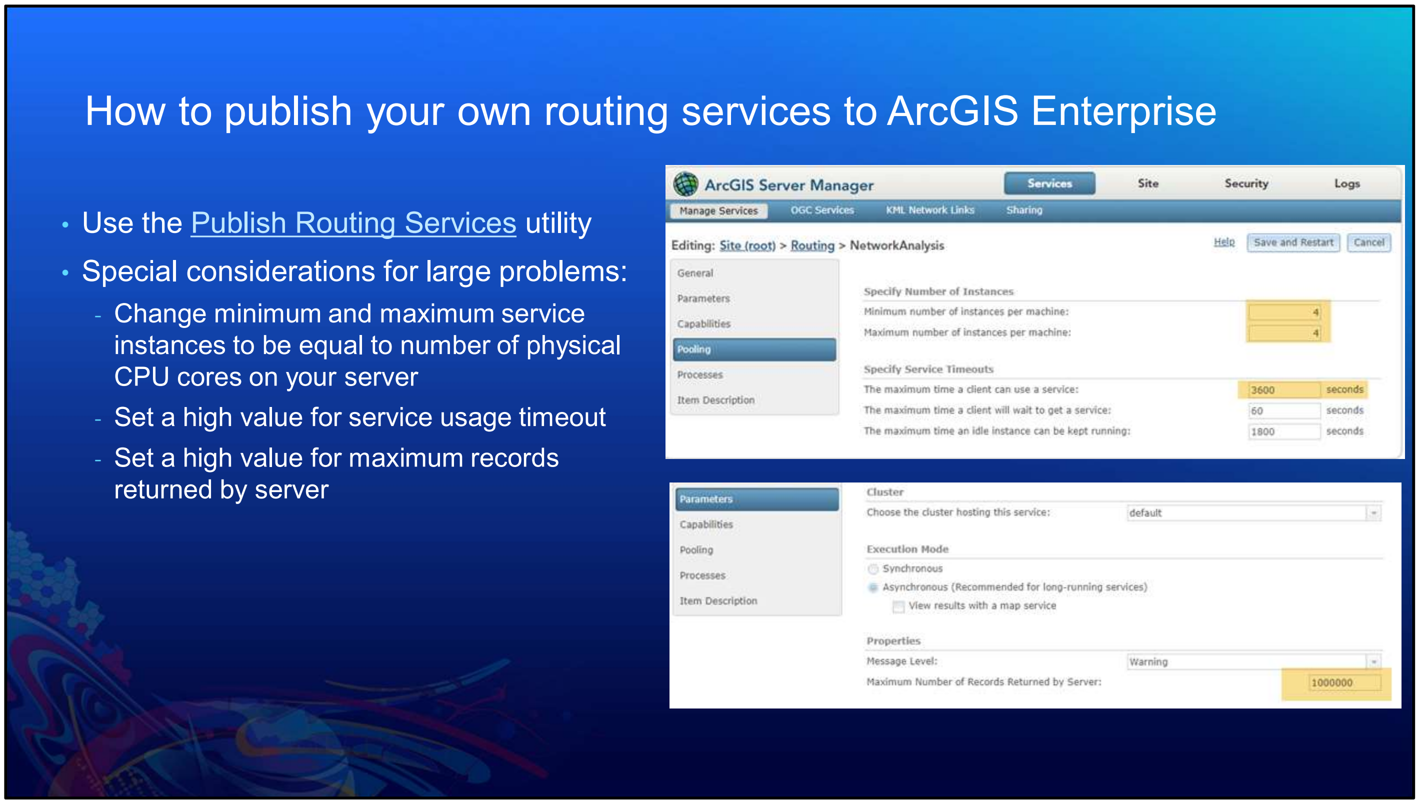
If you're working with ArcGIS Online, you have to worry about two things: credits and limits.

The ArcGIS Online services charge credits for each solve. For OD Cost Matrix, it costs 0.0005 credits *per input origin-destination pair*. 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. For this reason, it's especially important to reduce your problem size using the techniques we described earlier. You don't want to send irrelevant destinations to the service because you will be charged for them.

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.

The other thing you need to consider with services is whether the service imposes some limits on the numbers of inputs. For example, the ArcGIS Online services limit OD Cost Matrix analyses to 1000 origins and 1000 destinations. These limits determine the maximum chunk size you can use when breaking up your large problem.

You can use the GetWebToolInfo() function to retrieve tool limits from a service dynamically in a script. That way you don’t have to hard-code any values that might change. We’ll show this in our sample script in just a minute.



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

Editing: Site (root) > Routing > NetworkAnalysis

Specify Number of Instances

Minimum number of instances per machine: 4

Maximum number of instances per machine: 4

Specify Service Timeouts

The maximum time a client can use a service: 3600 seconds

The maximum time a client will wait to get a service: 60 seconds

The maximum time an idle instance can be kept running: 1800 seconds

Cluster

Choose the cluster hosting this service: default

Execution Mode

Synchronous [✓ Asynchronous (Recommended for long-running services)]

☐ View results with a map service

Properties

Message Level: Warning

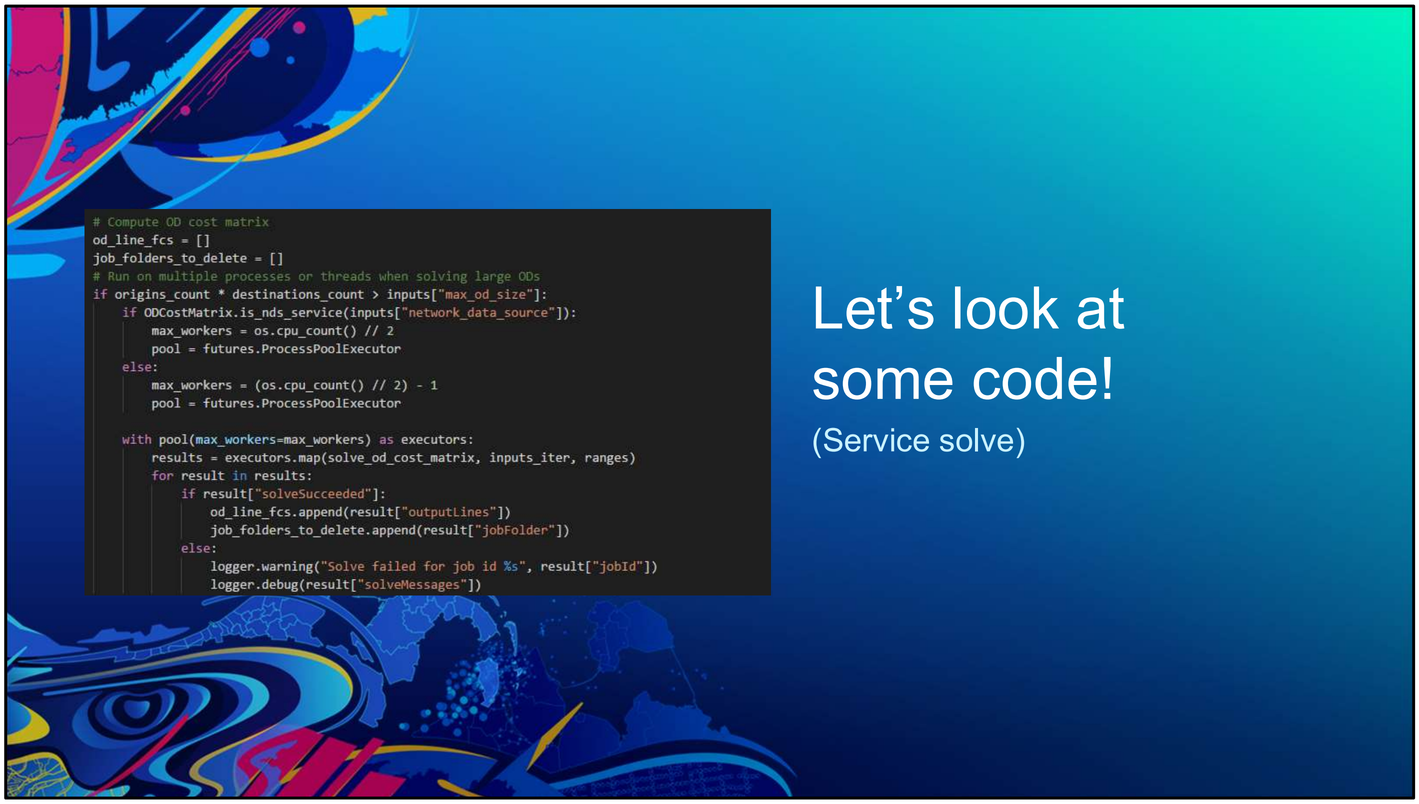
Maximum Number of Records Returned by Server: 1000000

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 certain settings you should 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.

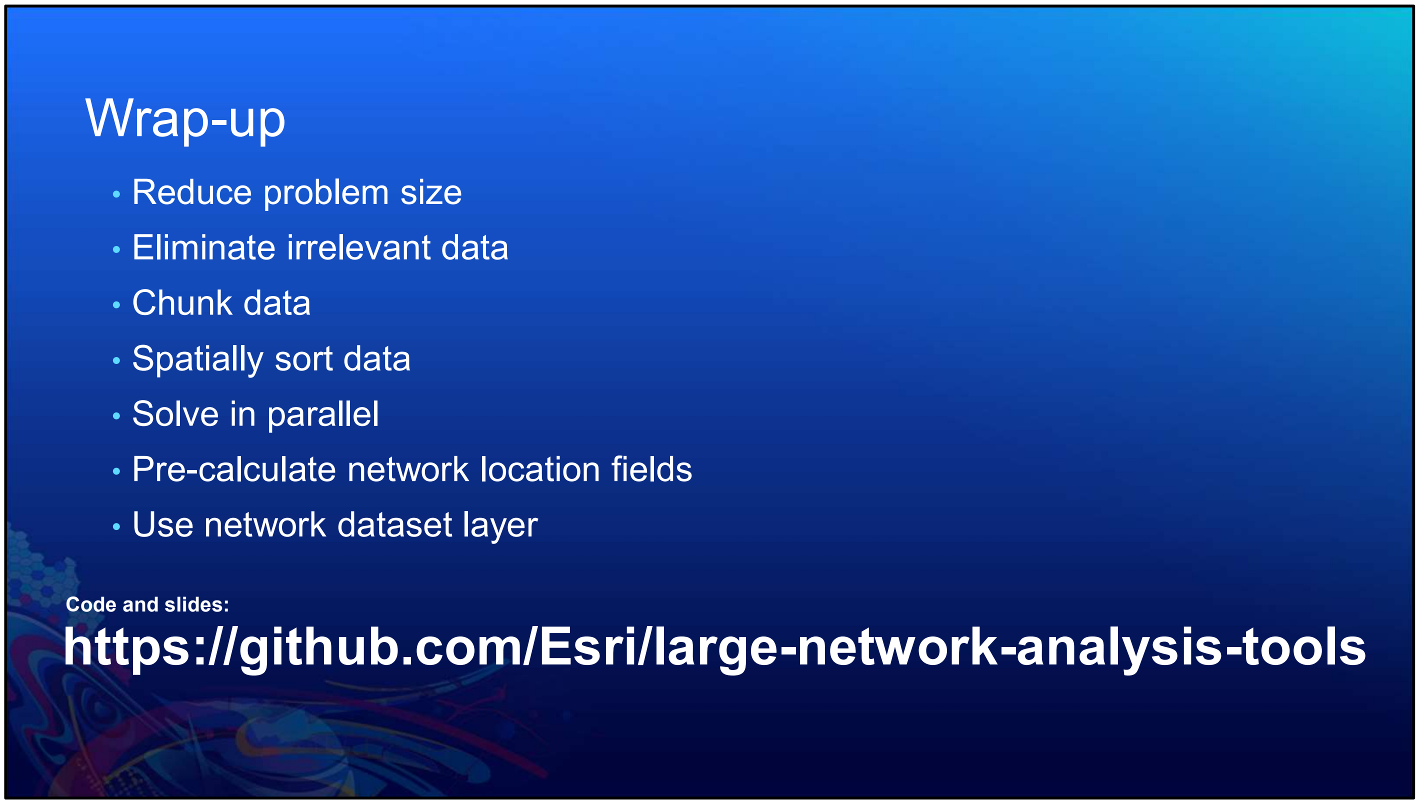


```python
# Compute OD cost matrix
od_line_fcs = []
job_folders_to_delete = []
# Run on multiple processes or threads when solving large ODs
if origins_count * destinations_count > inputs["max_od_size"]:
    if ODCostMatrix.is_nds_service(inputs["network_data_source"]):
        max_workers = os.cpu_count() // 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"])
```

Let’s look at some code!

(Service solve)



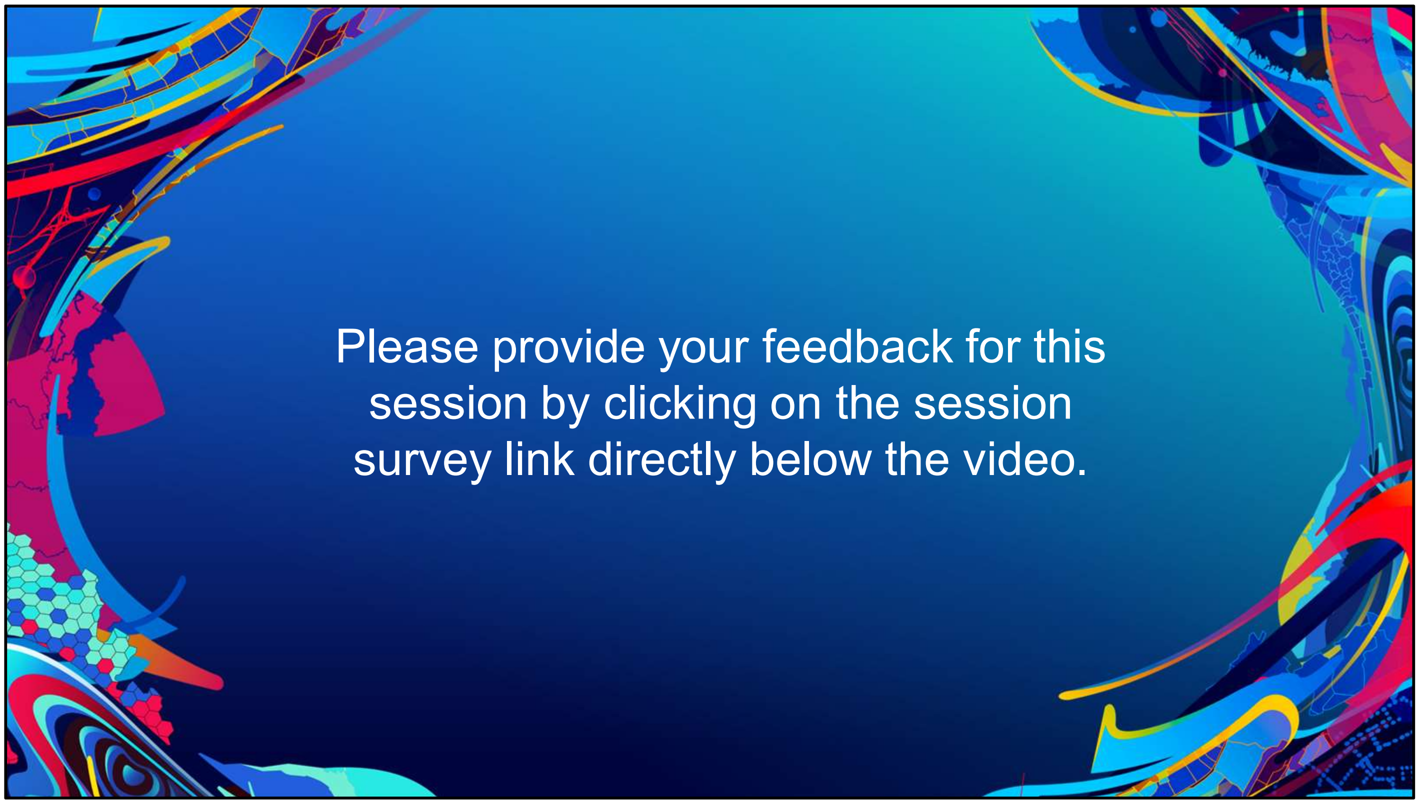
Wrap-up

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

Code and slides:

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

esri
THE SCIENCE OF WHERE®



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