Background

There are **two main components** implemented.

- **UX** implemented with **JavaScript**
- **Optimization Engine** implemented with **Python**

UX Component

The **main function** of the component resides in `\afms\src\AFMS\FSOptimizer\run.js`

Description

UX will get the following inputs (`file` and `output_folder`) and post it to the **web URL** as listed below

- `file == scenarioFile` (all the parameters setting for a particular scenario)
- `output_folder == fs_optimizer_results` (output folder name)

POST TRIGGER

```
http://localhost:5000/run
```

Optimization Engine Component

The **main function** of the component resides in `\optimizer\server.py`

Description

`server.py` is the **server file** for receiving `POST` commands from the **UX component**.

The **main API call** of the component resides in `\optimizer\engine_frame.py`

> **FUNCTION**
>
> ```
> API_main_run_engine(input_folder, output_folder,
> need_pre_process=True)
> ```

- **Main API** is called by `server.py`
- *input_folder* : `scenarioFile` is extracted and placed in a folder for the optimization model
- *output_folder* : for writing out all the result files

Local Machine Setup

Let's setup the tool for usage.

Getting Started

Operating System

- Windows 10

Required Software

- Python 3.7 or higher
- SCIPOptSuite-8.0.0-win64-VS15
- Node-v16.15.0-x64

Dependencies

- Refer to `readme_6sep2022`

Running the tool locally

Front-End Setup

Client (UX Component)

1. Start **Command Prompt** or **Terminal**
2. Run the following commands

```
cd afms
yarn start
```

3. Open browser `http://localhost:8080`

Client (UX Component, no internet connectivity required)

1. Start **Command Prompt** or **Terminal**
2. Run the following commands

```
cd afms
yarn build
cd afms\dist
python –m http.server 8080
```

3. Open browser `http://localhost:8080`

Back-End Setup

Server (Optimization Engine Component)

1. Start **Anaconda Prompt**
2. Run the following commands

```
cd optimiser
```

3. Environment setup *(you only need to do this step if you are initialising the environment for the first time)*

```
conda create –n opt
conda activate opt
conda install geopandas
conda install pyproj
pip install flask
pip install flask_cors
pip install pyscipopt
```

4. Run the following commands

```
conda activate opt
python server.py
```

Docker Setup

If you wish to use the pre-built images, follow the steps below. You can either do it via Docker Compose or run two images separately.

Docker Compose

EASY SETUP

Copy the file below, save it as `docker-compose.yml`. Then run the following command in your **Command Prompt** or **Terminal**.

```
docker compose up
```

docker-compose.yml

```yaml
version: '1'
services:
  front-end:
    image: eldoraboo/afms:latest
    ports:
      - "81:80"
    depends_on:
      - back-end
  back-end:
    image: eldoraboo/optimizer:latest
    ports:
      - "5000:5000"
```

Alternatively, you may follow the instructions below to run two images separately.

Front-End Setup

```
docker pull eldoraboo/afms:latest
docker run  --name front-end -d -p 81:80 eldoraboo/afms:latest
```

Back-End Setup

```
docker pull eldoraboo/optimizer:latest
docker run --name back-end -d -p 5000:5000 eldoraboo/
optimizer:latest
```

Setting Up Docker

Setting Up Docker for the AFMS Project

Creating Dockerfiles

Front-End Dockerfile

Building Docker Images

Once you have created the Dockerfiles, you can build the Docker images.

Pushing to DockerHub

Retrieve Image ID

Exporting Docker Images

Retrieve Image Name

Pulling from DockerHub

Pulling Latest Images

Deploying Containers

Docker Run

Creating Dockerfiles

Front-End Dockerfile

Stage 1 - The Build Process

afms\Dockerfile

```
FROM node:12-alpine as build-deps
WORKDIR /usr/src/app
COPY package.json yarn.lock ./
RUN apk add --no-cache autoconf
RUN yarn
COPY . ./
RUN yarn build
```

Stage 2 - The Production Environment

afms\Dockerfile

```
FROM nginx:1.21-alpine
COPY --from=build-deps /usr/src/app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Completed Dockerfile

afms\Dockerfile

```
# Stage 1 - the build process
FROM node:12-alpine as build-deps
WORKDIR /usr/src/app
COPY package.json yarn.lock ./
RUN apk add --no-cache autoconf
RUN yarn
COPY . ./
RUN yarn build

# Stage 2 - the production environment
FROM nginx:1.21-alpine
COPY --from=build-deps /usr/src/app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Back-End Dockerfile

Stage 1 - Installing Dependencies

optimiser\Dockerfile

```
FROM continuumio/miniconda3:latest
WORKDIR /app
COPY *.py .
COPY requirements.txt .
RUN pip install -r requirements.txt
```

Stage 2 - conda & pip Setup

optimiser\Dockerfile

```
RUN pip install flask flask_cors
COPY environment.yml .
RUN conda config --set channel_priority strict
RUN conda create -n opt python=3.9
RUN /bin/bash -c "source activate opt"
RUN conda install -c conda-forge scip
RUN conda install -c conda-forge pyscipopt
RUN conda install -c conda-forge geopandas
RUN conda install -c conda-forge pyproj
```

Stage 3 - Activate Script

optimiser\Dockerfile

```
COPY server.py .
EXPOSE 5000
SHELL ["conda", "run", "-n", "opt" "/bin/bash", "-c"]
CMD ["python", "server.py"]
```

Completed Dockerfile

optimiser\Dockerfile

```
# Stage 1 - installing dependencies
FROM continuumio/miniconda3:latest
WORKDIR /app
COPY *.py .
```

Building Docker Images

Once you have created the Dockerfiles, you can build the Docker images.

Front-End Image

The `docker build` command builds the **Docker image** using the instructions in the Dockerfile. The `.` argument specifies the **location** of the Dockerfile, and the `-t` argument specifies the **name** of the image (in this case, `afms`).

```
docker build . -t afms
```

Once the Docker image has been built, you can use the following command to run the Docker container.

```
docker run -d -p 81:80 afms
```

Back-End Image

The `docker build` command builds the **Docker image** using the instructions in the Dockerfile. The `.` argument specifies the **location** of the Dockerfile, and the `-t` argument specifies the **name** of the image (in this case, `optimizer`).

```
docker build . -t optimizer
```

Once the Docker image has been built, you can use the following command to run the Docker container.

```
docker run -d -p 5000:5000 optimizer
```

Pushing to DockerHub

Retrieve Image ID

Retrieve the **image ID** using

```
docker images
```

and you should see something similar to this

```
REPOSITORY            TAG       IMAGE ID       CREATED         SIZE
eldoraboo/optimizer   latest    eddc6908c0fb   2 hours ago
4.39GB
eldoraboo/afms        latest    1d0bf74ef845   2 hours ago
44.9MB
```

Tag and Push Your Image

Tag your images with a name. In general, a good choice for a tag is something that will help you understand what this container should be used in conjunction with, or what it represents. Then push your image into DockerHub.

```
# for front-end image
docker tag 1d0bf74ef845 eldoraboo/afms:latest
docker push eldoraboo/afms

# for back-end image
```

Exporting Docker Images

Retrieve Image Name

Retrieve the **image name** using

```
docker images
```

and you should see something similar to this

```
REPOSITORY            TAG       IMAGE ID       CREATED         SIZE
eldoraboo/optimizer   latest    eddc6908c0fb   2 hours ago
4.39GB
eldoraboo/afms        latest    1d0bf74ef845   2 hours ago
44.9MB
```

Exporting as .tar files

```
# for front-end image
docker save eldoraboo/afms > afms.tar

# for back-end image
docker save eldoraboo/optimizer > optimizer.tar
```

Pulling from DockerHub

Pulling Latest Images

```
# for front-end image
docker pull eldoraboo/afms:latest

# for back-end image
docker pull eldoraboo/optimizer:latest
```

Deploying Containers

Docker Run

```
# for front-end image
docker run  --name front-end -d -p 81:80 eldoraboo/afms:latest

# for back-end image
docker run --name back-end -d -p 5000:5000 eldoraboo/
optimizer:latest
```

Docker Compose

EASY SETUP

Copy the file below, save it as `docker-compose.yml`. Then run the following command in your **Command Prompt** or **Terminal**.

```
docker compose up
```

docker-compose.yml

```
version: '1'
services:
  front-end:
    image: eldoraboo/afms:latest
```

How to Use AFMS

Features added to the existing AFMS Project.

Introduction

Widget

Format of Hub Files

Macro Hub Distribution

Format of Supporting Files

Physical Node

Input Variables

| name | description | remarks | |

📄 Example Scenarios

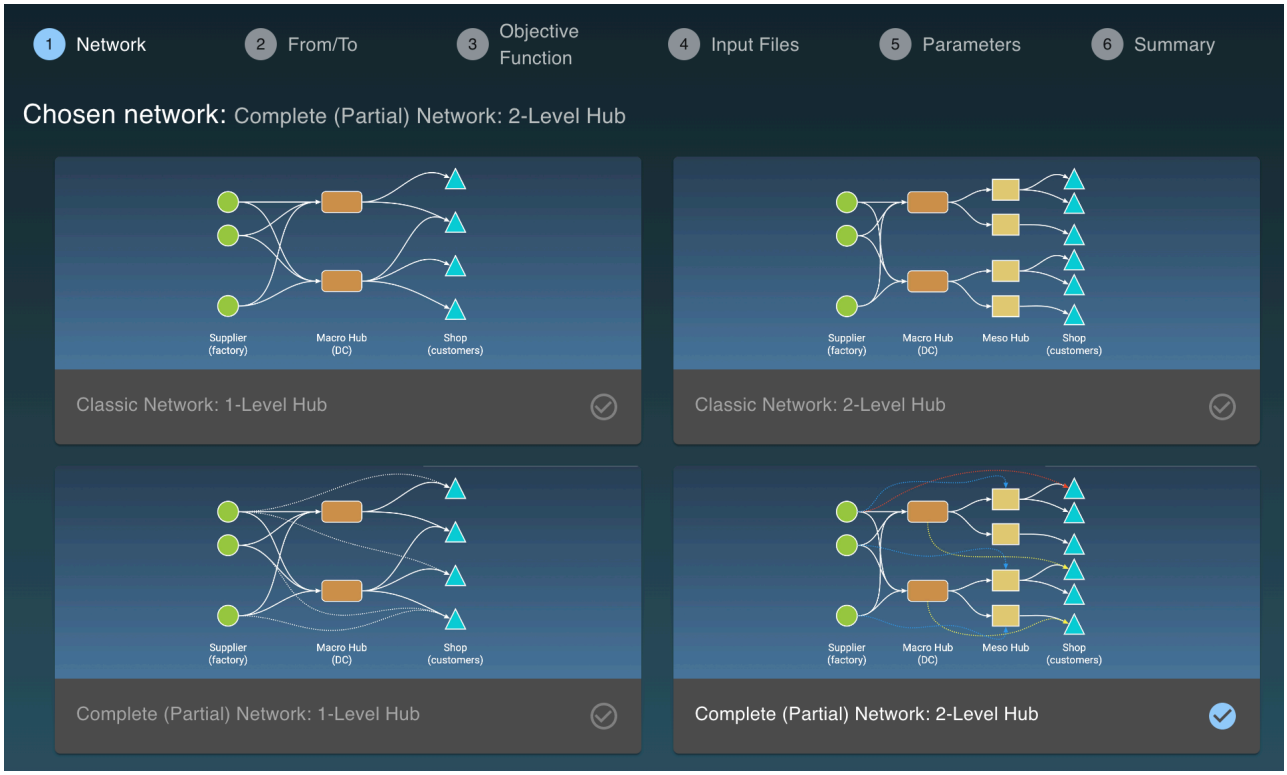
Scenario 1

Introduction

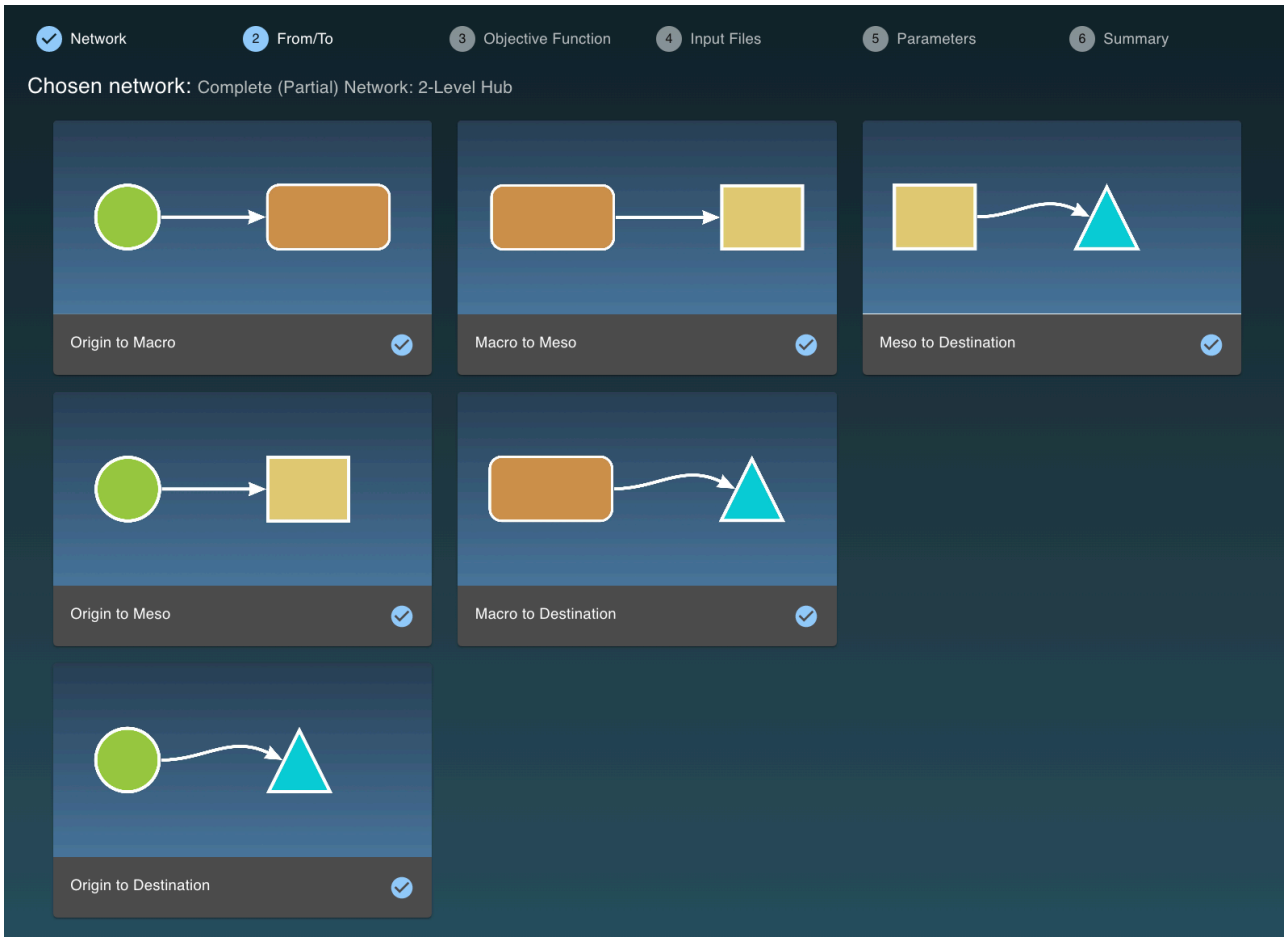
Widget

Choosing Hubs Connectivity

Choose the **network** for this scenario.

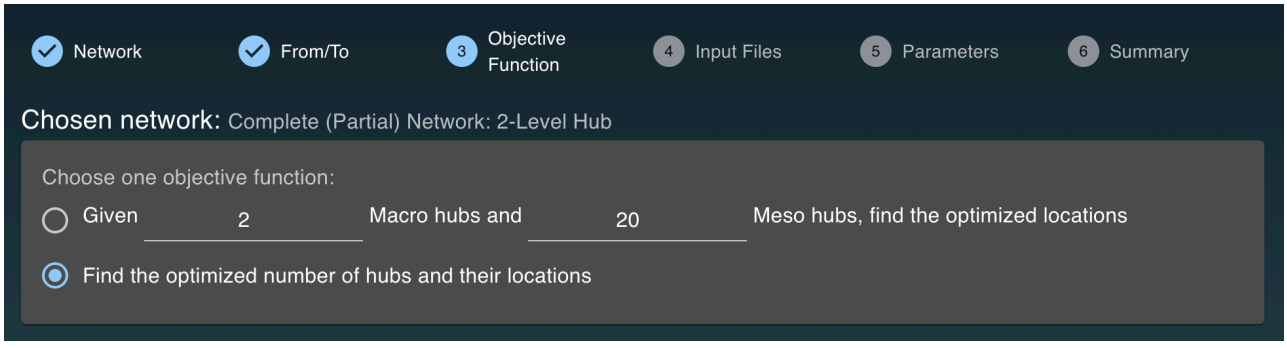


Then select **connections** you would like to optimize.

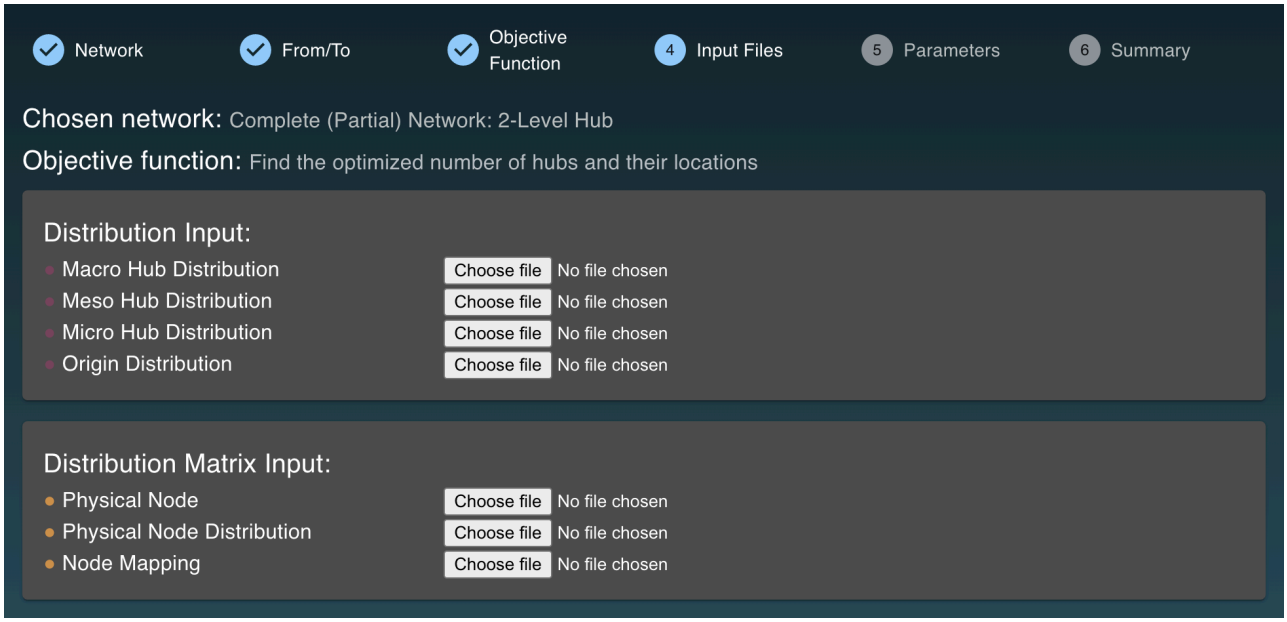


Setting Parameters & Inputs

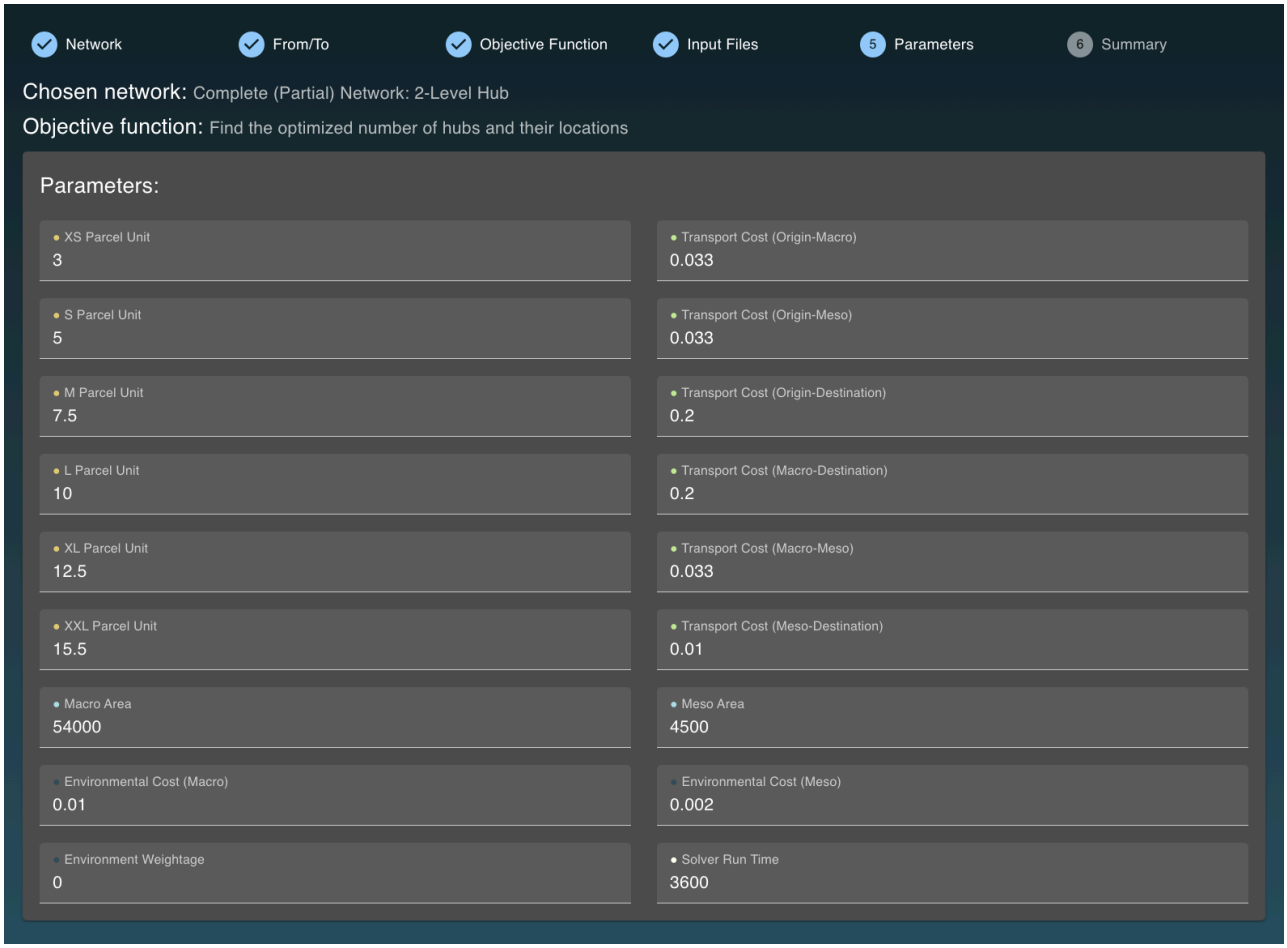
For the **optimization**, you may choose to optimize based on just the **locations**, or both the **locations** and number of **hubs**.



Then, put in the **input files** according to the **format required**.

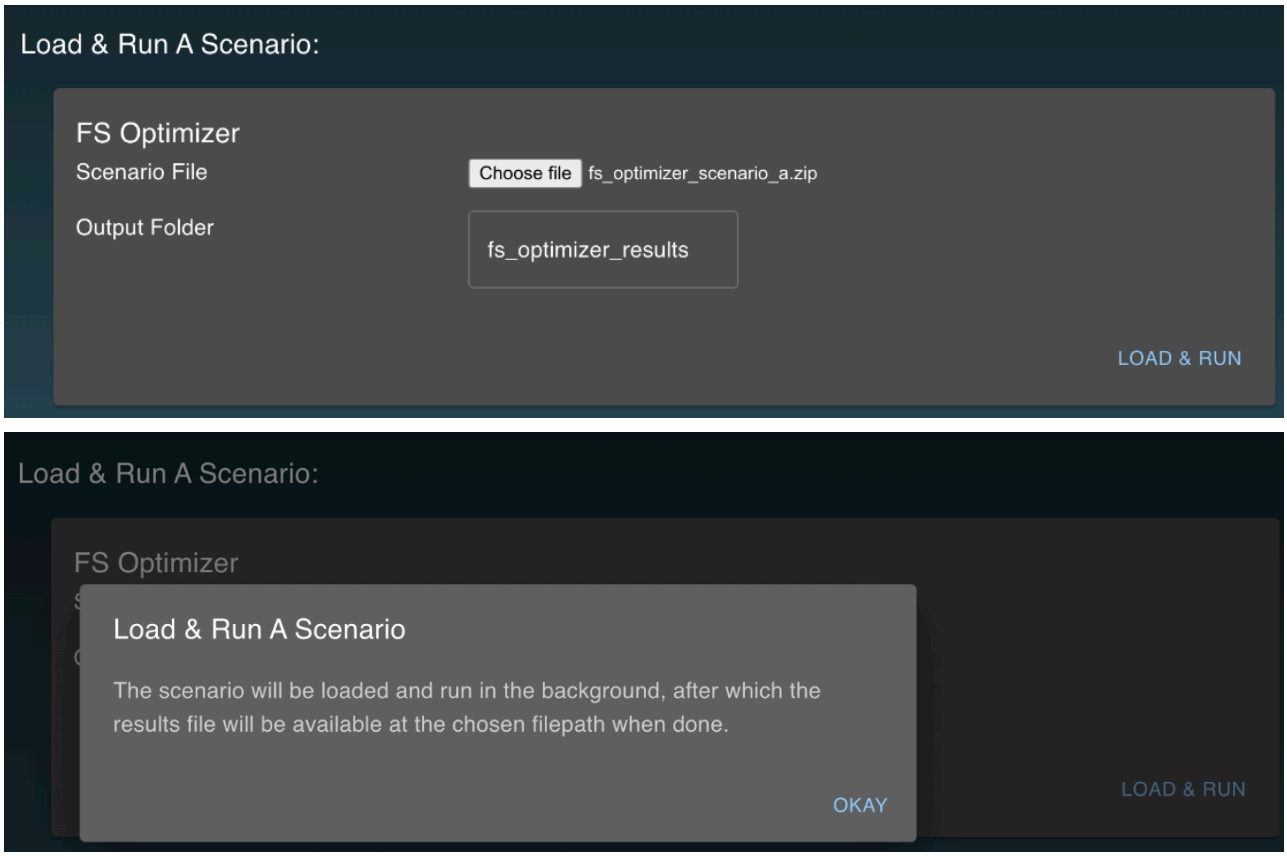


Lastly, set the **parameters**, like the **weight** of parcels and **cost** between nodes.



Load & Run Scenario

Load and run the scenario. Wait for it to load in the **background**. Once it is done, the **results file** will automatically be prompted for downloading.

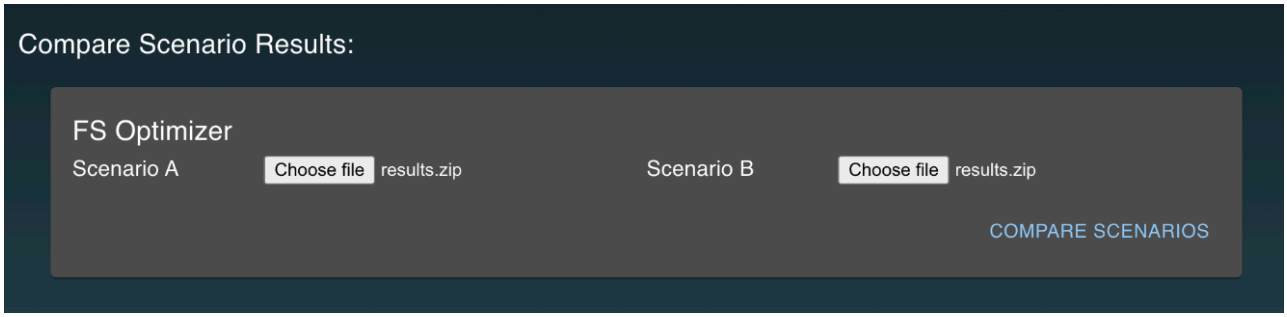


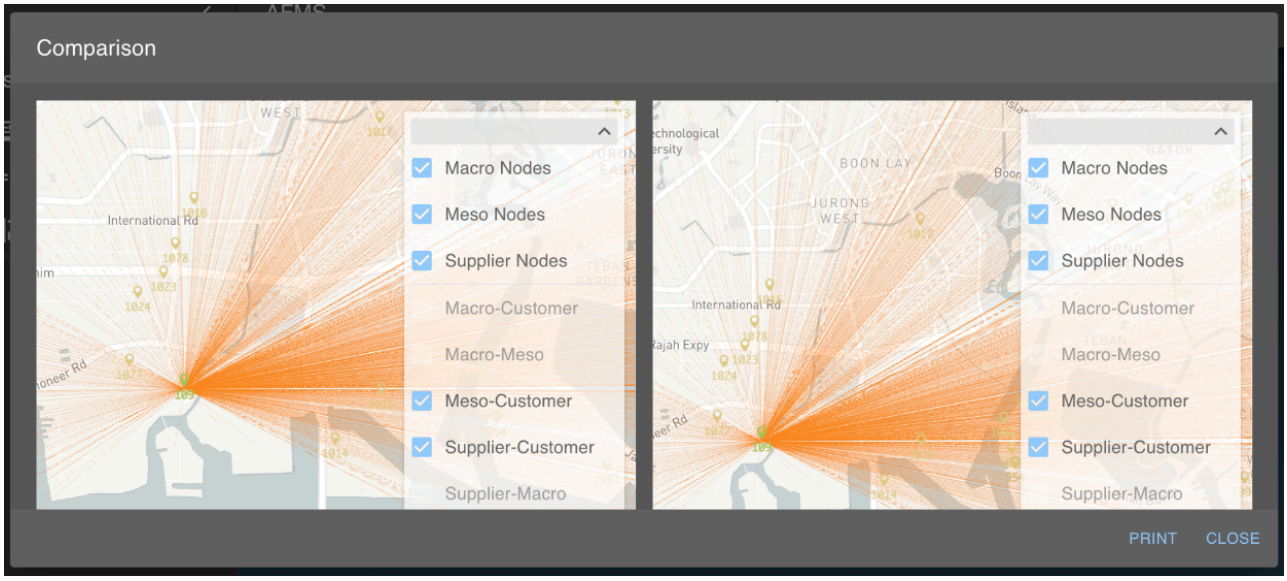
This is what the **console log** of the optimizer looks like.

```
49e+05 |  38.27%| unknown
2023-03-10 01:47:19   169s|     1 |     0 | 12956 |     - |  1405M |   0 | 220k|1757
49e+05 |  38.27%| unknown
2023-03-10 01:47:20   170s|     1 |     0 | 12957 |     - |  1405M |   0 | 220k|1757
49e+05 |  38.27%| unknown
2023-03-10 01:47:21  time | node  | left  |LP iter|LP it/n|mem/heur|mdpt |vars |cons
bound   |  gap    | compl.
2023-03-10 01:47:21   170s|     1 |     0 | 12958 |     - |  1405M |   0 | 220k|1757
49e+05 |  38.27%| unknown
2023-03-10 01:47:32 r 182s|     1 |     0 | 13406 |     - |intshift|   0 | 220k|1757
92e+05 |   0.01%| unknown
2023-03-10 01:47:51   201s|     1 |     0 | 14307 |     - |  1427M |   0 | 220k|1757
92e+05 |   0.01%| unknown
2023-03-10 01:47:51   201s|     1 |     0 | 14344 |     - |  1404M |   0 | 220k|1757
92e+05 |   0.00%| unknown
2023-03-10 01:47:51   201s|     1 |     0 | 14344 |     - |  1404M |   0 | 220k|1757
92e+05 |   0.00%| unknown
2023-03-10 01:47:51
2023-03-10 01:47:51 SCIP Status        : problem is solved [optimal solution found]
2023-03-10 01:47:51 Solving Time (sec) : 201.20
2023-03-10 01:47:51 Solving Nodes      : 1
2023-03-10 01:47:51 Primal Bound       : +1.13169184798496e+05 (4 solutions)
2023-03-10 01:47:51 Dual Bound         : +1.13169184798496e+05
2023-03-10 01:47:51 Gap                : 0.00 %
```

Compare Scenarios

We can now **view the results** and compare 2 **different** scenarios.




Comparison
WEST
International Rd
Macro Nodes
Meso Nodes
Supplier Nodes
Macro-Customer
Macro-Meso
Meso-Customer
Supplier-Customer
Supplier-Macro
BOON LAY
JURONG WEST
Rajah Expy
PRINT
CLOSE

Format of Hub Files

Macro Hub Distribution

- macro_node.xlsx
- (id, existing/candidate)

Example

id	physical_id	latitude	longitude	size	staff	subzone	traffic	psm_month	population	exist	capacity
101	244	1.3266	103.8765	3744	1	9	14388.44	18.78	13661	N	1.00E+08
102	253	1.311795	103.8662	8608	8	75	12216.11	15.345	23233	N	1.00E+08

Meso Hub Distribution

- meso_node.xlsx
- (id, existing/candidate)

Example

id	physical_id	latitude	longitude	size	staff	subzone	traffic	psm_month	population	exist	capacity
1001	1001	1.326207	103.8895	600	0	6	12443.24	16	41990	N	1.00E+07
1002	1002	1.319802	103.9029	1000	0	7	10885	26.54	63853	N	1.00E+07

Micro Hub Distribution

- customer_node.xlsx
- (id, xs, s, m, l, xl, xxl)

Example

id	physical_id	latitude	longitude	subzone	traffic	psm_month	population	xs	s	m	l	xl	xxl	Aggregate
100001	100001	1.304746	103.8264	2	10127.04	15.67819	3054	691	1281	410	93	13	0	Y
100002	100002	1.307246	103.8307	3	9819.711	15.67819	7129	798	1335	385	95	15	0	Y

Origin Distribution

- supplier_node.xlsx

- (id, xs, s, m, l, xl, xxl)

Example

id	physical_id	latitude	longitude	size	staff	subzone	traffic	psm_month	population	xs	s	m	l	xl
1	130	1.362004	103.9015	47220	10	10	9207.321	20.94	20667	7	1865	7	3	1
2	110	1.436646	103.8437	12444	8	127	11335.84	13.14	71407	217	131930	1698	594	371
3	109	1.310654	103.6975	103603	153	217	5020.25	15.67819	27	526952	721495	206870	55173	9201

Format of Supporting Files

Physical Node

- physical_node.xlsx
- list of physical nodes (given id of entity provided by data owners)
- (physical_id, lat, long)

Example

physical_id	latitude	longitude
130	1.362004	103.9015
110	1.436646	103.8437

Physical Node Distribution

- dist_physical_node.xlsx
- Travel distance matrix for each pair of nodes
- (physical_id pairs matrix)

Example

physical_id	130	110	109
130	0	10.49625	23.38562
110	10.49625	0	21.45703
109	23.38562	21.45703	0

Node Mapping

- map_vn_to_pn.xlsx
- a mapping of virtual nodes (unique id for coding purposes) to physical nodes
- (id, physical_id)

Example

id	physical_id
1	130
2	110
3	109

Input Variables

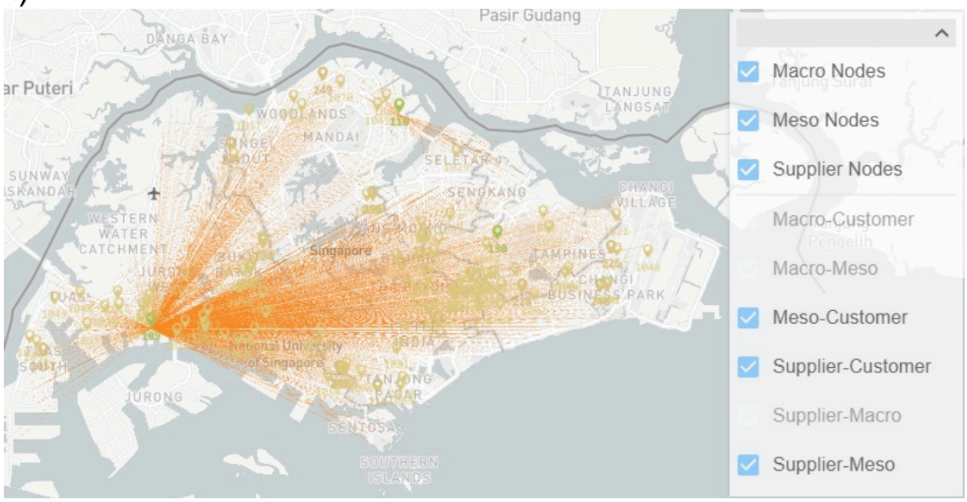
name	description	remarks	
id	unique id representing each node		
physical_id	id given by data owner, for displaying in UX		
latitude	latitude of facility location	CRS (‘EPSG:4326’)	
longitude	longitude of facility location	CRS (‘EPSG:4326’)	
size	area of facility in meters (to calculate the operation cost)	fixed size based on area_meso/area_macro parameters	not in use
staff	manpower at facility (to calculate the operation cost)		not in use
subzone_id	facility located in a particular subzone (to calculate the environmental cost)		not in use
traffic	number of trips generated within the subzone/hex (to calculate the environmental cost)		not in use

name	description	remarks	
psm_month	rental cost per square-meter per month (to calculate the operation cost)	operation cost for the total area per day of the facility. Area is based on area_meso/area_macro parameters	
population	number of people within the subzone/hex (to calculate the environmental cost)	environmental cost incurred for per person per day based on the env_cost parameters	
Aggregate	boolean (Y/N), Y where population is aggregated into hex	default = Y	
xs, s, m, l, xl, xxl	size categories of delivery boxes (to calculate total supply/demand)	transportation cost amount per day based on parcel size parameters and transportation rate parameters	
capacity	maximum capacity of the facility	maximum capacity of the facility	
exist	boolean (Y/N), if Y: node is an existing facility, if N: node is a new candidate facility		

Example Scenarios

Scenario 1

Scenario 1 (default)



Hubs:
Number of macro = 2
Number of meso = 20
Macro area [m^2]
Meso area [m^2]

Relative size of parcels:
XS parcel unit
S parcel unit
M parcel unit
L parcel unit
XL parcel unit
XXL parcel unit

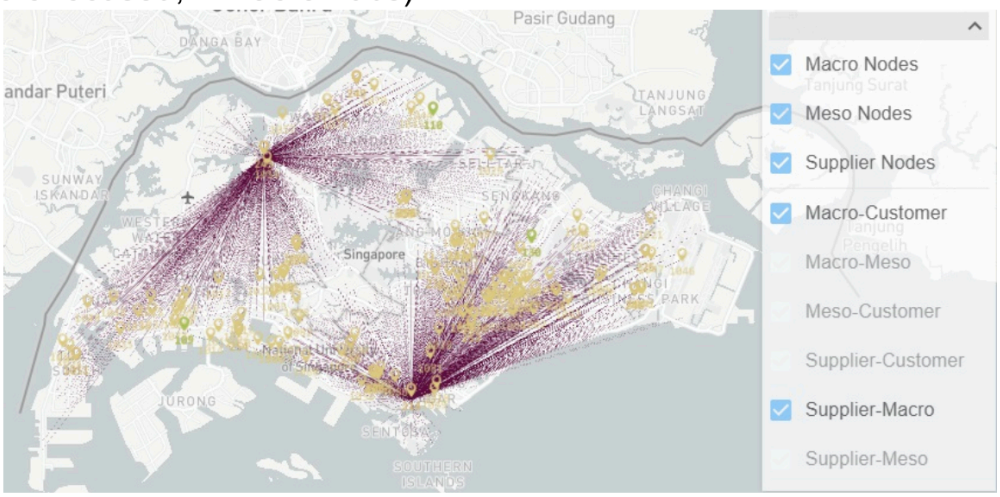
Other Cost:
Environmental Cost (Macro) [$/(person*day]
Environmental Cost (Meso) [$/(person*day]
Environmental Weightage [0,1]

Others (limits):
Solver Run Time [None, >0 seconds]

Transportation Cost:
Origin-Macro [$/km]
Origin-Meso [$/km]
Origin-Destination [$/km]
Macro-Destination [$/km]
Macro-Meso [$/km]
Meso-Destination [$/km]

Scenario 2

Scenario 2 (Macro focused, 2 Macro hubs)



Hubs:

Number of macro = 2
Number of meso = 20
Macro area [m^2]
Meso area [m^2]

Relative size of parcels:

XS parcel unit
S parcel unit
M parcel unit
L parcel unit
XL parcel unit
XXL parcel unit

Other Cost:

Environmental Cost (Macro) [$/(person*day]
Environmental Cost (Meso) [$/(person*day]
Environmental Weightage [0,1]

Others (limits):

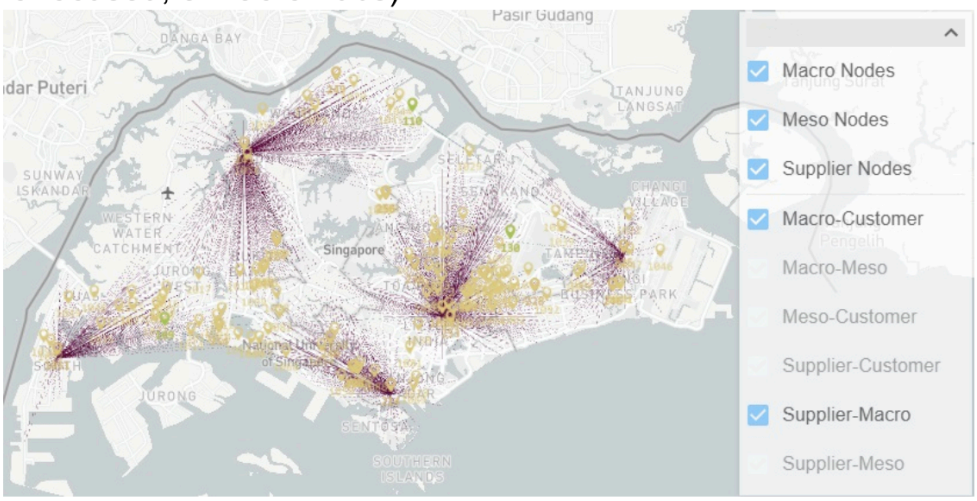
Solver Run Time [None, >0 seconds]

Transportation Cost:

Origin-Macro [$/km] = very small
Origin-Meso [$/km]
Origin-Destination [$/km]
Macro-Destination [$/km] = very small
Macro-Meso [$/km]
Meso-Destination [$/km]

Scenario 3

Scenario 3 (Macro focused, 5 Macro hubs)



Hubs:

Number of macro = 5
Number of meso = 20
Macro area [m^2]
Meso area [m^2]

Relative size of parcels:

XS parcel unit
S parcel unit
M parcel unit
L parcel unit
XL parcel unit
XXL parcel unit

Other Cost:

Environmental Cost (Macro) [$/(person*day]
Environmental Cost (Meso) [$/(person*day]
Environmental Weightage [0,1]

Others (limits):

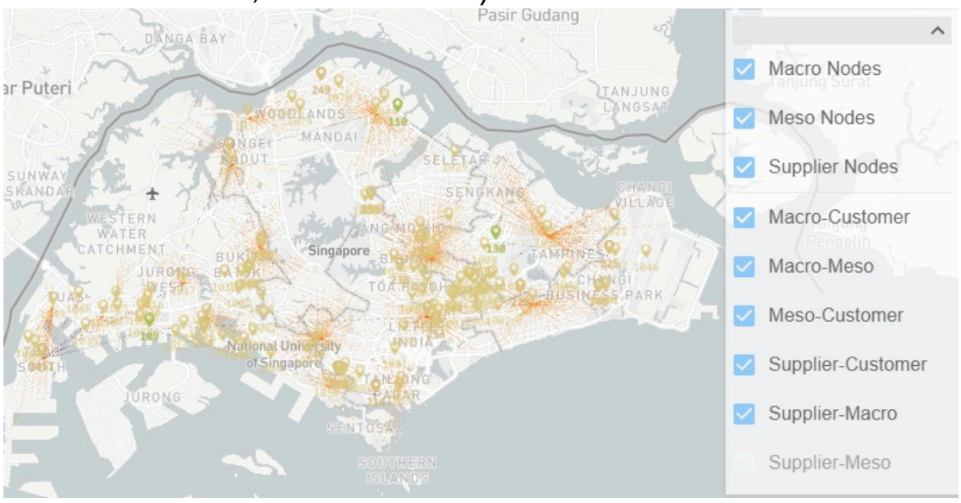
Solver Run Time [None, >0 seconds]

Transportation Cost:

Origin-Macro [$/km] = very small
Origin-Meso [$/km]
Origin-Destination [$/km]
Macro-Destination [$/km] = very small
Macro-Meso [$/km]
Meso-Destination [$/km]

Scenario 4

Scenario 4 (Macro+Meso focused, 5 Macro hubs)



Hubs:

Number of macro = 5
Number of meso = 20
Macro area [m^2]
Meso area [m^2]

Relative size of parcels:

XS parcel unit
S parcel unit
M parcel unit
L parcel unit
XL parcel unit
XXL parcel unit

Other Cost:

Environmental Cost (Macro) [$/(person*day]
Environmental Cost (Meso) [$/(person*day]
Environmental Weightage [0,1]

Others (limits):

Solver Run Time [None, >0 seconds]

Transportation Cost:

Origin-Macro [$/km] = very small
Origin-Meso [$/km]
Origin-Destination [$/km]
Macro-Destination [$/km]
Macro-Meso [$/km] = very small
Meso-Destination [$/km]

6

File Locations

Load & Run Scenario

- `\saved_scenario\scenario_1`
- `\saved_scenario\scenario_2`
- `\saved_scenario\scenario_3`
- `\saved_scenario\scenario_4`

Compare Scenarios

- `\fs_optimizer_results_1\results.zip`

- `\fs_optimizer_results_2\results.zip`
- `\fs_optimizer_results_3\results.zip`
- `\fs_optimizer_results_4\results.zip`

Added Features

Features added to the existing AFMS Project.

Downloading Result Files

To access the files processed in the optimizer backend, we improved the API to include a downloadfile method that allows users to **download the results** stored in the output folder…

Downloading Result Files

To access the files processed in the **optimizer** backend, we improved the **API** to include a `download_file` method that allows users to **download the results** stored in the *output folder*. Then we add a **function** in the frontend to **auto-download** the file once the **optimization process** is completed.

As only **port 80** is exposed, this method enables users to retrieve the files via the frontend.

Improving the API Function

Original `run()` function

optimiser/server.py

```
@app.route('/run', methods=["POST"])
def run():
    ...
    return jsonify({"status": "success"})
```

Improved `run()` function

optimiser/server.py

```
@app.route('/run', methods=["POST"])
```

Adding a Function to Clear Cache

We want to make sure we **do not cache the outputs** of the previous optimization runs, so we **clear the cache** after every call to the function.

optimiser/server.py

```
@app.after_request
def add_header(response):
  response.headers['X-UA-Compatible'] = 'IE=Edge, chrome=1'
  response.headers['Cache-Control'] = 'public, max-age=0'
  return response
```

Auto-Download Function

```
...

  fetch(`${apiUrl}/file?fileName=${fileName}`)
      .then((response) => response.blob())
      .then((blob) => {
          const url = window.URL.createObjectURL(blob);
          const a = document.createElement("a");
          a.href = url;
          a.download = `${fileName}.zip`;
          a.click();
      })
      .catch((error) => {
          console.error(error);
      });
  });
```

Further Improvements

Possible further improvements for the AFMS Project.

Vulnerabilities

Vulnerabilities like sonatype and CVE were detected within the images of the Docker layer.

Error Handling

For invalid input values and scenarios with no solutions, the tool does not display any message.

Vulnerabilities

Vulnerabilities like sonatype and CVE were detected within the images of the Docker layer.

- It could be fixed by altering the versions of packages used, or utilising a different function.
- Some cannot be solved by the above method, so that’s the challenge.

Error Handling

For invalid input values and scenarios with no solutions, the tool does not display any message.

- Users would not be able to know if the results are optimized.
- There is no guide on what nodes are incorrect, or what parameters must be changed to fit the scenario.

MSO: Text Classification

Documentation for additional project done on MSO Data.

Introduction

NLI-based Zero Shot Text Classification

Sandbox

Explore the power of zero-shot classification with this interactive demo using the Facebook BART-large MNLI model, hosted on a Gradio space. This user-friendly interface allows yo…

Data Extraction

Overview

Model Setup

Overview

Complete Source Code

Implementing Few-Shot

Overview

Introduction

NLI-based Zero Shot Text Classification

This is a method to use **pre-trained language models** to classify text **without prior training on specific labels**. They do this by treating the text to be **classified as a premise** and **creating a statement** for each possible label.

This method works really well, especially when using **big pre-trained models** like **BART** and **Roberta**.

Zero-shot Classification Pipeline

The model can be loaded with the zero-shot-classification pipeline.

```
from transformers import pipeline

classifier = pipeline("zero-shot-classification",
                      model="facebook/bart-large-mnli")
```

Classify Sequences into Class Names

You can then use this pipeline to classify sequences into any of the class names you specify. If more than one candidate label can be correct, pass multi_class=True to calculate each class independently.

```
candidate_labels = ['travel', 'cooking', 'dancing', 'exploration']
classifier(sequence_to_classify, candidate_labels,
multi_class=True)

# {'labels': ['travel', 'exploration', 'dancing', 'cooking'],
#  'scores': [0.9945111274719238,
#   0.9383890628814697,
#   0.0057061901316046715,
#   0.0018193122232332826],
#  'sequence': 'one day I will see the world'}
```

Sandbox

Explore the power of **zero-shot classification** with this interactive demo using the **Facebook BART-large MNLI model**, hosted on a Gradio space. This user-friendly interface allows you to input your own **text and candidate labels** to see how the model classifies the text based on your provided categories. The model is highly versatile and can be used for a wide range of applications, including **topic categorization**, **sentiment analysis**, and more.

To use the demo, simply follow these steps:

1. Enter a **text snippet** you'd like to classify in the **"Text"** input field.
2. Provide a list of **candidate labels** separated by commas in the **"Class names"** input field. These labels represent the **possible categories** for the input text.
3. Click the **"Submit"** button to see the **classification results**. The model will return a **list of labels** along with their **respective confidence scores**.
4. (optional) Tick **"Allow multiple true classes"** to calculate each class independently.

Data Extraction

Overview

This Python script imports the necessary `pandas` library and defines a function called `parse` to **extract specific columns** from an Excel file. The main objective of this script is to process a dataset containing **parcel sorting feedback** and count the **number of occurrences** of each unique reporting category.

```
import pandas as pd
```

Functions

```
parse(file_path, sheet_name, column_name)
```

This function takes three arguments:

- `file_path`: A string representing the **path** to the input Excel file.
- `sheet_name`: A string representing the **name of the sheet** within the Excel file to process.
- `column_name`: A string representing the **name of the column** to extract from the specified sheet.

The function reads the **Excel file** using the provided `file_path` and `sheet_name`, then extracts the specified `column_name`. It converts the **extracted column into a list** and

returns it.

```
def parse(file_path, sheet_name, column_name):
    df = pd.read_excel(file_path, sheet_name=sheet_name)
    column = df[column_name].tolist()
    return column
```

Main Code

1. Calls the `parse` function to extract the **unique reporting categories** from the Excel file and stores them in the `categories` variable as a list.
2. Calls the `parse` function again to **extract the subject descriptions** from the Excel file and stores them in the `feedback` variable as a list.
3. Calculates the **number of feedback entries** in the dataset and stores it in the `number` variable.
4. Creates a dictionary called `categories_count` to store the **count of occurrences for each category**, initializing each count to 0.
5. Defines a variable `count` and initializes it to 0. This variable can be used later to **count occurrences** of specific subjects or other elements.

```
categories = list(set(parse("Copy of
parcel_sorting_feedback_2020_2022_updated.xlsx",
"parcel_sorting_feedback_2020_20", "Reporting_Category")))
feedback = parse("Copy of
parcel_sorting_feedback_2020_2022_updated.xlsx",
"parcel_sorting_feedback_2020_20", "Subject_Description")

number = len(feedback)
categories_count = {category: 0 for category in categories}
count = 0
```

Model Setup

Overview

This Python script leverages the `transformers` library to perform **zero-shot classification** of feedback entries. The script loads a pre-trained model (`bart-large-mnli`), defines a function to **classify the feedback** using **zero-shot classification**, and another function to **extract the top categories** for **each feedback entry**. Finally, the script iterates over the feedback list and updates the `categories_count` dictionary based on the **top predicted categories**.

```
from transformers import pipeline
from heapq import nlargest

classifier = pipeline("zero-shot-classification",
                      model="facebook/bart-large-mnli")
```

Functions

zero_shot

```
zero_shot(doc, candidates)
```

This function takes two arguments:

- `doc`: A string representing the **input text** to be classified.

- `candidates`: A list of strings representing the **candidate categories**for classification.

The function uses the **pre-trained zero-shot classification model** to classify the input text and returns a **dictionary** with the **candidate categories as keys** and their respective **classification scores as values**.

```
def zero_shot(doc, candidates):
    dictionary = classifier(doc, candidates)
    labels = dictionary['labels']
    scores = dictionary['scores']
    return dict(zip(labels, scores))
```

top

```
top(text, cats)
```

This function takes two arguments:

- `text`: A string representing the input text to be classified.
- `cats`: A list of strings representing the candidate categories for classification.

The function calls the **zero_shot function** to get the **classification scores** for the input text. Then, it extracts the **top three categories** based on their scores, updates the `categories_count` dictionary, and **returns the top three categories**.

```
def top(text, cats):
    results = zero_shot(text, cats)
    topthree = nlargest(3, results, key=results.get)
    for top in topthree:
        categories_count[top] += 1
```

Main Code

1. Loads the **pre-trained zero-shot classification model** from the `transformers` library.
2. Iterates over the **feedback list** and calls the `top` function for each feedback entry, passing the **feedback text** and the **categories list**.
3. Prints the updated `categories_count` dictionary after processing all **feedback entries**.

To use this script, you need to have a **list of feedback entries** (e.g., feedback) and a **list of candidate categories** (e.g., categories). You can then call the `top` function to process each feedback entry and update the `categories_count` dictionary accordingly.

```
for feed in feedback:
    count += 1
    top(feed, categories)

print(categories_count)
```

Complete Source Code

```
import pandas as pd

def parse(file_path, sheet_name, column_name):
    df = pd.read_excel(file_path, sheet_name=sheet_name)
    column = df[column_name].tolist()
    return column

categories = list(set(parse("Copy of
parcel_sorting_feedback_2020_2022_updated.xlsx",
"parcel_sorting_feedback_2020_20", "Reporting_Category")))
feedback = parse("Copy of
parcel_sorting_feedback_2020_2022_updated.xlsx",
"parcel_sorting_feedback_2020_20", "Subject_Description")

number = len(feedback)
categories_count = {category: 0 for category in categories}
count = 0

import concurrent.futures
from transformers import pipeline
from heapq import nlargest

classifier = pipeline("zero-shot-classification",
                      model="facebook/bart-large-mnli")

def zero_shot(doc, candidates):
    dictionary = classifier(doc, candidates)
    labels = dictionary['labels']
    scores = dictionary['scores']
    return dict(zip(labels, scores))
```

Implementing Few-Shot

Overview

This Python script uses the `spacy`, `classy_classification`, `csv`, and `pandas` libraries to perform few-shot classification of parcel sorting feedback entries. The script defines a function `few_shot` that trains a text categorizer on a small dataset and classifies a given input text. The training data is provided as a CSV file.

Required Libraries

```
import spacy
import classy_classification
import csv
import pandas as pd
import os
```

Function

few_shot

```
few_shot(doc, csv_file)
```

This function takes two arguments:

- `doc`: A string representing the **input text** to be classified.
- `csv_file`: A string representing the **path to the CSV file** containing the **training data.**

The function reads the CSV file, processes the **training data**, trains a **text categorizer model** using the `classy_classification` package, and **classifies the input text**. It returns a dictionary with the **candidate categories as keys** and their respective **classification scores as values**.

```
def few_shot(doc, csv_file):
    df = pd.read_csv(csv_file.name)
    data = {}
    sample_size = 10

    candidate_labels = df['label'].unique().tolist()

    for label in candidate_labels:
        candidate_values = df.query(f"`label` ==
'{label}'").sample(
            n=sample_size)['text'].values.tolist()
        data[label] = candidate_values

    nlp = spacy.blank("en")
    nlp.add_pipe(
        "text_categorizer",
        config={
            "data": data,
            "model": "sentence-transformers/all-mpnet-base-v2",
            "device": "gpu"
        }
    )

    dictionary = nlp(doc)._.cats
    return dictionary
```

Main Code

To use this script, you need to have a **text input** (e.g., `feedback`) and a CSV file containing the **training data**. You can then call the `few_shot` function to process the **text input** and get the **classification results**.

```
few_shot("feedback",
os.path.join(os.path.dirname(__file__),"files/train1.csv"))
```

Format of Training Data

The **training data CSV file** should have the **following format**:

label	text
label_1	feedback_1
label_2	feedback_2
label_3	feedback_3

- `label`: The label column contains the **category** corresponding to each training example.
- `text`: The text column contains the **training examples**.