Deliverable 2.2

Digital Twin Models Library

Deliverable number: (D.2.2)
Authors: Rodrigo Tapia, Ioanna Kourounioti, Dimitra Politaki, Tasos Kakouris
Authors' affiliation (Partner short name): TUDelft, INLE

Deliverable No.		2.2	
Work package No.	2	Work package Title	Digital Twin Model and Simulation Environment
Task No.	2.2	Task Title	Digital Twin Models Library
Date of preparation of this version:		11/11/2022	
Authors:		Rodrigo Tapia, Ioanna Kourounioti, Dimitra Politaki, Tasos Kakouris	
Status (F: final; D: draft; RD: revised draft):		F	
File Name:		LEAD - D2.2 Model Library Update v2.3	
Version:		1.4	
Task start date and duration		01/09/2020 – 31/03/2022	

This document is issued within the frame and for the purpose of the LEAD project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 861598.

Revision History

Version No.	Date	Details
0.8	28/02/2022	Sent to model owners for validation.
1.0	09/03/2022	Sent for internal review.
1.1	25/03/2022	Reviewed document
1.2	07/04/2022	Reviewed document
1.3	12/04/2022	Final document
1.4	11/11/2022	Resubmitted version
2.1	01/08/2023	First updated draft
2.2	29/08/2023	WP2 reviewed version
2.3	05/09/2023	Final version

Reviewers List

Name	Company	Date	Signature
Abdelhadi BELFADEL	IRTX	09/03/2022	
Victoria CRESPO	Panasonic	06/04/2022	

Contents

List of figures

List of tables

Abbreviations

ABM	Agent Based Model
API	Application Programming Interface
DT	Digital Twin
IPR	Intellectual Property Rights
KB	Knowledge Base
LoS	Level of Service
LL	Living Lab
LSP	Logistic Service Provider
M2	Metamodel
ML	Model Library
PI	Physical Internet
PT	Physical Twin
SP	Stated Preference
UCC	Urban Consolidation Centre
UFT	Urban Freight Transport

Executive Summary

This deliverable describes the actions taken regarding the creation of the Model Library (ML) in the context of Task 2.2. The ML assembles a set of open-source, case-specific software applications which will be applied in LEAD Digital Twins (DTs). With the ML the LEAD platform will be able to create different scenarios for the Living Labs and other future uses.

For a model to be included in the ML, it must have three items: the code, the JSON file describing Inputs, Outputs, execution instructions and the model documentation. The publishing of the ML will be in two steps. The first one, was to publish in a private GitHub repository. The second one is be the final publishing when the platform is fully functional, also via GitHub.

The current ML comprises 21 different models, which can be connected in a customized way by the platform, so that different cities can compose their own DT and can test the different LEAD strategies and policies. The model types included are: Optimization models, Demand models, Impact assessment models, Network models and Agent Based models. Moreover, in this work the schema of the ML database is presented.

This deliverable also defines the LEAD Metamodel (M2) that allows for a comparison and selection of individual models, providing guidance for the use of models and, finally, supporting the development of new models. The M2 organizes the models according to their functionality, role within the modelling frameworks and the model type.

Finally, the status of the models is presented. This document is the updated version at the end of the project, with the models effectively applied/used in the Living Labs' (LL) DTs.

1. Introduction

The objective of this section is to provide a brief overview of the D2.2 of the LEAD project by describing its scope and its main objectives and by aligning the task and its subtasks to the rest of the project deliverables and tasks.

1.1 Scope of the deliverable

The scope of this deliverable is to present the outputs of Task 2.2. It assembles a set of open-source, case-specific software applications of the models presented in D1.2, which will be applied in LEAD Digital Twins (DTs). Together these models form an open-source Model Library (ML) that can be applied in future DTs of other cities. The ML comprises different types of models that allows the platform to twin different cities and be able to test the different LEAD strategies and policies. The majority of the efforts during this task has been devoted to model development and adaptation to the LEAD platform.

In addition, this deliverable defines the LEAD Metamodel (M2) which allows comparison and selection of individual models, provides guidance for the use of models and supports the development of new models.

Task 2.2 consists of the following subtasks, which are presented in the following sections of the deliverable:

Table 1: Relationship between Grant Agreement and sections of the deliverable.

LEAD GA Requirements	Description	Section(s) of present deliverable
ST 2.2.1: Establishing inventory of Digital Twin software, describing all characteristics.	Logical structure	2.2
	Delivery and access conditions	2.1, 2.4
ST 2.2.2: Models library	Software Library	2.2, 3.2
	Functional & technical specifications	2.1, 2.2
	Tests & integrity	2.4
ST 2.2.3: Metamodel	Metamodel specification	3.1
	Metamodel application	3.1
ST 2.2.4: Publishing the library	Publish library and monitor use	2.3
	Software use	2.3

Note: *Some of the ML models will be updated and improved during the project lifetime through observations from LLs and DT applications. Therefore, this deliverable will be updated at the end of the project.*

1.2 Relationship with other deliverables

In **Task 1.2** a knowledge base of the library base of reference models was developed. The knowledge and library base will provide the reference models as inputs in the ML.

Task 2.1 will provide the technical requirements for the DTs as well as the connections between the models of the ML developed in Task 2.2.

Task 2.3 develops an API that connects the ML (Task 2.2) to the LEAD Platform. This API and visual interface will give the opportunity to query the ML, choose and apply the model they need.

Task 2.4 takes the model chosen/needed/defined by the user from the ML library and helps the user apply and manage the simulation process.

The **LLs** developed in **WP3** will provide the modelling requirements to develop their DTs and provide a testing instance for the LEAD models.

1.3 Structure of the document

The document is structured as follows:

1. Chapter 2 presents and describes the structure of the ML.
2. Chapter 3 describes the Metamodel and the models that will be included in the ML.
3. Chapter 4 summarizes the deliverable and presents the next steps.

2 Model Library

The model library (ML) is a database of models that consolidates these in a single location for the LEAD platform to deploy and assemble the DTs. Each DT follows a **model-centric approach**, meaning that the DT is composed by one or more models. Each model, with its assumptions, inputs and outputs, has a central role in the definition, calibration and application of the DT. The main benefit of the model-centric approach is that it allows the platform to cope with multiple use cases, from a single ML. Applications of the platform to the 6 LLs will demonstrate the real-world applicability of the LEAD platform.

A model centric approach creates an important design challenge: to harmonise the multiple coding languages, data structures, software and outputs in order for the models to be used. To solve this, connector functions needed to be developed. The connector functions link models in a 1 to 1 way. The connector handles all data transformations for the conversion of the outputs of one particular model into inputs of the next model. This explicit connection of models has the advantage to:

- Allow models that are suitable for a determined modelling purpose to be used without many changes, with the link relying in the connector function. This gives the ML flexibility to include models that are made for other settings or are proprietary (free to use or licensed).
- Allow high modularization of the ML. Each model behaves as a standalone model that is connected to other if needed and via a clear method.
- Allow easy mapping of possible combination of models for a DT. Explicit 1 to 1 connections allow the admin and new users to identify which model sequences are available (and implemented) by querying the ML.
- The data transformation is invisible for the users.
- A unified urban freight ontology is not necessary. The 1 to 1 connection allows the developers to use their own ontology of urban logistics elements, since the input data harmonisation is done at a model level (i.e. through connectors).

Although the ML is intended to have open models and software, proprietary software has been included in the database. This has been done to show and test the flexibility of the approach taken and testing proprietary models that are already in use by the cities (for example Visum), innovative commercial models (such as the route optimization of LMT) or other applications (such as COPERT). For these cases, the code is not available and an API interface can be used.

Currently, the models included in the ML are the ones that the different LLs have selected to twin their use cases. Following a minimum viable product (MVP) approach, these specific models and models' connections have first been developed before any other models and connections are explored. It is important to note that the models of the ML are continuously evolving. This implies that the models available at any given time in the library can be subject to changes, fixes, and new features.

2.1 Model requirements

2.1.1 Model requirements

For any model to be submitted to the model library it is required to include: 1) the code (or API code), 2) a JSON file describing the running parameters and 3) the documentation generated for new users and for platform implementation. The requirements per model are detailed below:

- Code: The code for the model is uploaded to the project's github repository
- JSON File: The JSON model specification file is uploaded to the github repository. The JSON file contains the information required to populate the model related tables of the ML database. The instructions for this file are presented in Annex I: JSON readme.
- Documentation: The documentation consists of two main parts. The first part aims to introduce, describe requirements and the model to the end user and the last part gives instructions on how to run the model to the platform administrator. A suggested template for the documentation can be found in Annex II: Documentation template table of contents.

As introduced above, some connections with proprietary software via APIs have been included in the library to allow some current users/cities to use the LEAD platform with the commercial/proprietary models that they have already implemented. This means that currently the ML includes different types of software, some being open source, other proprietary and other created/adapted for the platform.

2.1.2 Connection between models' requirements

A similar requirement has been established for the model connectors, where the code and the JSON file are required but not the documentation.

- Code: The code for the connector model is uploaded to the project's github repository.
- JSON File: The JSON file is uploaded to the github repository. The JSON file contains the information required to populate the connector related tables of the ML database. The instructions for this file are presented in Annex I: JSON readme.

2.2 Model library database

The ML is a database where the key information of the models is stored and can be queried. The ML consists of 11 tables. The database schema is based on a relational database, where a series of tables are created and related through keys. The ML follows the principles of normalisation, where the attributes and tables are organised with the objective of reducing data redundancy and improve internal consistency of the database. The preliminary database schema can be seen in Figure 1.

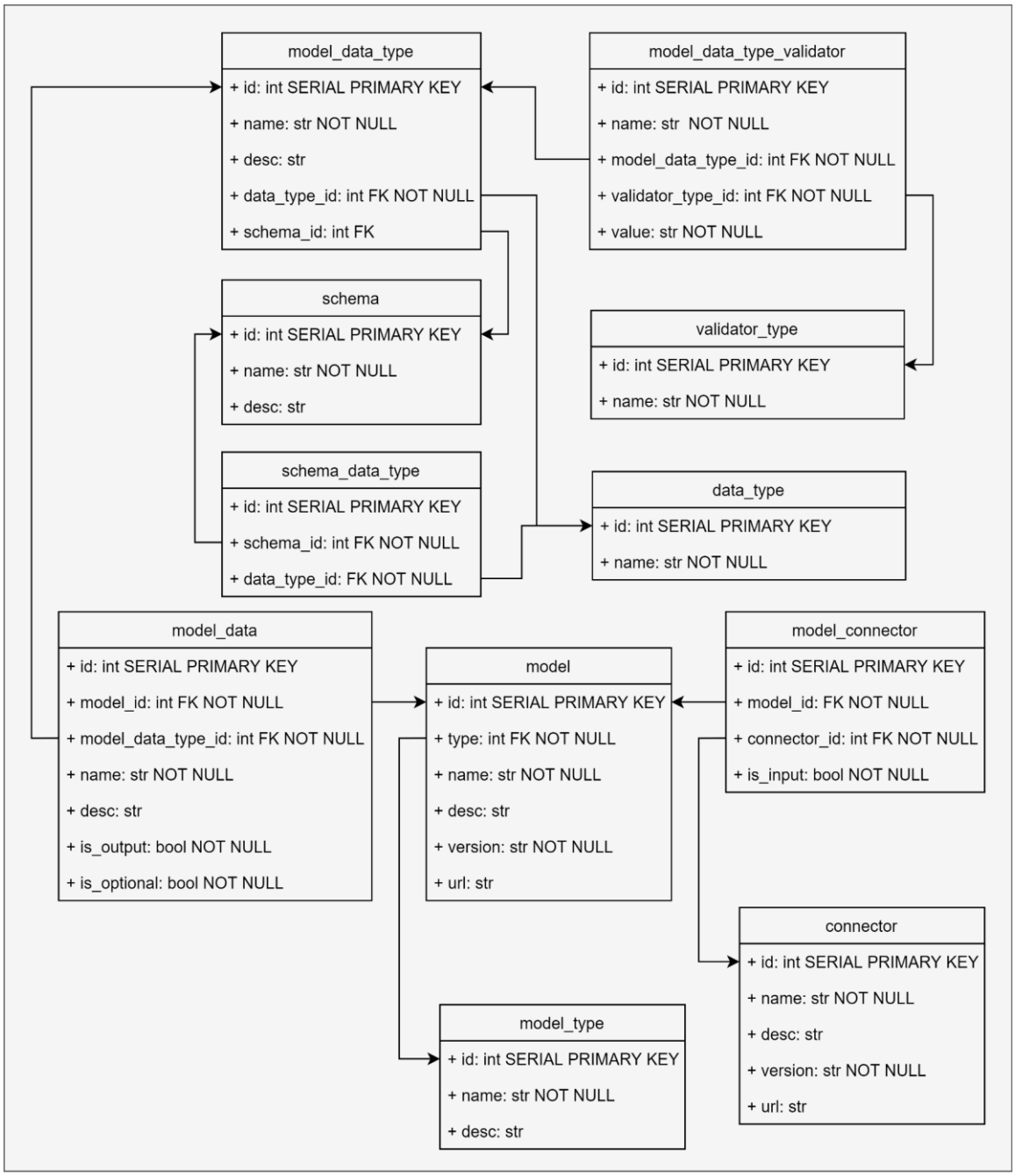


Figure 1: Model Library structure

The tables that consist the model library structure are:

- *Model*: Table that contains and describes the models. Each model version is considered an independent entry.
- *Model_type*: Table that has all the model types according to the definitions from deliverable 1.2
- *Model_data*: Table that relates each input and output with the corresponding model.

- *Model_data_type*: Table that describes the relationship between the model data and their respective types. This characterization consists of the data schema, data type and validations needed.
- *Model_data_type_validator*: Table that contains the data type validation (i.e. it stores the validation requirements for each input).
- *Validator_type*: Table that describes the validators. (To be developed further in D2.4)
- *Schema*: Description of the schemas. (To be developed further in D2.4)
- *Schema_data_type*: Description of the data types of the schemas. (To be developed further in D2.4)
- *Connector*: Table that has all the model connectors.
- *Model_connector*: Table that relates the connector to the models. Each element in this table refers to either the model that is an input or the outputs that will be created by the model.

According to the model architecture (D2.1) and first versions of DDDAS (D2.4) and DSS (D2.3) the ML will be included as a separate entity in the platform accessible through an API. The final version can differ, so the queries and final schema will be presented in future deliverables (2.4).

2.3 Model library publishing

The publishing of the ML is done in two steps. The first step was the submission of the models that are ready for implementation in the LEAD Github[1]. The Github will be populated with all models developed in LEAD, documentation, JSON files. Testing dataset will be provided in the platform by the end of the project, as this is an evolving process.

The LEAD repository is accessible to all partners developing code to be used for the LEAD DT platform. Further work on testing and publishing the ML and the LEAD platform will be described in T2.3 and T2.4. Figure 2 shows the header of the repository and more screenshots can be found in Annex IV.

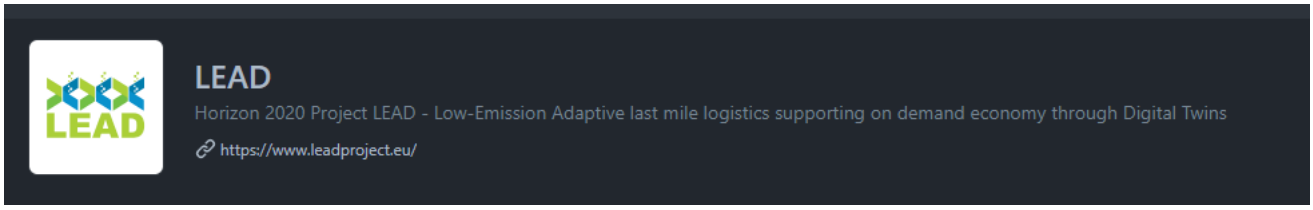


***Figure 2 LEAD Github* screenshot**

2.4 Models library update

The ML is constantly evolving, by adding new models, new connections or by updating the codes of the models. Currently, the process of adding new connectors and models is *ad hoc* due to the early stages of the implementation of the platform and will be formalised in future deliverables. The process consists of:

[1] https://github.com/Horizon-LEAD

1 Local development and verification
2 Uploading and versioning of model (or connector) code to an accessible location (github repository detailed in section 2.3) complying with all the requirements from section 2.2.
3 Integration of the model to the platform based on its metadata, source code, and container image as stored in ML
4 Perform tests of the model within the LEAD platform.

3 Models

3.1 Metamodel

A Metamodel (M2) has been built to categorise and classify all the models in the ML. Each specific instance of the M2 will be a configuration of a DT of a LL, relating the models and connectors. This means that the M2 provides information on the valid connections between models, together with a view of their possible roles in the DT. The M2 provides a common framework of otherwise heterogeneous models and allows functional and theoretical consistent DTs.

3.1.1 Metamodel description

From the model inputs and outputs, based on the template filled in by each model owner, a M2 has been elaborated. The M2 links all the individual models in a logical sequence in line with the model scope and role within a broad modelling framework. By having a logical and theoretically consistent framework a new user can have an idea of the roles of each model and their possible connections between them. Each instance of this meta-model is the DT configuration of a specific LL city in LEAD. These instances are shown in Annex III: Living Labs DT scenarios.

The M2 is divided into three main types according to their role in the DT: KPIs, tools and twinning models.

The KPI models are models that convert the outputs of other models into relevant KPIs that have not been calculated before. The models in this type are COPERT (UPM), Noise (UPM) and Electric emissions (UPM).

The tools are models that support the LLs in the development of their DT and are composed from STAR (UPM) and Choice Experiment Design (MOLDE).

The twinning models are the models that aim to represent the urban logistic setting of the cities. They are divided by their resolution level (Zone or Parcel/Shipment level) and by their scope (Strategic, Tactical and Operational (Tavasszy & de Jong, 2014)). The models at a strategic level that work with zones the ML contains Echelon Model (ZLC), Charging locations (INLE) and Population synthesizer (IRTX). The tactical models that work at a zone level are the Parcel Generator (TUDELFT) and Shipment Model (TUDELFT). The tactical models that work at a parcel (or shipment) level are the Parcel Synthesizer (IRTX), Parcel Market (TUDELFT), Parcel Tour Formation (TUDELFT), Shipment Model (TUDELFT), MATSim (IRTX) and Discrete Choice Modelling (MOLDE). Finally, the models that work at an operational and parcel (or shipment) level are UDR (INLE), JSprit (IRTX), Network Assignment (TUDELFT), Visum (BKK), NetLogo (MOLDE)

and Route Optimization (LMT). The aforementioned models are explained in detail in the next section.

The model types follow the categorization from D1.2 (Tapia, Kourounioti, 2020):

- Optimization models deals with the minimization of a cost related function.
- Network models deal with routing and allocation to road networks.
- Impact assessment analyse and measure the impact of different policies (includes externality measurement).
- Agent Base Models (ABMs) simulate individual agent's behaviour.
- Demand models focus in obtaining and analysing preferences of agents (including demand synthesis).

In total, the ML consists of 4 externality and impact assessment models, 2 optimization models, 5 network models, 5 ABMs and 5 Demand models (3 demand synthesis and 2 choice-model related). No impact assessment models are included in this version of the ML because it is out of the scope of the DTs of the LLs. Figure 3 shows the version of the M2 with the connections needed to develop the DT of the LEAD LLs that were in place at the first release (April 2022). The M2 is being updated constantly as new models or new connections are introduced. Figure 4 shows the current version with all the adjustments

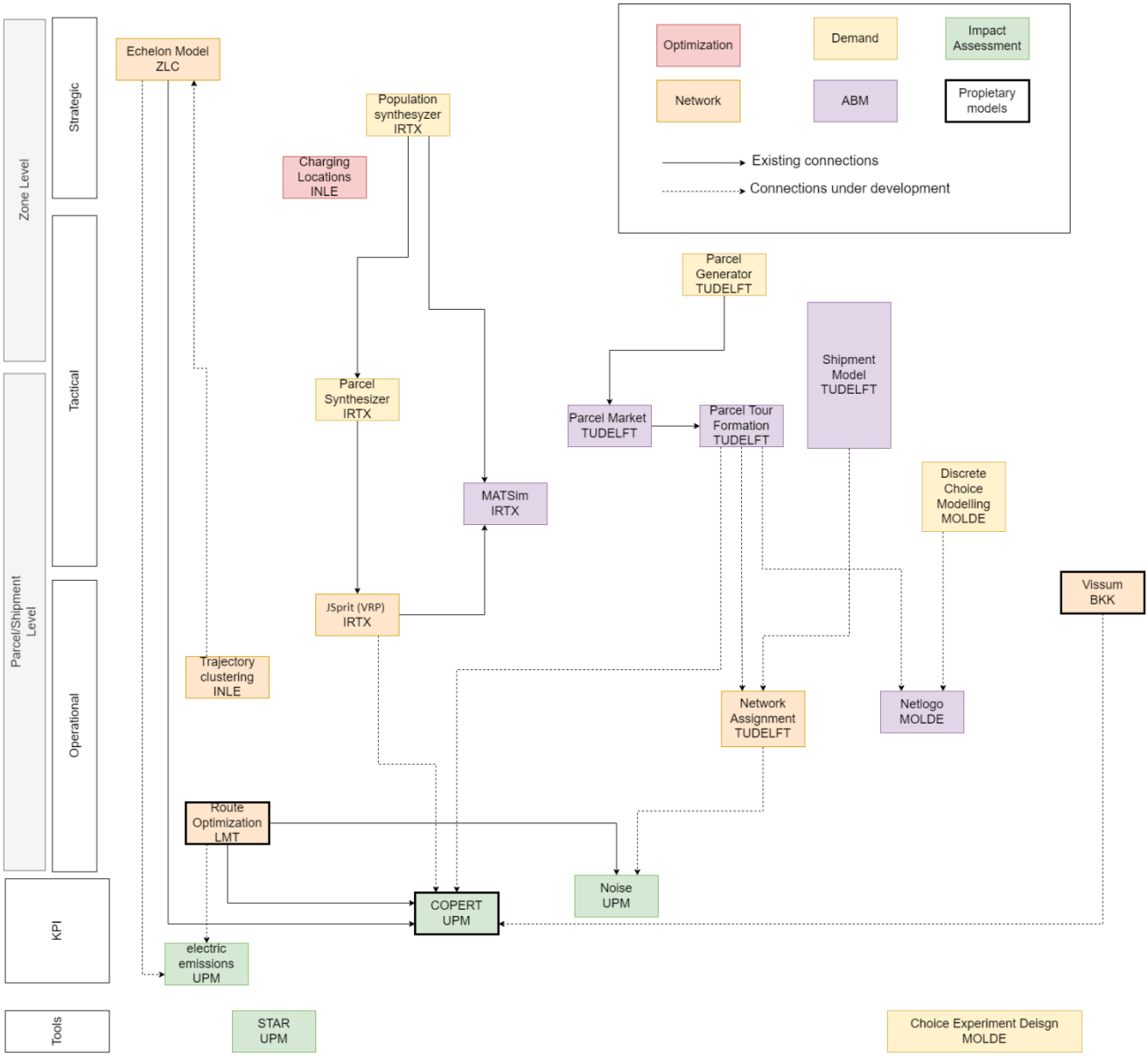


Figure 3: Metamodel for LEAD project at first release

The metamodel has been updated according to the effective development of the individual DTs of the LLs (Deliverables 3.4, 3.6, 3.8, 3.10, 3.12, 3.14) with the new version shown in Figure 4. Some models changed, for example trajectory clustering model was renamed (and rescoped) to UDR and NetLogo not included in the model library. Moreover, In the new metamodel some of the planned connections were not considered necessary to implement at this stage (such as the connection with Netlogo), while other connections that were not present in the M2 of Figure 3 were added, especially for KPI measurement (such as UDR with COPERT).

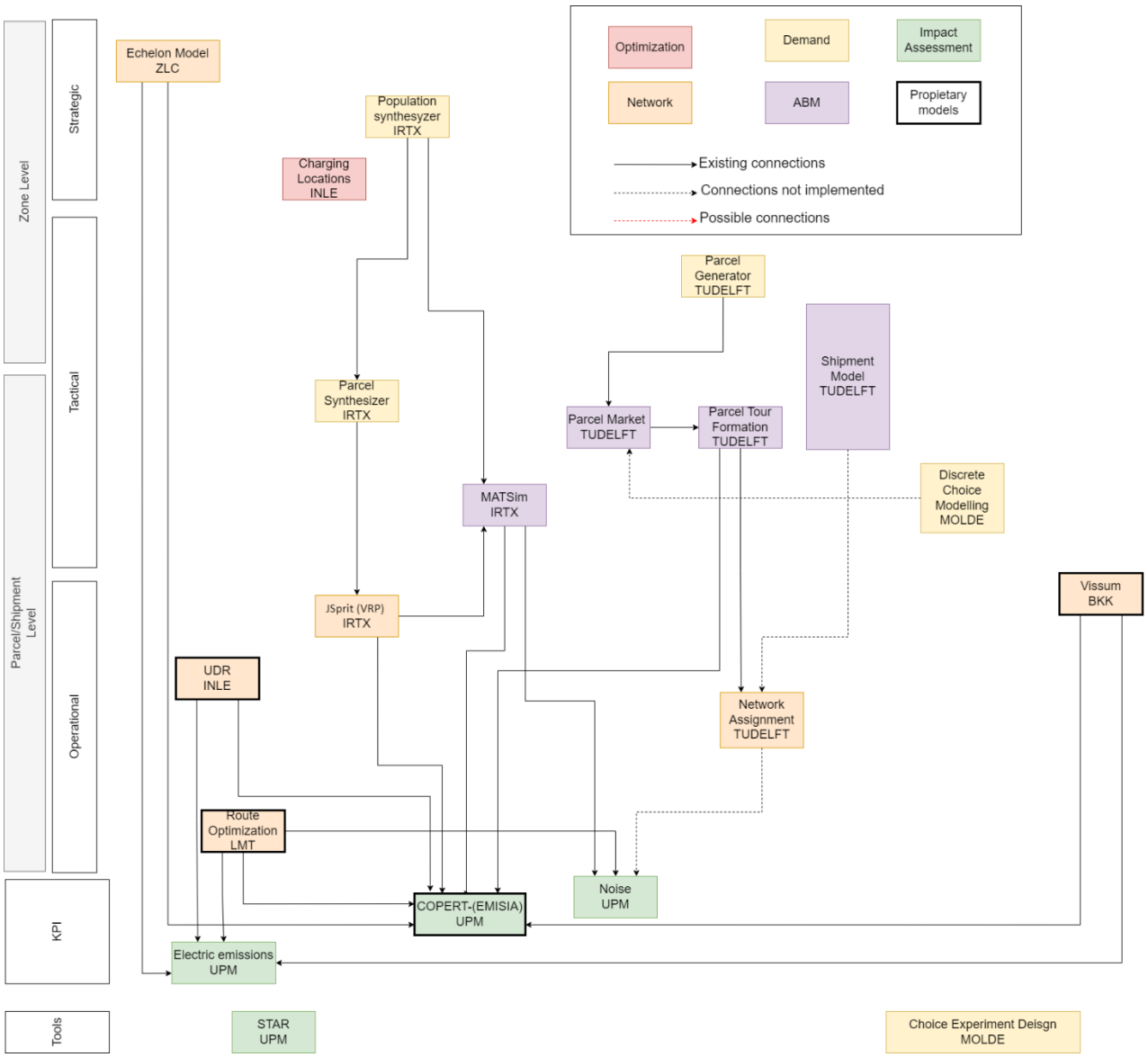


Figure 4: Current version of metamodel

The characteristics of the models in the library and the modular approach allows having more connections than the ones showed above, according to theoretical understanding of the functioning of these models. Some of these possible connections are mapped in Figure 5, but they have not been implemented due to the fact that these are critical to implement the DT of the LLs. However, these possible connections are shown in order to illustrate the possibilities of cross-use of models from different model owners,the flexibility of the current ML and to point out possible developments.

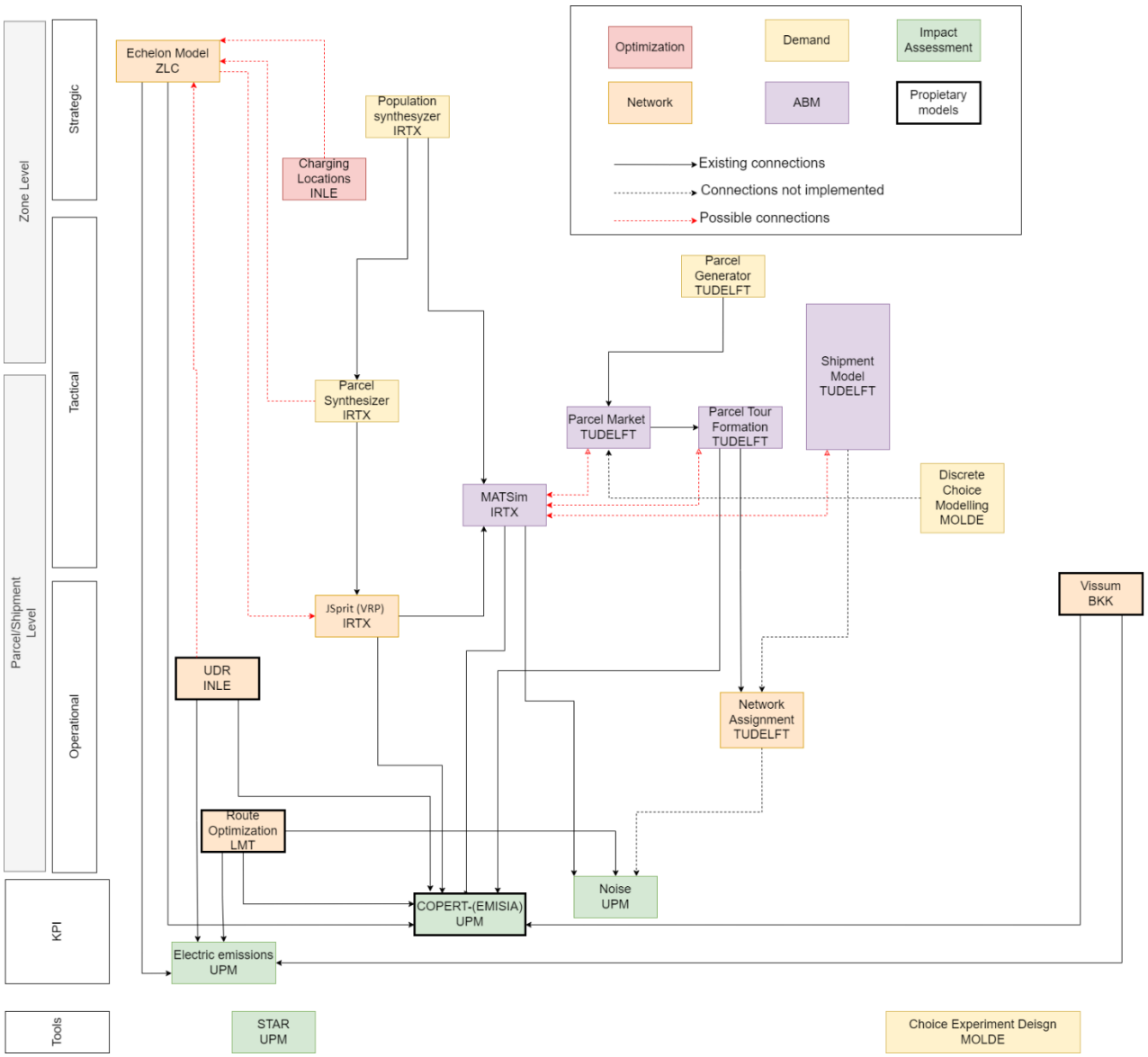


Figure 5: Possible expansions of Metamodel (red arrows)

3.1.2 Mapping the scenario context for applications

One of the objectives of the M2 is to help in designing the scenarios of the LLs (i.e. the selection of models to be included in the twinning of a LL) and support the development of new models (i.e. identify missing models). The use of the M2 starts when a LL sets their desired objectives and defines the KPIs (or other outputs) to be measured in their DT based on the available model outputs. With the KPIs requirements, a suitable model is searched through the ML. If no model is found, a gap in the ML has been identified, and the process to look for and include a new model can be explored.

If a suitable model is found and selected, the inputs required to run the model are identified and compared with the data sources available to the user. In case the data is enough to run the model, then a model can be implemented in the DT. If not, another related model can be selected through analysis of the M2 or querying the ML and the inputs for this new model are analysed. Figure 6 shows this process.

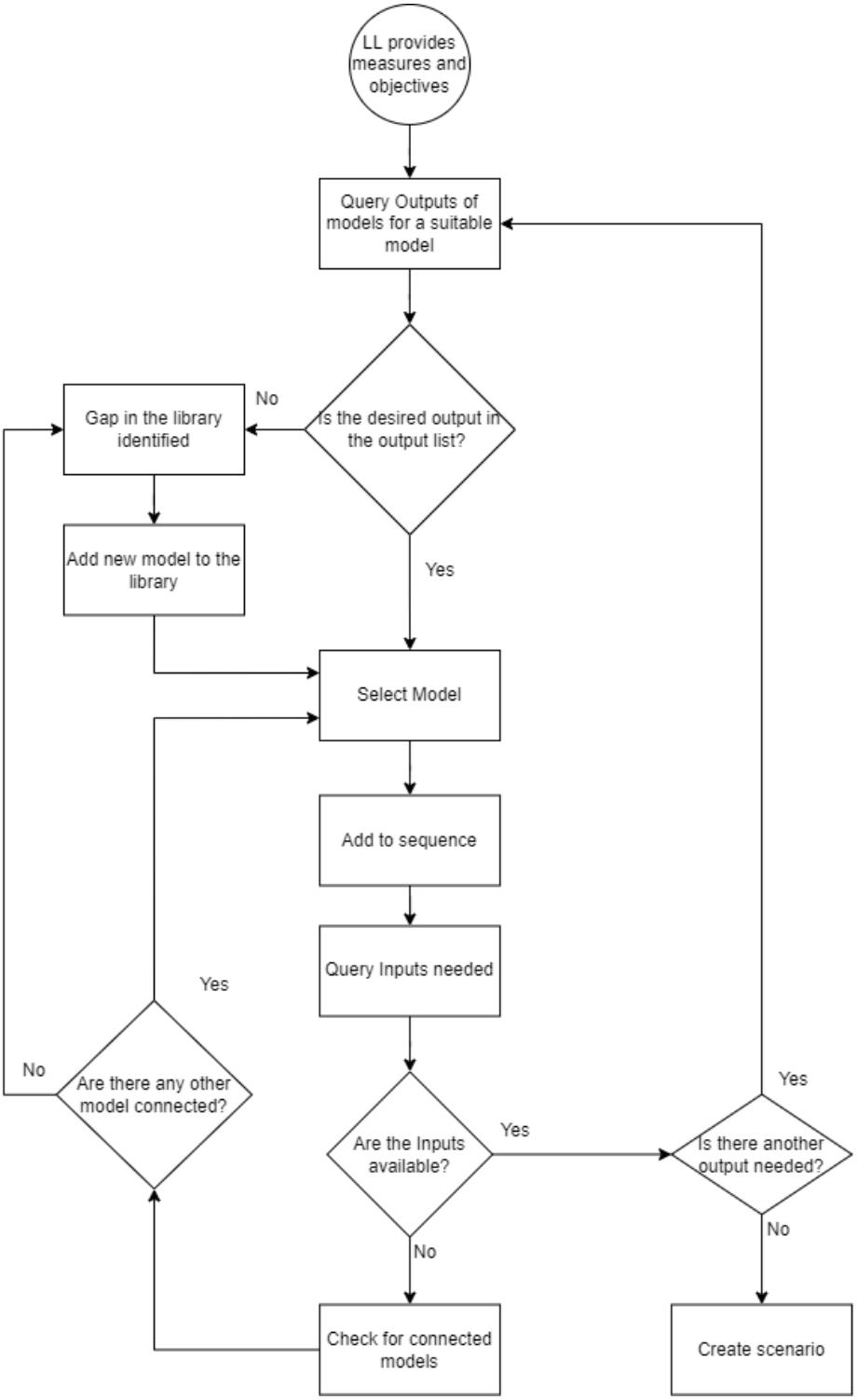


Figure 6: Model sequence selection

3.2 Inventory of models

3.2.1 Description of models

In this section, all the models of the library are briefly described. For the models that are already in the library, more details can be found in Annex IV: , where the complete documentation of the models is found. Note that some of these models are not yet fully developed, and will evolve further during the project. Moreover, as the implementation phase advances and new needs arise from the LLs, the models will change to adapt to them. The final planned update of this deliverable will contain the final state of the models.

3.2.1.1 *VISUM – BKK/SZE*

VISUM is traffic planning software designed for transport planners to empower cities with developed macro transport model. The Centre for Budapest Transport Plc. Worked out an independent, continuously maintained and regularly updated transport model for analysing the future transport investments in the capital and the conurbation. By creating the base model, it can be ensured that the same model – regardless of the engineering firm – be the basis in every development project, and the data received during modelling can be used for loopbacks and comparison.

The modelling process based on the 4-step modelling methodology:

1. Trip generation: based on statistic information (population, traffic attractive facilities – schools, stores, workplaces etc.) the origin and destination traffic volume are generated for each zone.
2. Trip distribution: the distribution of the trips between the zones based on the distance and other parameters
3. Mode choice: Based on several parameter, like car ownership, access time and cost by different modes, etc. The choice of the mode is happening in this step
4. Assignment: search for the best route for each zone relation and each mode

The results of the model are the loaded network and many statistic parameters.

3.2.1.2 *Charging Stations – INLE*

The charging station model aims at finding the best locations to position the EDV charging station. It is essentially a location-allocation problem that typically deals with provisioning of resources between facilities based on historic demand. The generic p-median solver for mobility driven location-allocation algorithm is used to find the best places among the SONAE's stores to locate the charging stations, considering mobility information and delivery needs.

The p-median approach is one such model that aims to minimise the total demand-weighted distance between the demand points and the facilities. This NP-Hard problem aims to locate 'p' facilities (charging stations) to serve 'n' demand (SONAE stores), by minimising the total demand-

weighted distance between the facilities and the demand. To calculate the costs of journeys from a demand point to a p-facility this tool relies on access to Open Route Service[2].

The parameters needed to feed the model are geographical information, vehicle specifications such as vehicle autonomous, delivery information such as distance and time, store and delivery locations and its output are the charging station locations in geographic coordinates (latitude, longitude).

3.2.1.3 *UDR – INLE*

To be able to estimate the number of Electric Delivery Vehicles (EDVs) as well as kilometres and CO2 required to cover specific parcel delivery demand, INLE has developed UDR digital model based on the Capacitated Electric Vehicle Routing Problem with Time-Windows (e-C-VRP-TW). The model can run on a dataset of historical data for a 6-month period and estimate, for each dispatch window, each day and month, the required supply resources to cover delivery demand for a single depot. In this way, by assuming that the solution of e-C-VRP-TW (i.e., vehicle routes) is how the delivery demand is covered in the real world, the model can simulate how the last-mile network operates. Since the e-C-VRP-TW is an NP-Hard problem, the model can solve multiple instances of e-C-VRP-TW for small capacity (i.e., low volume) orders, such as in Porto LL UC3, with relatively 'small' solution spaces. The digital model is 'closely integrated' with the Digital Twin platform, closed-source, and built upon several open-source packages like OpenTripPlanner routing engine, OpenStreetMap datasets and makes use of two connectors, one to COPERT and one to the Porto-adapted EVCO2 digital model.

3.2.1.4 *Synthetic Population – IRT SystemX*

The synthetic population model represents a methodology (Hörl and Balac, 2021) for creating digital replicas of households, persons, and their daily activities of a territory. While the model has originally been developed for the Île-de-France region around Paris, it can be executed solely based on open and publicly accessible data sets (including population and enterprise census data, income surveys, household travel surveys, and others) for any place in France. Furthermore, it has been extended to a range of other use cases such as São Paulo and California, which show its generalizability and justify its inclusion in the generic modelling library. Additionally, to the synthetic travel demand of a territory, the model provides supply information such as a road network and public transport schedule, both based on open data sets. In the pipeline, the model is used to create a realistic representation of the local population and mobility system to assess the impact of the addition of novel logistics services to the local context.

3.2.1.5 *Parcel synthesizer – IRT SystemX*

The parcel synthesis model has been developed within LEAD to produce a digital representation of the parcel demand of a territory, represented by the number of parcels received by individual households on an average day. The synthesis process is disaggregated and based on generating a realistic number of parcels for any household of a synthetic population on the basis of its sociodemographic attributes. The model can be flexibly fed with marginal information sociodemographic dimensions such as age classes, socioprofessional classes or household sizes, depending on the data available for the specific use case. Within the LEAD platform, the model is

[2] *https://openrouteservice.org/*

used to generate the input for other models which are able to assess the impact of the movements generated from delivering these parcels.

3.2.1.6 *Jsprit – IRT SystemX*

Jsprit[3] is a flexible tool to solve classic vehicle routing problems, in which a to be determined or predefined number of vehicles is placed in a depot and optimized to perform a number of jobs in the transport system while minimizing a cost metric such as the total distance driven, or the monetary cost of the operator. Specific formulations of the VRP (Vehicle Routing Problem) can be solved with Jsprit, such as the VRP with Pickups and Deliveries (VRP-PD), or a version that is constrained by time windows for each delivery (VRP-PD-TW). Furthermore, given a list of vehicle types with different cost characteristics, Jsprit is able to perform a join route and fleet composition problem. In the LEAD platform, the model can be used to find the optimal routing for a fleet of vehicles and a set of tasks to be performed. The model is implemented using a generic interface such that, in fact, similar VRP solvers such as VROOM[4] can be compared in terms of runtime and solution quality for different use cases.

3.2.1.7 *MATSim – IRT SystemX*

MATSim[5] (Horni et al., 2016) is a general-purpose multiagent transport simulation framework. It consists of a traffic simulation component, which simulates the movements of people and vehicles in detail on a fine-grained road network, and a behavioural component which simulates various decisions such as mode choices in response to the observed traffic conditions in the transport system. It is hence able to study the impact of new logistics solutions on the local population in terms of emissions, noise, and congestion (for all of which analysis extensions are available) and to observe the complex interplay in terms of adapted travel behaviour given the novel elements in the transport network. In the LEAD platform, MATSim is used to simulate the movements of personal and logistics vehicles in detail in order to aggregate system-level measures such as the total distance travelled, congestion, and other KPIs.

3.2.1.8 *Route Optimization – LMT*

Last Mile Team Route Optimisation is a proprietary model of LMT. It is an integral part of the Last Mile Digital Platform, a commercial asset in TRL6/8 used in LEAD as the underlying technology of Madrid LL.

At the core of the model are different road transport specific Artificial Intelligence algorithms plus a comprehensive, in-house developed, IP registered, technology stack. The core enables to build Digital Twins of urban logistics networks in cities, urban and peri-urban areas considering all available resource, people, their working hours, customer delivery windows, municipal, regional, national and European rules and legislation in diverse road-transport related regulated fields as vehicle types, low emission zones, other geo-fenced vehicle, time or otherwise restricted areas, specific transport regulations etc. to ensure the best possible optimization to any specific need not

[3] https://jsprit.github.io/
[4] *http://vroom-project.org/*
[5] https://matsim.org/

only from an economical point of view but also from an environmental sustainability and driver working conditions and well-being points of view.

3.2.1.9 *Discrete choice model – Molde*

Analytical tool to model agents' preferences, assess adoption/acceptance/reaction, optimise stimuli selection and identify compromise solutions. Discrete choice models (DCM) are econometric models aimed at analysing the behaviour of a decision-maker when choosing among different (discrete) options. Both stated preference (SP) and revealed preference (RP) data can be used as input. This model will be used to investigate stakeholders' preference heterogeneity in order to forecast their choice behaviour related to the LL5 scenarios.

3.2.1.10 *Choice Experiment Design- Molde*

This model represents an analytical tool that will provide the building blocks to design the choice experiments included in the SP survey to acquire the necessary data to be used for DCM. In the LL5 case, a SP survey will be administered. This will help forecasting agents' decisions, by asking respondents to select the most preferred option within a choice set in hypothetical situations given a specific set of conditions created thanks to a specific choice experiment design (CED). In more detail, alternatives, included in a choice set and shown to respondents, represent goods or services that differ from each other based on given attribute levels. CED allows constructing suitable choice tasks by appropriately combining attribute levels.

3.2.1.11 *Parcel Generation – TU Delft*

The parcel generation module generates the demand from population values and parameters. The demand is randomly generated and aggregated in zones. As a result, we get a parcel demand per zone, together with the carrier that will fulfil the delivery.

The parameters needed to feed the models are the average parcel generation rates per household (for B2C and C2C segments) and employee (in the case of B2B). Moreover, courier market shares and the proportion of L2L (i.e. parcels that are sent within the same city) parcels are needed.

With that data, and assuming uniform generation per household the number of parcels is generated per area. Randomly, those parcels are distributed per courier according to their market share.

For the parcels that are L2L, the origin is generated randomly following the population distribution of the urban area. For the other parcels, the origin is assumed to be the closest depot from the courier in the area.

3.2.1.13 Parcel Market – TU Delft

The parcel market has as an objective to define the fulfilment of the parcels to their final destination. The fulfilment of the parcels is the process in which each of the parcels gets to its final destination. The fulfilment can be done via one or more logistic service providers (LSP), including a crowdshipping platform. As a result, parcel origin and destination are broken down into individual trips between connection points by one carrier. The output of this model is a list per LSP on how

each parcel is transported from the origin to the destination, including pick up tours for the L2L shipments.

The first step is to generate the delivery networks of the LSP and to generate the transhipment points where the parcels can change between LSPs. Depending on the density of the connections between the LSPs, a hyperconnected network can be simulated. Once the steps for the delivery are established, the delivery is divided into each individual trip by each carrier. Each trip is then aggregated by LSP in order to have the individual delivery tasks.

3.2.1.14 Parcel Scheduling – TU Delft

The parcel scheduling model receives a list of parcel origin and destination per Logistic Service Providers (LSP) and generates the sequence of stops to fulfil them. The parcel origin and destination are separated depending on if they are consolidated (hub to hub) or delivery to the final customer, allowing each type to have its separate vehicles.

The first step is to cluster the deliveries that are closer together to complete full truckloads. The parcels that are not included as full truckloads are then merged with the parcels with other zones close by. In those clusters, different vehicles are assigned to those tasks with the closest zone to the depot as a first stop. As a result, the tours are formed, and the vehicle km needed for the delivery are estimated.

3.2.1.15 Shipment model– TU Delft

The parcel shipment model generates the shipments for products that are not in the parcel market. The models use synthetic population generated through census data, to generate individual tours of shipments. This model is yet to be added into the library.

3.2.1.16 Network allocation – TU Delft

This model takes the tours (zone stop list) and allocates them to the network. The links between the zones are centroid to centroid and are consistent with the Skim matrices used in other related models. As a result, the links are populated with the delivery vehicles and the routes can be obtained. As an output, a shapefile with the vehicle counts can be obtained.

3.2.1.17 *COPERT – EMISIA added by UPM*

Computer Program to calculate Emissions from Road Traffic – COPERT is the EU standard vehicle emissions calculator. COPERT was developed by EMISIA and added to the library by UPM. It calculates emissions and energy consumption for a specific country or region. The model includes expressions for cold start and evaporative emissions and accounts as far as possible for national variations in such parameters as vehicle parking duration, vehicle age, driving patterns, fuel composition, and climate. Emissions are estimated using average speed-dependent factors for hot emissions.

3.2.1.18 Electric vehicle GHG emissions estimation – UPM

This methodology to estimate the greenhouse gas emissions (GHG) from electric vehicles is associated to the electrical mix used to produce electricity. The method estimates the electric

vehicles CO2eq emissions based on the sum of the mean electricity consumed by each vehicle (kWh) and the carbon intensity -the CO2eq emissions per energy unit- of the electricity generation system of each region or study area (gCO2eq/kWh). For instance, the greenhouse gas emissions intensity of power generation in the EU has been continuously decreasing over the last three decades, in 2020 the EU-27 GHG emissions intensity was 230.7 gCO2eq/kWh6 (EEA, 2021).

Carbon intensity figures for each country or region are generally updated on an annual basis. In addition, they are updated whenever circumstances require. These emission factors take into account only direct emissions associated with electricity production and do not include indirect emissions associated with the construction of generation plants, transport of fuels, maintenance, etc.

3.2.1.19 *STAR – UPM*

The Sustainability Tool for the Appraisal of Transport Projects (STAR) aims to assist decision-makers and policymakers to select the most suitable transport alternative from the sustainability viewpoint. First, STAR identifies a set of sustainability criteria suitable for the transport alternatives that are to be compared. The criteria are grouped into different components of sustainability (economic, social and environmental). Each criterion is quantified for each alternative. Criteria can be measured through monetary units, other units, or qualitative scales. Specific weights are assigned to each criterion based on the project context and evaluation judgments –preferences for pairwise comparisons of sustainability criteria– by decision-makers and experts in the field using the Delphi and REMBRANDT techniques. Finally, STAR establishes a procedure to assess the Sustainability rating of each alternative compared to the baseline scenario. This step may consider the impact of uncertainty through sensitivity analysis. The tool aims to help decision makers adopt decisions that optimize sustainability.

3.2.1.20 *NOISE POLLUTION MODEL – UPM*

The noise pollution model in LEAD library is a simplified methodology that will determine the percentage of residents in the study area exposed to noise emissions. The methodology consists of two interlinked models.

On the one hand, there is the noise emissions model from the vehicles used for the delivery of goods. On the other hand, there is the sound propagation model. The results of the noise emissions model will feed the propagation model whose results will be the noise incidence level on the facades of the residential buildings in the city area.

Both models (emission and propagation) will be simplified for the sake of simplicity of implementation but will be sufficiently consistent to allow the assessment of changes in the noise environment, depending on the types of vehicles used, the number of trips, the routes taken and the speed of the vehicles during these trips.

[6] *https://www.eea.europa.eu/data-and-maps/daviz/co2-emission-intensity-9/#tab-googlechartid_googlechartid_googlechartid_googlechartid_chart_11111*

It is desirable that every Living Lab undertakes a small measurement survey to characterize the emissions of each type of vehicle they are considering, so that the noise emissions model can be customized or adjusted accordingly.

The sound propagation model will only consider those vehicles circulating on the street closest to each façade segment, which seems quite reasonable, considering that the emissions of delivery vehicles are small, that its flow is very reduced compared to regular traffic, and that in an urban environment, the sound is strongly confined by buildings acting as barriers.

The results of the methodology are a figure that will reflect the percentage of residents exposed to noise emissions. In order to do this, it will be necessary to determine a reference threshold indicating what noisy means, since by addressing only such a specific noise source, the reference values commonly used for traffic noise might be meaningless. The threshold shall be selected in such a way that the changes resulting from this pollutant can actually be assessed.

Alternatively, a table with noise ranges with 5 decibel intervals and the percentage of people in each range can be presented as an output.

3.2.1.21 *Echelon Model – ZLC*

The Echelon model provides a rough estimation of the number of vehicles, distance and time required for delivering per type of vehicle considering two scenarios. The first scenario is based on direct shipments from a branch surrounding the city and using one type of vehicle. The second scenario is based on introducing a hub within the delivery area to shift the load between the vehicle supplying the hub from the branch to the hub to the vehicle delivering to the final customers considering the vehicles are different.

3.2.2 Characterisation of models

Table 2 summarizes the ML components highlighting the main outputs of the models.

Table 2: Model Library components

ID	MODEL NAME	MODEL OWNER	TYPE OF MODEL	Key outputs
1	VISUM*	BKK/SZE	Network model	Number of vehicles Distances
2	Charging station	INLE	Optimization	Best places to locate Electric Distributed Vehicle (EDV) charging stations.
3	UDR	INLE	Network model	Possible shortest delivery routing paths
4	Synthetic population	IRTX	Demand	Synthetic population (households, persons, daily activities) Infrastructure (road network, public transport network)
5	Parcel synthesizer	IRTX	Demand	List of households with individual number of parcels received on an average day

ID	MODEL NAME	MODEL OWNER	TYPE OF MODEL	Key outputs
6	MATSim	IRTX	ABM	System-level KPIs (congestion, noise, emissions)
7	Jsprit	IRTX	Network model	Individual optimized routes for a set of vehicles serving a list of deliveries with time constraints
8	Route optimization*	LMT	Network model	Step-by-step routes per vehicle operational data. Step-by-step routes per vehicle geospatial data. Number and average distance driven per vehicle type.
9	NetLogo**	Molde	ABM	Parcel tour – allocation of the deliveries per type of preference
10	Discrete Choice Modelling	Molde	Demand model	Stakeholder's preferences (from of utility function)
11	Choice Experiment Design	MOLDE	Data collection design	Stated Preference choice tasks
12	Parcel generation	TUD	Demand model	Number of parcels
13	Parcel Market	TUD	ABM	Number of parcel trips Bringer detour Crowdshipped parcels Crowdshipping success rate Crowdshipping rate per parcel
14	Parcel tour formation	TUD	ABM	Distance per vehicle type Distance per courier Depot volumes
15	Shipment model	TUD	Demand model/ABM	Number of shipments
16	Network Assignment	TUD	Network Model	Distance per vehicle type Delivery routes
17	STAR**	UPM	Global Impact Assessment.	The ranking order of alternatives based on the global sustainability evaluation
18	COPERT*	EMISIA added by UPM	Impact Assessment (externality measurement)	Energy consumption – TJ Carbon Dioxide – CO2 – t Fine Particulate Matter – PM2.5 – t Nitrogen Dioxide – NO2 – t Volatile Organic Compounds VOC – t
19	NOISE	UPM	Impact Assessment (externality measurement)	Percentage of residents exposed to noise emission.
20	Electric vehicle GHG emissions estimation	UPM	Impact Assessment (externality measurement)	Energy consumption – TJ GHG – CO2eq – t
21	Echelon	ZLC	Network Model	Number of vehicles per type

ID	MODEL NAME	MODEL OWNER	TYPE OF MODEL	Key outputs
				Total distance per type of vehicle Total time per type of vehicle

* Proprietary model

** Not included in final implementation

3.2.3 Status of models

The status of *not included* is added to flag the models that were originally considered but not implemented due to the expected outputs were either covered by other model already or the LL did not require the model anymore.

Table 3 shows the current status of the models of the ML. The status of the models is disaggregated into the 3 requirements described in section 2.1. The categories to describe the status are:

- Code status:
  - **Not included:** Model is not included in this version of the platform.
  - **Under development**: Model not ready.
  - **Tested in local computer**: Model tested in local computer.
  - **Ready to upload**: Model ready to be uploaded to the repository.
  - **In repository**: Model included in the repository.
  - **Implemented in platform**: Model already tested in the platform.
- JSON file status:
  - **Not included:** Model is not included in this version of the platform.
  - **Under development:** JSON file not ready.
  - **Ready for upload**: JSON file ready to be uploaded to the repository.
  - **In repository:** Model included in the repository.
  - **Implemented in platform**: JSON file already tested in the platform.
- Documentation status:
  - **Not included:** Model is not included in this version of the platform.
  - **Under development:** Documentation not ready.
  - **Ready for upload:** Documentation ready to be uploaded to the repository.
  - **In repository**: Documentation uploaded to the repository.
  - **Published**: Documentation published in the repository.

The status of *not included* is added to flag the models that were originally considered but not implemented due to the expected outputs were either covered by other model already or the LL did not require the model anymore.

Table 3: Status of the Model Library

ID	Model name	Model owner	Code status	JSON file status	Documentation status
1	VISUM*	BKK/SZE	Implemented in platform	Implemented in platform	In repository

ID	Model name	Model owner	Code status	JSON file status	Documentation status
2	Charging station	INLE	Implemented in platform	Implemented in platform	Ready to upload
3	UDR*	INLE	Implemented in platform	Implemented in platform	In repository
4	Synthetic population	IRTX	Implemented in platform	Implemented in platform	In repository
5	Parcel synthesizer	IRTX	Implemented in platform	Implemented in platform	In repository
6	MATSim	IRTX	Implemented in platform	Implemented in platform	In repository
7	Jsprit	IRTX	Implemented in platform	Implemented in platform	In repository
8	Route optimization*	LMT	Implemented in platform	Implemented in platform	In repository
9	NetLogo*	Molde	Not included	Not included	Not included
10	Discrete Choice Modelling	Molde	Implemented in platform	Implemented in platform	In repository
11	Choice Experiment Design	MOLDE	Implemented in platform	Implemented in platform	In repository
12	Parcel Market	TUD	Implemented in platform	Implemented in platform	In repository
13	Parcel generation	TUD	Implemented in platform	Implemented in platform	In repository
14	Parcel tour formation	TUD	Implemented in platform	Implemented in platform	In repository
15	Shipment model	TUD	Implemented in platform	Implemented in platform	In repository
16	Network Assignment	TUD	Implemented in platform	Implemented in platform	In repository
17	STAR	UPM	Not included	Not included	Not included
18	COPERT*	UPM	Implemented in platform	Implemented in platform	In repository
19	NOISE	UPM	Implemented in platform	Implemented in platform	In repository
20	Electric vehicle GHG emissions estimation	UPM	Implemented in platform	Implemented in platform	In repository
21	2echelon	ZLC	Implemented in platform	Implemented in platform	In repository

* Proprietary model

3.2.4 Status of Connectors

Table 4 shows the status of the connectors between the models of the ML. It is worth to mention that the connectors shown here are the connectors that are critical for the implementation of the DTs (black filled and dotted lines of Figure 3 and Figure 5).

The status of the connectors is disaggregated into the 3 requirements described in section 2.1. The categories to describe the status are:

- Code status:
  - **Not included:** Connector is not included in this version of the platform.
  - **Under development**: Connector not ready.
  - **Tested in local computer**: Connector tested in local computer.
  - **Ready to upload**: Connector ready to be uploaded to the repository.
  - **In repository**: Connector included in the repository.
  - **Implemented in platform**: Connector already tested in the platform.
- JSON file status:
  - **Not included:** Connector is not included in this version of the platform.
  - **Under development:** JSON file not ready.
  - **Ready for upload**: JSON file ready to be uploaded to the repository.
  - **In repository:** Model included in the repository.
  - **Implemented in platform**: JSON file already tested in the platform.

Table 4: Status of connectors

ID	Connector name	Model from	Model to	Connector owner	Code status	Json file status
1	2Echelon_2_Copert	2Echelon (ZLC)	COPERT (UPM)	ZLC	Implemented in platform	Implemented in platform
2	lmt_LEAD_input_to_ COPERT	Route optimization (LMT)	COPERT (UPM)	LMT	Implemented in platform	Implemented in platform
3	lmt_LEAD_input_to_ NOISE	Route optimization (LMT)	Noise (UPM)	LMT	Ready for upload	Ready for upload
4	2Echelon_2_Electric Emissions	Echelon (ZLC)	Electric Emissions (UPM)	ZLC/UPM	Implemented in platform	Implemented in platform
5	LML_TO_EV	Route optimization (LMT)	Electric Emissions (UPM)	UPM	Implemented in platform	Implemented in platform

ID	Connector name	Model from	Model to	Connector owner	Code status	Json file status
6	Conn_Gen2Market	Parcel Generator (TUDelft)	Parcel Market (TUDelft)	TUDELFT	Implemented in platform	Implemented in platform
7	Conn_Market2Sched	Parcel Market (TUDelft)	Parcel Tour Formation (TUDelft)	TUDELFT	Implemented in platform	Implemented in platform
8	Conn_Sched2Ntw	Parcel Tour Formation (TUDelft)	Network assignment (TUDelft)	TUDELFT	Implemented in platform	Implemented in platform
9	Conn_Ship2Ntw	Shipment Model (TUDelft)	Network assignment (TUDelft)	TUDELFT	Not included	Not included
10	Conn_Ntw2Noise	Network assignment (TUDelft)	Noise (UPM)	TUDELFT	Not included	Not included
11	Conn_Sched2COP	Parcel Tour Formation (TUDelft)	COPERT (UPM)	TUDELFT	Implemented in platform	Implemented in platform
12	Conn_popsyn2parcelsyn	Population Synthesyzer (IRTX)	Parcel Synthesyzer (IRTX)	IRTX	Implemented in platform	Implemented in platform
13	Conn_parcelsyn2jsprit	Parcel Synthesyzer (IRTX)	Jsprit (IRTX)	IRTX	Implemented in platform	Implemented in platform
14	Conn_jsprit2copert	Jsprit (IRTX)	COPERT (UPM)	IRTX	Implemented in platform	Implemented in platform
15	Conn_jsprit2matsim	Jsprit (IRTX)	MATSim (IRTX)	IRTX	Implemented in platform	Implemented in platform
16	Conn_popsyn2matsim	Population Synthesyzer (IRTX)	MATSim (IRTX)	IRTX	Implemented in platform	Implemented in platform
17	Visum_2_Copert	Visum (BKK)	COPERT (UPM)	BKK/SZE	Implemented in platform	Implemented in platform

ID	Connector name	Model from	Model to	Connector owner	Code status	Json file status
18	Conn_Sched2NL	Parcel Tour Formation (TUDelft)	NetLogo (MOLDE)	MOLDE	Not included	Not included
19	Conn_DCM2NL	Discrete Choice Modelling (MOLDE)	NetLogo (MOLDE)	MOLDE	Not included	Not included
20	Visum_2_EVCO2	Visum (BKK)	Electric Emissions (UPM)	BKK/SZE	Implemented in platform	Implemented in platform
21	UDR-2-EVCO2	UDR (INLE)	Electric Emissions (UPM)	INLE	Implemented in platform	Implemented in platform
22	UDR-2-COPERT	UDR (INLE)	COPERT (UPM)	INLE	Implemented in platform	Implemented in platform
23	irtx-matsim-copert-connector	MATSim (IRTX)	COPERT (UPM)	IRTX	Implemented in platform	Implemented in platform

4 Summary and next steps

This deliverable presents the current version of the LEAD ML and gives an overview of the logic of the ML, the models included and the approach for its application to cases. The ML was developed following a model-centric approach where models are included in the ML and explicit relationships are developed between them via model connectors. Model developers will need to give information to create the connectors and the model into the ML. The deliverable also describes the documentation needed to support model implementation by LEAD and future users of the ML library.

In addition, this deliverable defines the LEAD meta-model (M2) which allows comparison and election of individual models, provides guidance for the use of models, and supports the development of new models. It includes the approach to the M2 use and provides some M2 instances.

This updated version of the ML shows how it evolved to adapt to the needs of the LLs. The development of the ML depends on the twinning demands of the LLs. This implies that the ML adapts and supports the needs of the instances of DTs and not pushes developments to the LLs, making the LEAD platform use-case oriented instead of technology oriented. Consequently, some connections that looked important to develop at the first version of the ML and M2 were not pursued and that new connections not considered originally were effectively implemented. This updated version of the ML has been published via the project's GitHub repository.

5 Bibliography

Hörl, S., Balac, M., 2021. Synthetic population and travel demand for Paris and Île-de-France based on open and publicly available data. Transportation Research Part C: Emerging Technologies 130, 103291. https://doi.org/10.1016/j.trc.2021.103291

Horni, A., Nagel, K., Axhausen, K.W. (Eds.), 2016. The Multi-Agent Transport Simulation MATSim. Ubiquity Press. https://doi.org/10.5334/baw

Tampakis, P., Pelekis, N., Doulkeridis, C., & Theodoridis, Y. (2019). Scalable Distributed Subtrajectory Clustering. Proceedings – 2019 IEEE International Conference on Big Data, Big Data 2019, 1, 950–959. https://doi.org/10.1109/BigData47090.2019.9005563

Tapia, R., & Kourounioti, I. (2022). LEAD D1.2 Knowledge Base Reference Models, LEAD Project.

Tavasszy, L., & de Jong, G. (2014). Modelling Freight Transport. In Lorant Tavasszy & G. de Jong (Eds.), Modelling Freight Transport. Elsevier Inc.

Fernández Balaguer, S (2023), :LEAD D3.4, Madrid Value case, Final demonstrator, LEAD project

Groenewoud, R & Tapia, R (2023) D3.4, The Hague Value case, Final demonstrator, LEAD project

Mougeot, B, Roussellet, E, Horl, S & Briand, Y (2023) D3.4, Lyon Value case, Final demonstrator, LEAD project

Büki, A, Teslér, P & Horváth, A Y (2023) D3.4, Budapest Value case, Final demonstrator, LEAD project

Nascimento, C, Bråthen, S, Hansson, L, Gatta, V & Marcucci, E Y (2023) D3.4, Oslo Value case, Final demonstrator, LEAD project

Reis, P, Rizopoulos, D & Politaki, D Y (2023) D3.4, Portp Value case, Final demonstrator, LEAD project

6 Annex I: JSON readme

The LEAD platform aims to integrate the models created within the project's context. The procedure should allow a LEAD platform user to be able to upload a model by specifying several parameters of the input, output and execution of the model. The following specification consists of an effort to create a well-defined and machine-friendly way to enable such features on the platform.

Top Level Schema

```
{
    "application": {},
    "input": [],
    "output": []
}
```

Application

The application field should contain information regarding the identification and execution parameters of the model. An example is presented below to provide a use case and offer a template, along with a description for each field.

```
{
    "application": {
        "key": "parcel-gen",
        "version": "0.1.0",
        "name": "Parcel Generation",
        "description": "The Parcel Generation model generates [...]",
        "types": ["ABM"],
        "application-information": {
            "url": "git://path-to-code",
            "type": "script-python",
            "environment": "Ubuntu 20.04.3 LTS",
            "build-instruction": "pip install -r requirements.txt",
            "execution-instruction": "python3 src/Parcel_Generation.py {sim-name} {inp
ut-folder} {output-foler} {parameters-file} {time-skim-matrix} {distance-skim-matrix}
{area-shape-file} {socioeconomic-data} {parcel-nodes} {courier-market-shares}",
            "execution-example": "python3 src/Parcel_Generation.py Lable InputFolder O
utputFolder Params_ParcelGen.txt skimTijd_new_REF.mtx skimAfstand_new_REF.mtx Zones_v4
.shp SEGS2020.csv parcelNodes_v2.shp CEPshares.csv",
        }
    }
}
```

The combination of `id-version` is used to identify a model and must be unique while the `name` and `description` are the fields that are going to be shown to the users and can be the same e.g. across different versions of a model.

A model type is a tag used to characterize the model into different categories. The list can be expanded by the user. Every model type consists of a `key`, a `name` and a `description`.

```
[
    {
        "key": "ABM",
```

```
        "name": "Agent Based Model",
        "description": "Agent Based Model"
    }
]
```

The `application-information` contains the information needed so that a model can be packaged, containerized and deployed to the platform. More specifically:

- `url`: the URL of the repository where the code of the model can be found.
- `type`: can be one of the following (*more to be added as more models become available*)

```
[ "script-python", "script-r", "executable-binary", "executable-jar", "api" ]
```

- `environment`: could be either the operating system where the model is running as given by `lsb_release -a`, the tag of the docker image used, or the URL of a Dockerfile from the repository
- `build-instruction`: a field containing the command[s] needed to prepare the environment or the executable. Empty string if none is needed.
- `execution-instruction`: a field containing the command which initiates the execution of the model. Curly braces with the input `key` are used to define which input is expected to the specified argument location.
- `execution-example`: an execution example based on provided test data

Input

Input refers to any arguments given as input to the model.

Top level structure:

```
{
    "key": "",
    "name": "",
    "description": "",
    "type": "",
    "schema": {},
    "validators": []
}
```

- the `key` is used to identify the input argument and must be unique
- the `name` and `description` arguments are the fields exposed to the platform users
- the `type` defines the type of the input. Can be any of the following

```
    {
        "types": [
            "integer", "float", "string", "boolean",
            "integerArray", "floatArray", "stringArray", "booleanArray",
            "object", "doubleArray",
            "file", "url"
        ]
    }
```

- the `schema` field is used to define the inner structure of the composite types defined (e.g. `object`, `doubleArray`). In those cases, a schema is expected where each element is matched to one of the basic types. As an example, a schema is presented below:

```
{
    "schema": [
        {
            "name": "vehicle_speed",
            "desc": "The average speed of the vehicle",
            "type": "float"
        },
        {
            "name": "vehicle_capacity_boxes",
            "desc": "The capacity of the vehicle in number of boxes",
            "type": "integer"
        }
    ]
}
```

- the `validators` array consists of an array of objects defining further checks that need to be applied on the model's input. The available validators' types are predefined and implemented in the platform and the user can select among those. As a result, this array can also be expanded based on the input from the models. An example of the available supported validators is presented below together with an example of a value definition.

```
{
    "validators": {
        "columns": "{\"1\": {\"name\": \"CEP\", \"type\": \"str\"}}",
        "extension": ".shp",
        "regex": "\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\\.[A-Z]{2,}\b",
        "str_length_max": "50",
        "str_length_min": "10",
        "value_eq": "50",
        "value_ne": "50",
        "value_lt": "50",
        "value_le": "50",
        "value_gt": "50",
        "value_ge": "50",
        "object_props": [
            {"name": "N1", "type": "string"},
            {"name": "N2", "type": "integer"}
        ]
    }
}
```

Output

The output follows exactly the same principles as the input

7 Annex II: Documentation template table of contents

This Annex shows the contents of the documentation template sent to the model owners.

Contents

Figure 6: Table of contents of document template

The contents requested were:

1. **Introduction:** What is the scope of the model? What are their objectives? What type of model and which assumptions does it have?
2. **Requirements:**
    1. Software requirements: What is it needed to run the model? Please include versions of the language, software, libraries, etc.
    2. Inputs/Outputs: Description of each input

    3. Paths structure: How should the directory look like? Which folders should it have and where does the Inputs/outputs are read/written?
3. **Model Description:** This section describes the different files and scripts present in the model
4. **Instructions to run the model:** This is an extraction of the content of the JSON file in AnnexI
    1. Command line execution: Commands for preparing the libraries and check versions, commands to actually run the model and Identification of the arguments in the command line
    2. Requirements: Details of the software versions required and the file structure:
        i. which files are in each folder
        ii. what do they do
        iii. what type of file (csv, etc)
        iv. what requirements do they have (table with headers, what each row represent, …)

8 Annex III: Living Labs DT scenarios

Figures 6 to 11 show the instances of the M2 representing the LLs DTs. The generation of this instances are done by the LLs in WP3 and are out of the scope of this deliverable.

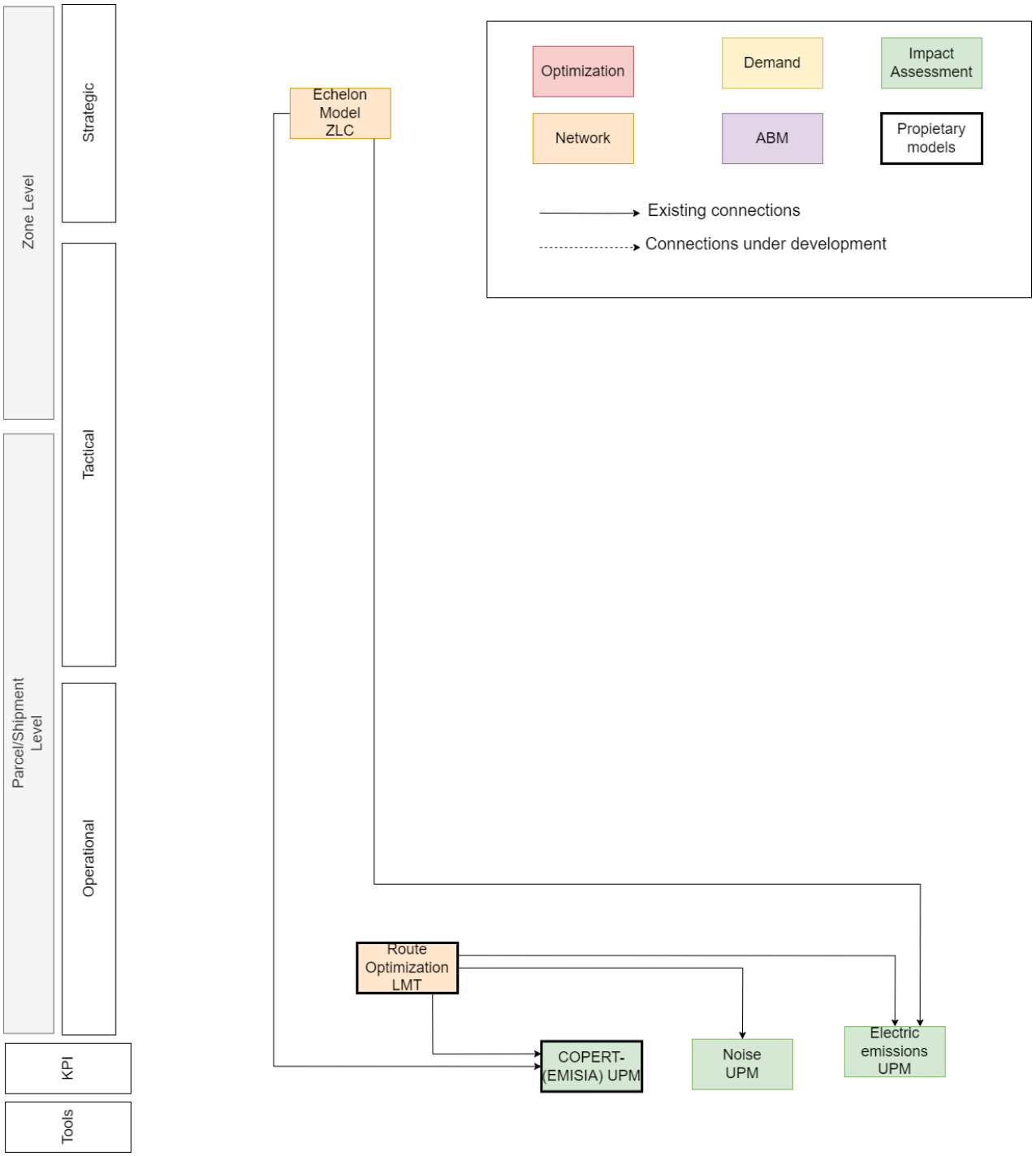


Figure 6: Madrid LL

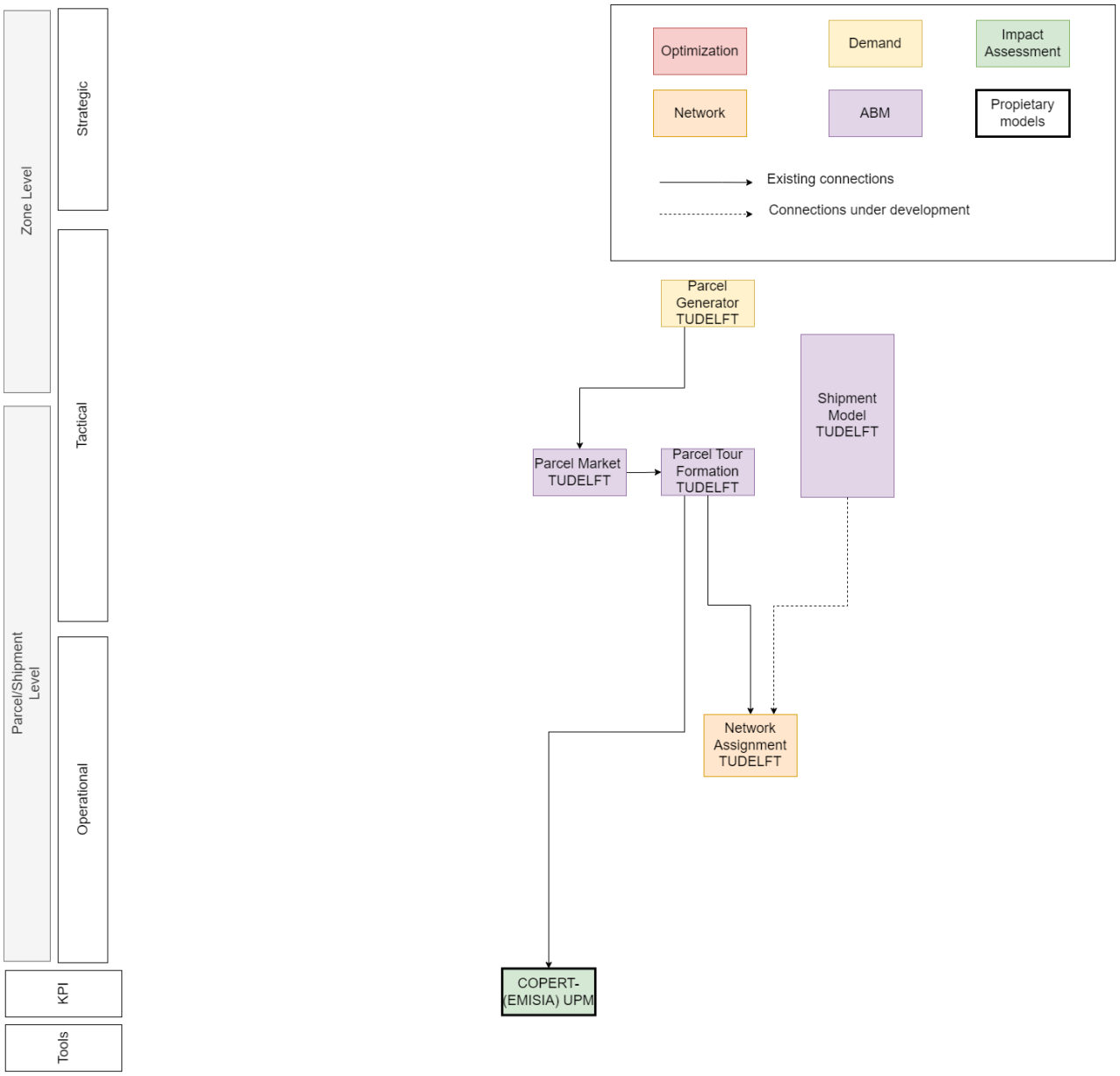


Figure 7: The Hague LL

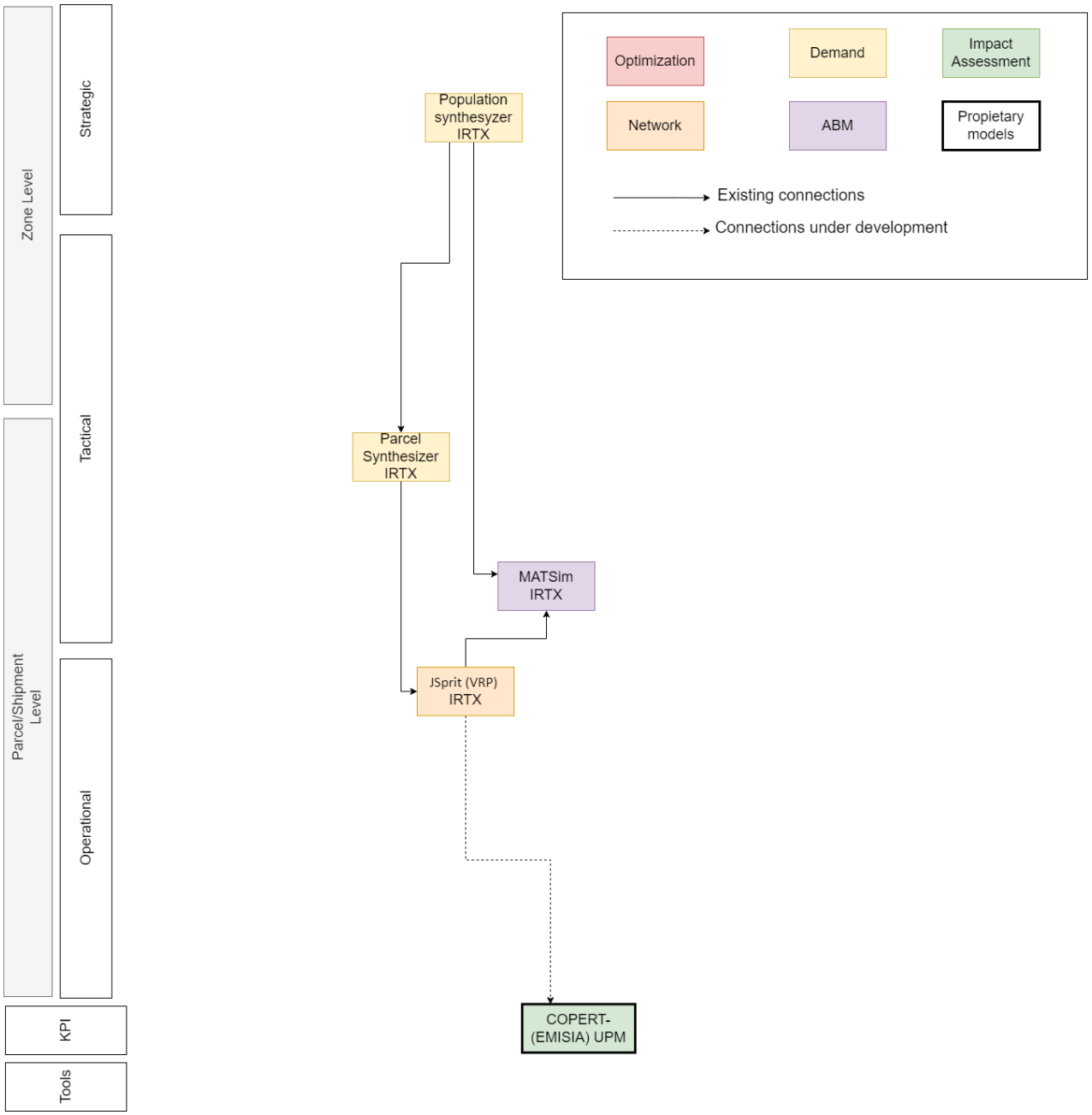


Figure 8: Lyon LL

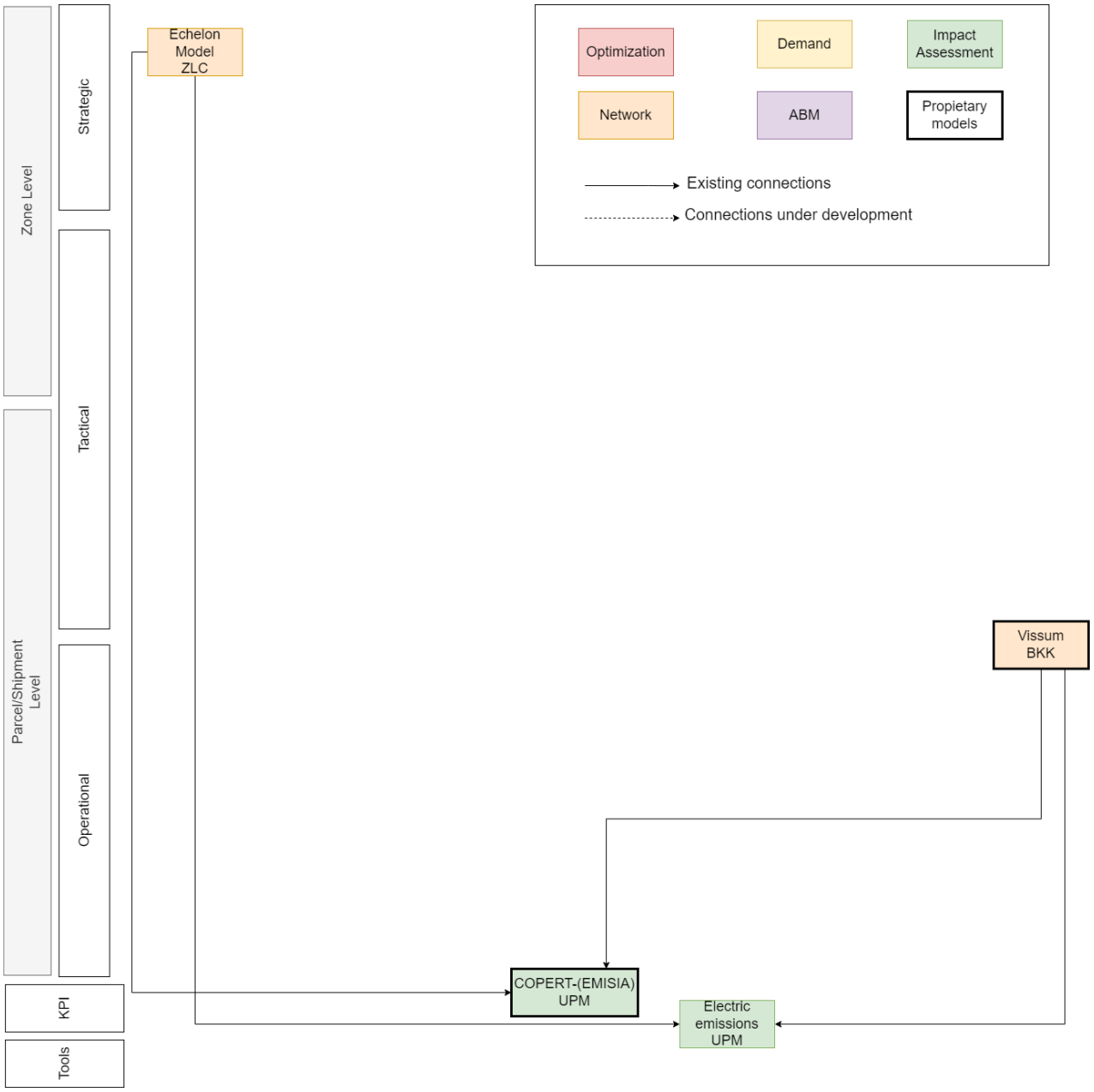


Figure 9: Budapest LL

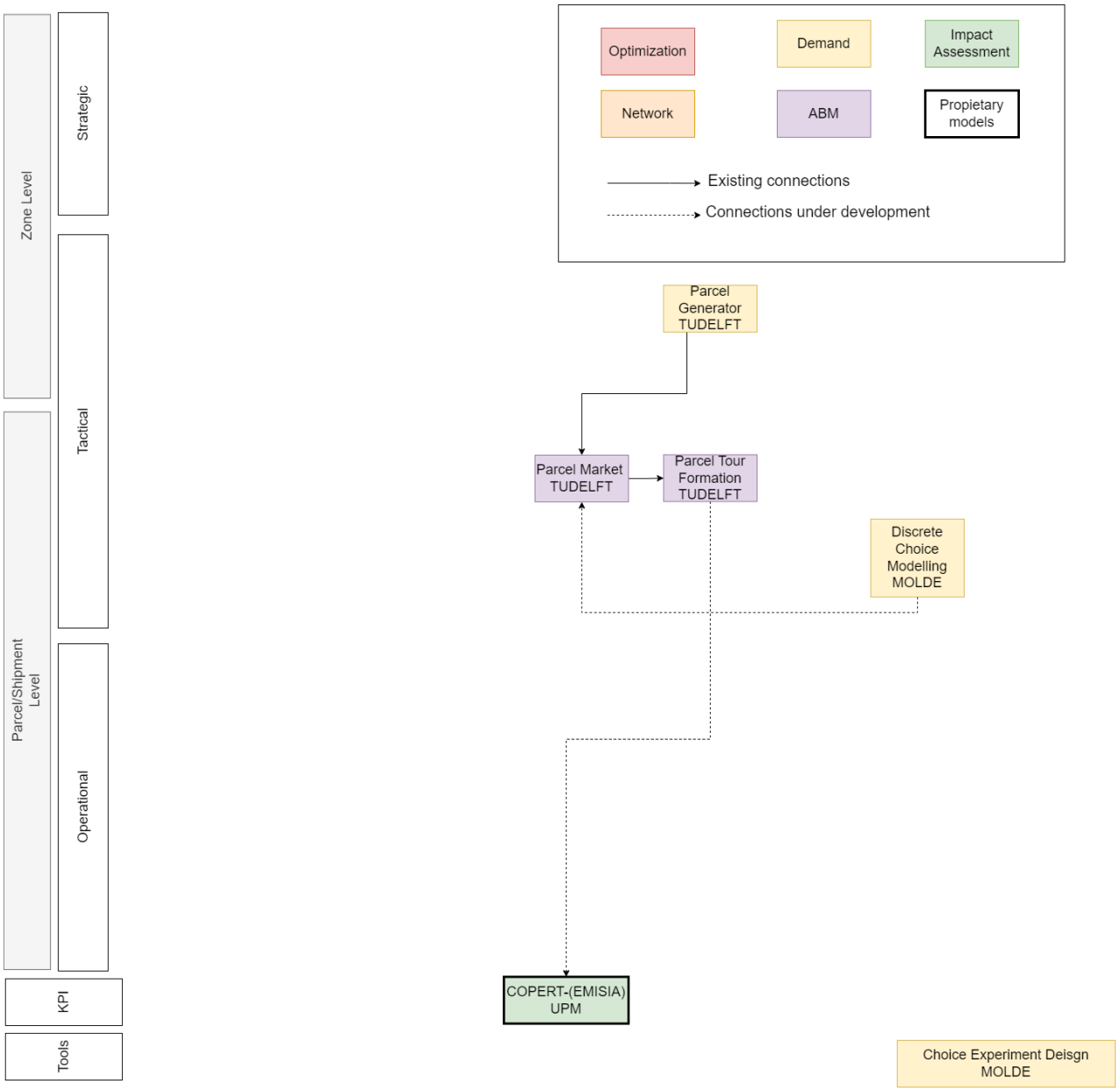


Figure 10: Oslo LL

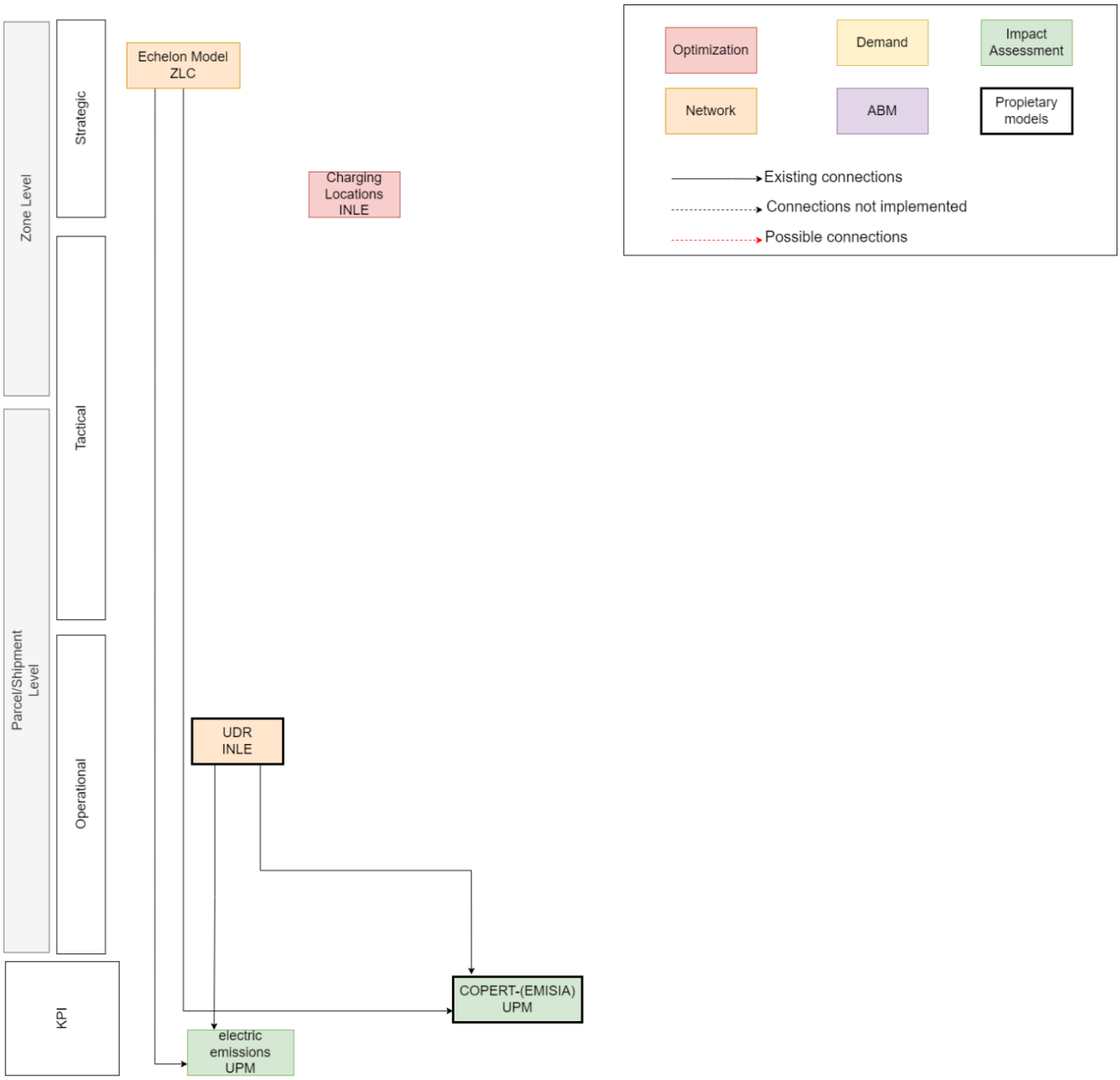


Figure 11: Porto LL

9 Annex IV: Github Screenshots

Figures 12 to 14 show some screenshots of the current state of the Github repository.

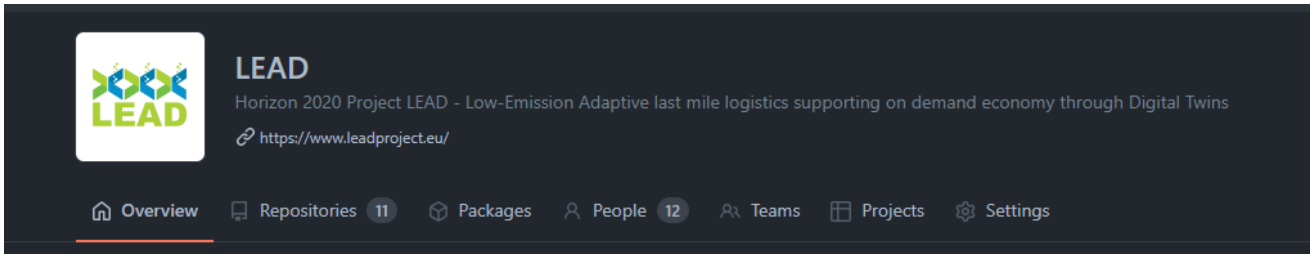


Figure 12: Porto LL

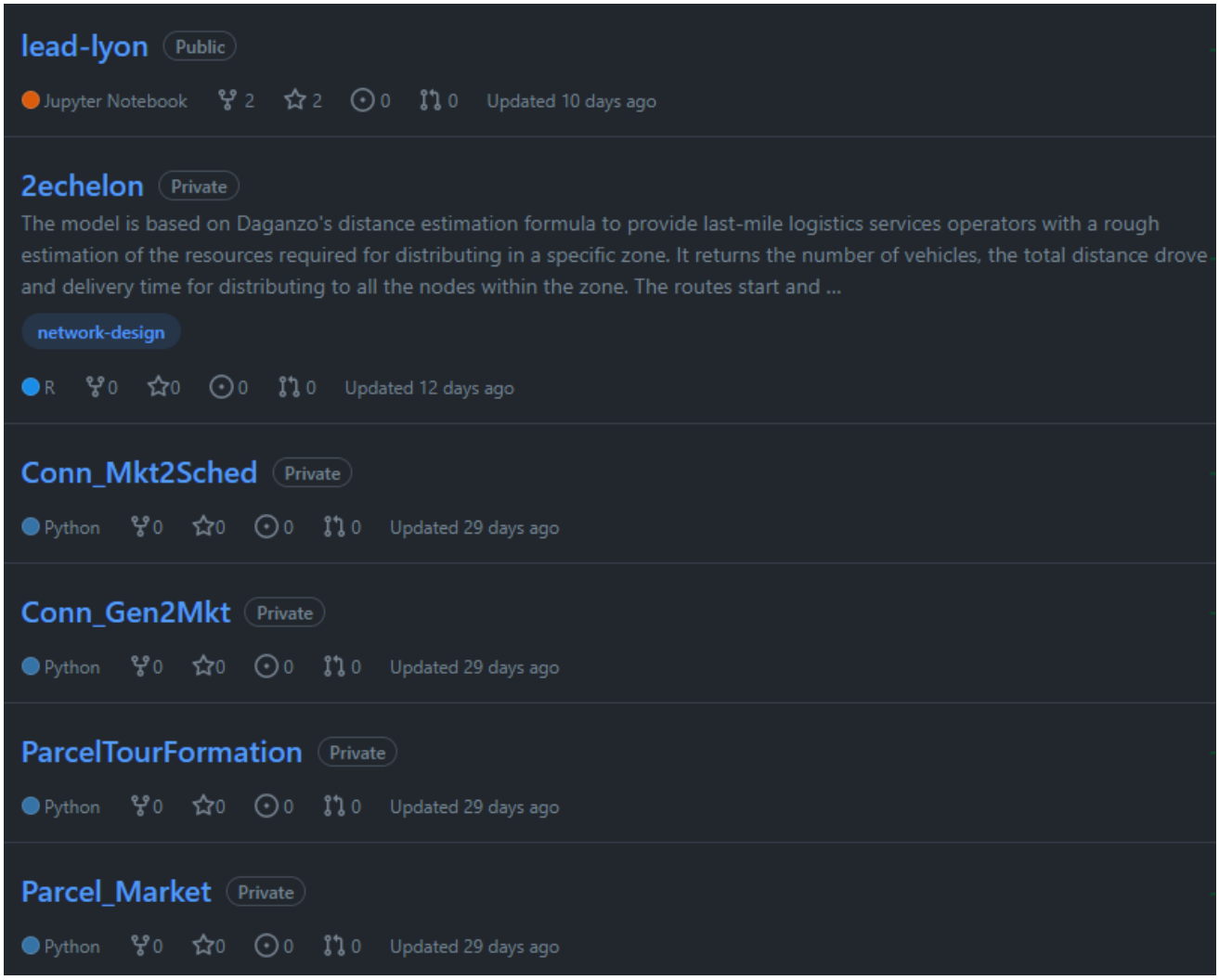


Figure 13: Models present in repository (part I)

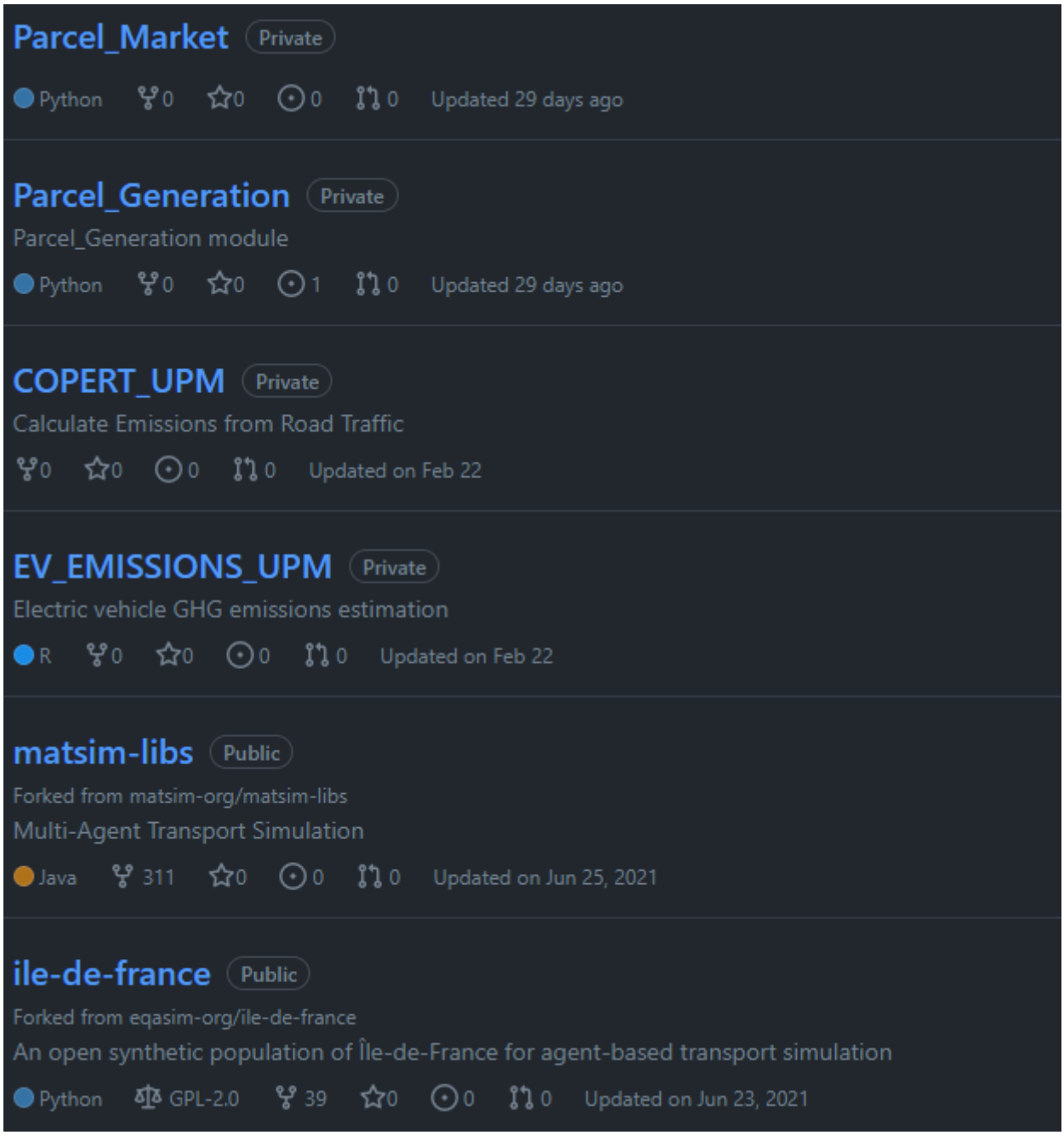


Figure 14: Models present in repository (part II)

10 Annex V: Complete model documentations

This Annex shows the model documentation that are uploaded in the repository.

VISUM-MTM model

Documentation

Author(s): Richárd Horváth dr.

Author’(s’) affiliation (Partner short name): BKK, SZE

Revision History

Version No.	Date	Details
1.0	14/07/2022	Initial version
1.1	26/10/2022	Second version (after first test running)

Contents

List of tables

1 Introduction

1.1 Scope and objectives

This document serves as technical documentation for the VISUM-Macroscopic Transport Model. It explains its scope, the requirements to run the program, the inputs and outputs, and how to access it.

The PTV VISUM is a transport modelling software. BKK, as partner of the LEAD project possess a full scope transport model for the city of Budapest, which is a strategic Macroscopic Transport Model (MTM). The VISUM software is used as a tool to run the MTM. MTM is actually a dataset which contains data in the form of matrices. With the model the passenger and freight traffic can be investigated. The model is available for the public and software independent, but all of the registered usage done by PTV VISUM.

The model can be used to investigate the effects of major modifications in the transport network (new tram line, new bridge, closed road, etc.) or in the territorial development structure (new shopping centre, new university, new residential area, etc.).

The modelling process based on the 4-step modelling methodology:

1. Trip generation: based on statistic information (population, traffic attractive facilities – schools, stores, workplaces etc.) the origin and destination traffic volume are generated for each zones.
2. Trip distribution: the distribution of the trips between the zones based on the distance and other parameters
3. Mode choice: Based on several parameter, like car ownership, access time and cost by different modes, etc. the choice of the mode is happening in this step
4. Assignment: search for the best route for each zone relation and each mode

The 1-3. steps are the part of the demand model, which was integrated to the VISUM in 2019. Before that these steps run out of the VISUM (used Excel sheets, where the calculations are done by predefined Macros.)

The steps above are true for the public and private transportation layers, but the freight transport layer has not been a part of the demand model, yet. For this layer there are fix matrices for 4 different freight transport group (based on the weighs of the vehicles: J1T: < 3,5 t; J2T: 3,5 – 7,5 t; J3T: 7,5 – 12 t; J4T: > 12 t), so with the freight layers happen only the assignment step.

In the first step, the network or the structure is modified in the model. Than user have to decide, that is there any further modification needed in the parameters (for example in the utility function), if yes, the user do it and then choose the appropriate elements of the procedure sequences. After that user start to run the model. The running time based on the number of the scenarios and the capacity of the computer (usually ~2 hours/scenario). The results of the model are the loaded network and many statistic parameters.

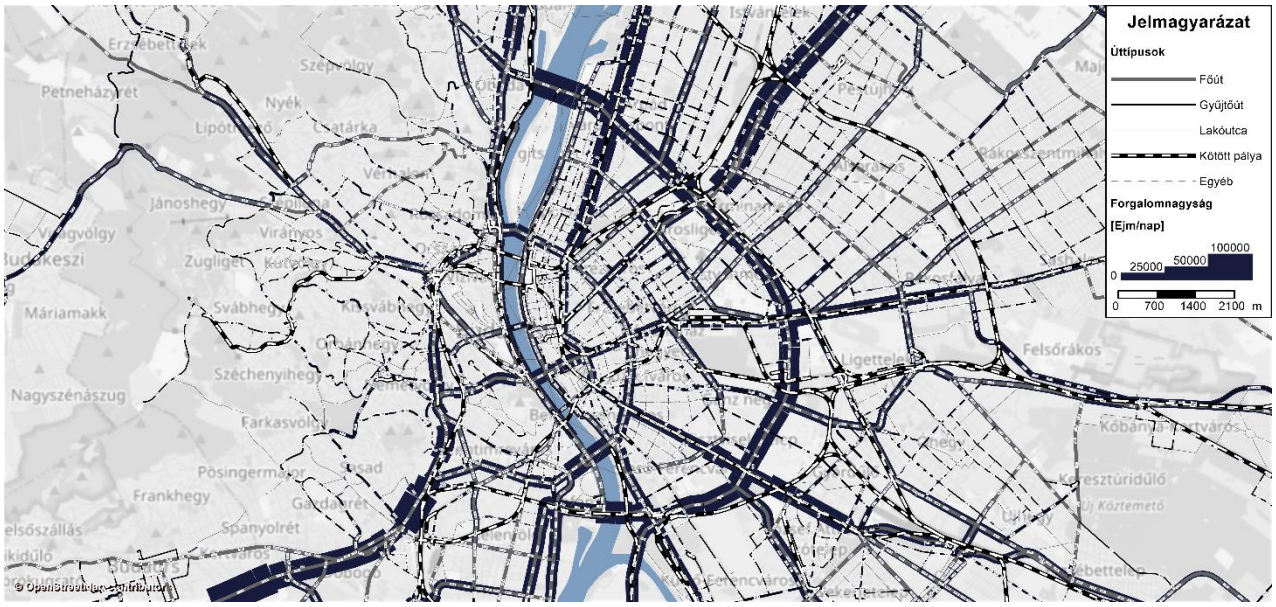


Figure 1 The loaded network

In LEAD project, the model is used to examine the effects of different delivery methods in the service area of logistics service provider (WSZL) in connection with the passenger transport of the area. In order to connect the model to the LEAD Platform, a Python based server program was created, because VISUM only runs on Windows platform. This program works as a RESTful server on Windows platform and receives queries and communicates with the other programs, which work on Linux platform. This program transfers the queries to the Visum system. After the Visum assignment steps, the short program sends the results back to the Linux based system.

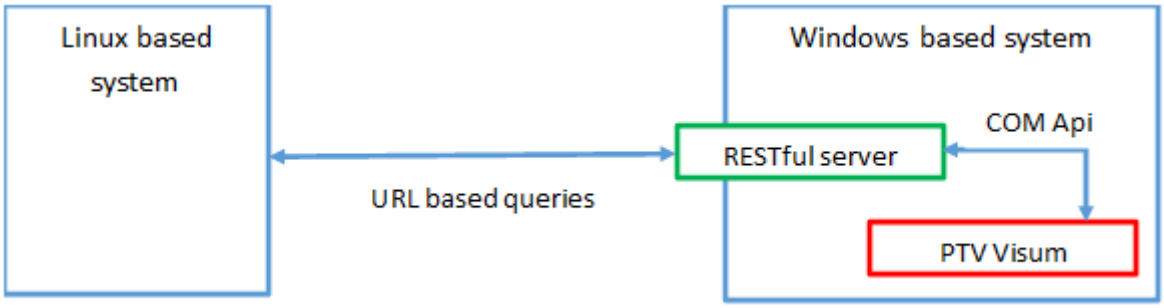


Figure 2 Server program

2 Requirements.

2.1 Software requirements

This model requirement the VISUM simulation software and an interface software which communicates the local VISUM with the external user. The interface software uses the Python language and the Flask, Pywin32, PyProj libraries.

2.2 Input/Outputs

2.2.1 Inputs

The model needs four main parameters as input:

- name of VISUM .ver file. The MTM model of BKK is very large, and it is not necessary to be uploaded, because the infrastructure does not change.
- list of addresses of customers
- list of delivery routes
- used vehicles and their types

Table 1 Customer.csv – Input

Inputs	Description
1. parameter	Customer number
2. parameter	Address – city
3. parameter	Address – street and house number
4. parameter	Address – postcode
5. parameter	Coordinate – latitude in WGS84 format
6. parameter	Coordinate – longitude in WGS84 format

Table 2 Journey.csv – Input

Inputs	Description
1. parameter	Journey number
2. parameter	Vehicle code
3. parameter	Departure/arrival time
4. parameter	Waiting / service time
5. parameter	Customer number

Table 3 Vehicle.csv – Input

Inputs	Description
1. parameter	Vehicle no (for VISUM)

2. parameter	Vehicle code
3. parameter	Vehicle description
4. parameter	Vehicle type code

Table 4 Vehicle_type.csv – Input

Inputs	Description
1. parameter	Vehicle type code
2. parameter	Vehicle type description

2.2.2 Outputs

VISUM generates a file in JSON format, which contains the delivery distances and the type of the vehicle. Table 3 shows the outputs generated in the Excel file.

Table 5 Result.json – Outputs

Outputs	Description
1. parameter	Vehicle code
2. parameter	Distance (km)

2.3 Paths structure

Root

3 Model Description

This section describes the different files and scripts present in the model

File name	Location	Description
customer.csv	root	File containing the data of shops
journey.csv	root	File containing the data of

		routes
vehicle.csv	root	File containing the data of used vehicle
vehicle_type.csv	root	File containing the data of used vehicle types

4 Instructions to run the model

4.1 Command line execution of the model on the server side

The interface program which receives the data and forwards it to the Visum can run on the server.

4.2 Command line execution of the model from the client side

```
curl --silent --location --request POST "localhost:5000/tupload" ^
--form "file_customer=@./customer.csv" ^
--form "file_journey=@./journey.csv" ^
--form "file_vehicle=@./vehicle.csv" ^
--form "file_vehicle_type=@./vehicle_type.csv" ^
--form "file_ver=@./test_model.ver" ^
--output "result.json"'
```

The BKK-MTM is very large, more than 1.9 Gbytes, so another calling method was created. In this method the .ver file is not necessary, but the name of .ver file is needed.

```
curl --silent --location --request POST "localhost:5000/tupload" ^
--form "file_customer=@./customer.csv" ^
--form "file_journey=@./journey.csv" ^
--form "file_vehicle=@./vehicle.csv" ^
--form "file_vehicle_type=@./vehicle_type.csv" ^
--form-string "file_ver=@./test_model.ver" ^
--output "result.json"'
```

Table 6 VISUM model execution values

Attributes	Possible Values	Description

POST	1.2.3.4:8000	URL and port of BKK
file_customer	<input_directory_path>	Add the input customer file and path
file_journey	<input_directory_path>	Add the input journey file and path
file_vehicle	<input_directory_path>	Add the input vehicle file and path
file_vehicle_type	<input_directory_path>	Add the input vehicle type file and path
file_ver	<name>.ver	Name of the VISUM .ver file or the name of .ver file on the server of BKK
output	<name>.json	Name of the result

4.3 Requirements

4.3.1 Folder - Root

The folder contains the needed data file (customer, journey, vehicle and vehicle_type) for the model.

Charging Station Facility Location Problem (CS-FLP)

Documentation

Author(s): Dimitrios Rizopoulos, Andreas Alexopoulos, Ioanna Fergadiotou

Author’(s’) affiliation (Partner short name): INLE

Revision History

Version No.	Date	Details
1.0	17.05.2023	The first version of the document.

Contents

List of tables

1 Introduction

1.1 Scope and Objectives

This document aims to provide the required information about how to run the CS-FLP model developed by Inlecom in the context of LL6 and delivered within D3.14, the Porto Value Case final report. CS-FLP is solved based on a weighted-p-median model, based on a Mixed Integer-Linear Problem (MILP) formulation solved with the OR-Tools library; the solution can provide an answer about the best locations (i.e., where) to place Charging Stations on an LSP's network. The CS-FLP model is a new digital model, adjustable to the needs of different LSPs in Europe, and has been based on previous work by Inlecom in the Track&Know project. The public repository of the previously elaborated algorithm can be found here: https://github.com/ibadkureshi/tnk-locationallocation.

A formal description of the CS-FLP is: Given a set of locations *N* and given that the LSP wants to install CS at *Y* out of the N locations, a solution to the CS-FLP provides which of the *N* locations the CS should be placed to minimize operational cost according to distances and demand. The public GitHub, including the MILP model of a weighted-p-median model, and the Python implementation of the model, can be found here: https://github.com/dimrizo/CS-FLP.

2 Requirements.

2.1 Software requirements

The implementation of the CS-FLP model is in Python (Version 3.7) language and is dependent on a series of open Python libraries, which can be found in the requirements.txt file. Another CS-FLP dependency is the OpenTripPlanner routing engine, which is used for the calculation of real-world network distances based on OpenStreetMap data.

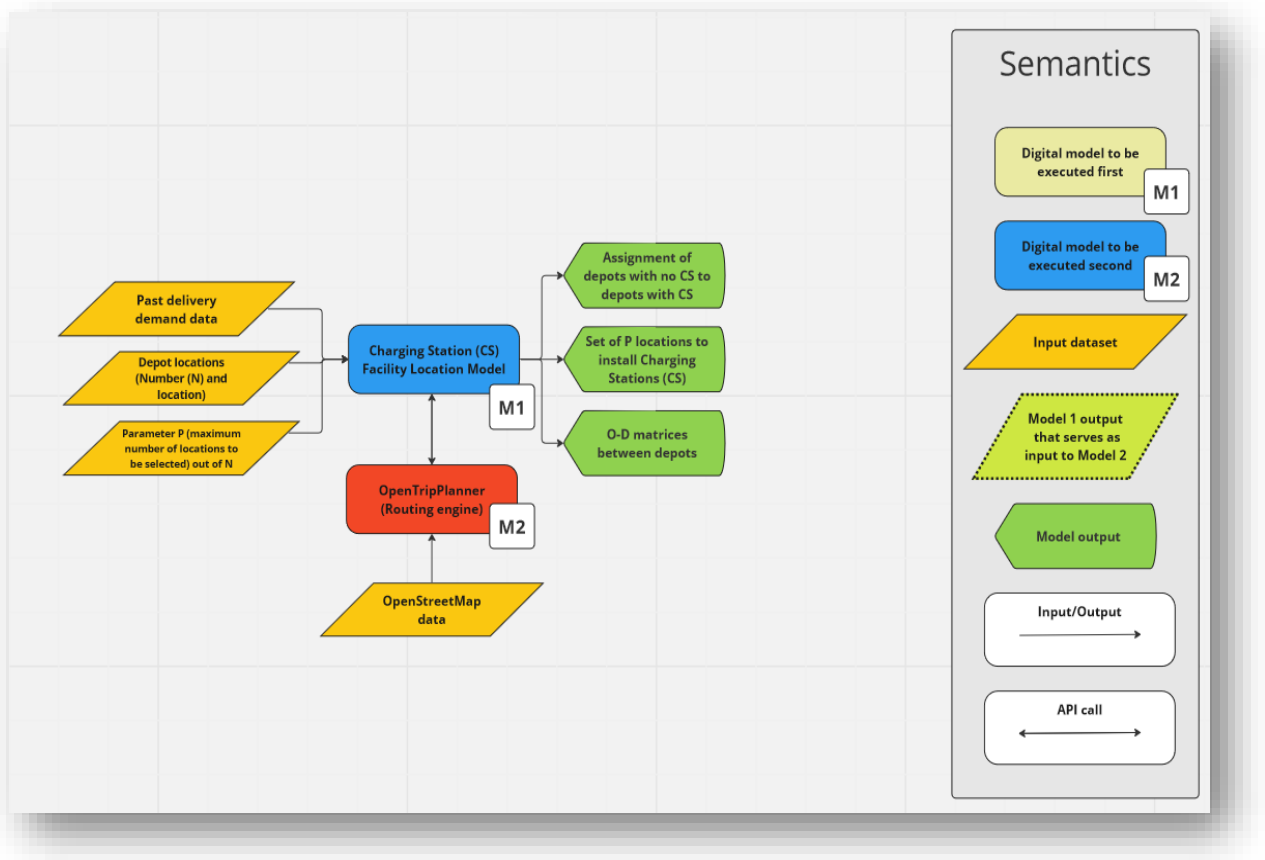


Figure 1: CS FLP's main components

2.2 Input/Outputs

2.2.1 Inputs

The CS-LFP model has the following inputs and its outputs.

Table 1 CS-FLP – Inputs

Inputs	Description
Parcel deliveries data	Past or real time data
Depot data	Data about depot locations and names (has to include a key ID that is connecting to the parcel deliveries file)

Parameter P	Which is the maximum number of facilities/depots to be selected out of the wider network of N facilities.
O-D Matrices for road network distances between depot and clients	These data are the real road network distances that in this model are dynamically generated by the OpenTripPlanner routing engine.
OpenStreetMap data	.osm.pbf file in the case that you need to deploy the OpenTripPlanner API for the dynamic generation of O-D matrices.

2.2.2 Outputs

Table 2. CS-FLP – Outputs

CS-FLP can be run based on one entry point weighted_p_median.py. After a successful execution the user should get the following results.

Outputs	Description
Facilities to be selected for the installation of CS	Based on the value of P and the solution of the MILP model, the model returns in the console the names of facilities that are selected for the CS network.
Assignment of depots (with no CS) to depots with CS, as indicated by the first output	An assignment of the non-selected depots to selected depots is returned based on the objective function and criteria considered.
Distances between depots	The origin-destination distances are provided for the N locations considered.

3 Model Description

This section describes the different files and scripts present in the model.

File name	Location	Description
weighted_p_median.py	Root folder	The entry point for the solution of the FLP problem.
read_file.py	Root folder	Reads file from hard drive. Also serializes it to hard drive, for faster loading times.
Utilities/data_pre_preprocessing (folder)	Root folder	Includes several python modules for data manipulation.
routing (folder)	Root folder	Includes all Python modules responsible for real-world routing results based on OpenTripPlanner and haversine distance if OpenTripPlanner fails.
opentripplanner (folder)	Root folder	Includes all data and software related to the re-use of OpenTripPlanner
input (folder)	Root folder	This is the folder where all input files should be placed.

The rest of the files included in the GitHub repo are complementary to the main ones presented above and are not necessary for execution. Below, the mathematical model implemented in weighted_p_median.py is included.

Nomenclature

Parameters	
i, j	*Indices are used to indicate facilities.*
N	*The number of available facilities in the last-mile network.*
P	*The maximum number of depots at which charging stations can be installed.*

D_i	*Demand at depot i.*
W_i	*Demand weight at depot i, based on a normalization of demand at depot i comparing to the total demand at all depots.*
$C_{i,j}$	*The transport cost from i to j.*
Variables	
$X_{i,j}$	*A binary variable that is equal to 1, if facility is assigned to facility j, when j has CS available.*
y_j	*A binary variable that is equal to 1, if depot j is selected for the CS installation.*

Constraints

Constraints	
$\sum_{j=1}^{N} X_{i,j} = 1, \forall\, i \in N$	*Every depot i has to be served by exactly one depot j.*
$\sum_{j=1}^{N} y_j \leq P$	*The sum of all variables y_j has to be less or equal to P, meaning that we may have charging stations at maximum P out of N depots.*
$X_{i,j} \leq y_j, \forall\, i,j \in N$	*In order for a facility i to be served by a facility j ($X_{i,j} = 1$), then we would need to have charging stations established there ($y_j = 1$).*

Objective function 1 for distance-based assessment

Objective function 1 for the CS-FLP	
$$MINIMIZE \sum_{i=1}^{N}\sum_{j=1}^{N} X_{i,j} * C_{i,j}$$	*Distance-based objective function minimizing distances-to-be traversed by EDV trucks performing delivery tours in depots with no charging stations, to depots with charging stations.*

Objective function 2 for demand-weighted distance-based assessment

Objective function 2 for the CS-FLP	
$$MINIMIZE \sum_{i=1}^{N}\sum_{j=1}^{N} (X_{i,j} * C_{i,j} * W_i)$$	*Demand-weighted distance-based objective function minimizing distances-to-be traversed by EDV trucks performing delivery tours in depots with no charging stations, to depots with charging stations.*
$$W_i = \frac{D_i}{\sum_{j=1}^{N} D_j}, \forall\, i \in N$$	*Formula for the calculation of the demand relative weight at depot i, based on the total demand across all depots.*

4 Instructions to run the model

4.1 Command line execution of the model

4.1.1 Instructions and commands

To run the model you need to setup two environments:

- Python 3.7 environment (e.g., based on Anaconda) and Python packages based on requirements.txt file,
- Java 8 Runtime Environment in order to run OpenTripPlanner routing engine through the .bat file located in root_folder/opentripplanner

Given the correct placement of input files into the input folder, then the model can be run by weighted_p_median.py.

4.1.2 Arguments

CS-FLP takes no argument through the cmd, the right input files need to be places in the root_folder/input folder.

If the OpenTripPlanner Grizzly server instance is running (Java runtime environment is setup), if all of the input files are placed at the right folders, and if the Python 3.7 environment is setup, then you can just run UDR model by the command line.

UDR: Unsuccessful and Small Deliveries Rescheduling/ scheduling model

Documentation

Author(s): Dimitrios Rizopoulos, Andreas Alexopoulos, Ioanna Fergadiotou

Author'(s') affiliation (Partner short name): INLE

Revision History

Version No.	Date	Details
1.0	24.02.2023	First version of the document.

Contents

List of tables

1 Introduction

1.1 Scope and objectives

The aim of this document is to provide the required information about how to run the UDR model that has been developed by Inlecom in the context of LL6 and is delivered within D3.14, the Porto Value Case final report. UDR is a logistics model that focuses on the last-mile delivery of goods. The model is based on the mathematical program called Capacitated Electric Vehicle Routing Problem with Time-Windows by utilizing Electric Delivery Vehicles (EDVs, e-C-VRP-TW) to calculate optimal solutions to the scheduling and routing of parcels from a warehouse to the LSP's clients. Within Porto LL, UDR has been utilized to estimate the number of EDVs required and the total distance to satisfy a logistics service provider's (LSP) customer demand. This is based on the locations of the clients, network distances, dispatch windows of the LSP, and the delivery windows required by the clients. UDR can operate on both historical and potentially real-time data, enabling digital twinning functionality. The results generated by UDR can be used to support decision-making regarding the establishment of 'small' orders (i.e., low volume) or rescheduling service by the LSP based on the required investment costs for EDVs. UDR has also been used together with UPM's EVCO2 to calculate/estimate the environmental impact of transport operations for the small orders service and the rescheduling case of SONAE.

2 Requirements.

2.1 Software requirements

The implementation of UDR is in Python (version 3.7) language and is dependent on a series of open Python libraries found in the requirements.txt file. Another UDR dependency is OpenTripPlanner routing engine, which is used for the calculation of real-world network distances based on OpenStreetMap data. A major component of UDR is the e-C-VRP-TW model, which is similar to VRP.

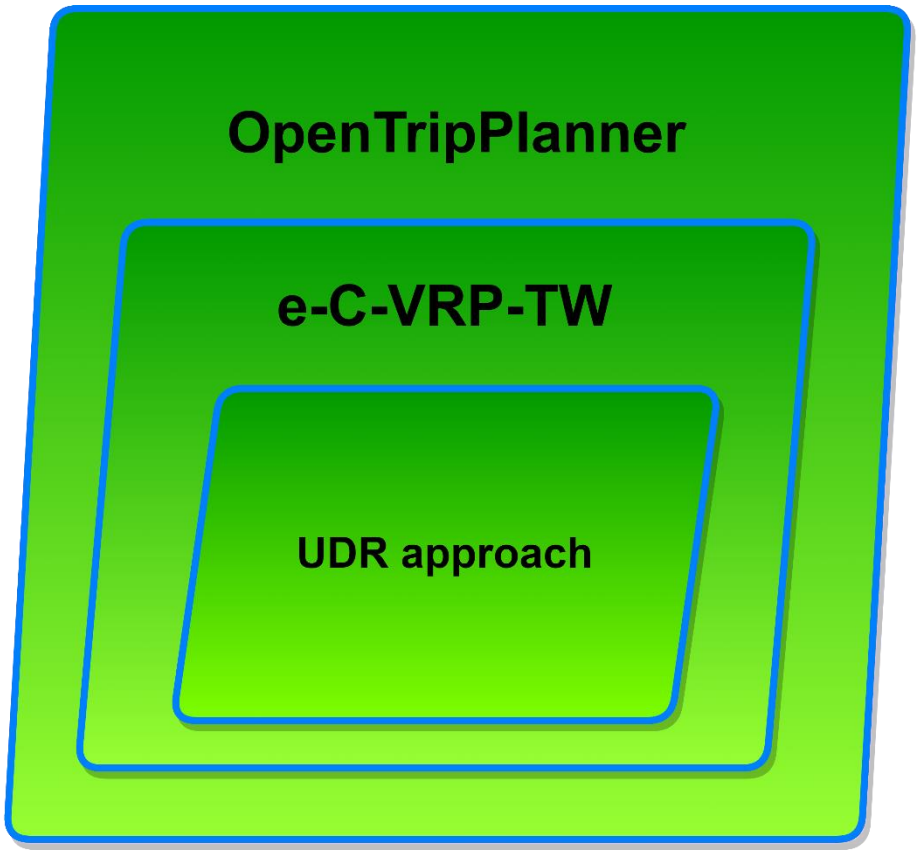


Figure 1: UDR's main components

2.2 Input/Outputs

2.2.1 Inputs

The UDR model has the following inputs and its outputs.

Table 1 UDR – Inputs

Inputs	Description
Parcel deliveries data	Past or real time data
Depot data	Data about depot locations and names (has to include a key ID that is connecting to the parcel deliveries file)
EDV Fleet specification	EDV max range (or consumption and battery capacity and number of batteries), load capacity, Number of EDVS used.

Delivery area postal codes	File containing number, IDs and centroid points of postal codes or city zones (may not be needed, depending on the parcel deliveries data, and if they anonymized, etc.)
O-D Matrices for road network distances between depot and clients	These data are the real road network distances that in this model are dynamically generated by the OpenTripPlanner routing engine.
OpenStreetMap data	.osm.pbf file in the case that you need to deploy the OpenTripPlanner API for the dynamic generation of O-D matrices.
Data filtering parameters	max volume per order (e.g. 3 boxes), keep only rescheduling orders (Boolean variable), max direct distance from depot (e.g. 35.000 meters), max number of delivery orders to keep (e.g. 10, 30 or 300)
Depot ID	In order to select a single depot from the dataset
Single Date or date range	Select date or data range for which the simulation will take place
LSP's dispatch windows	UDR model considers LSP dispatch windows and performs a temporal classification of deliveries and then runs e-C-VRP-TW in order to simulate real-world delivery.

2.2.2 Outputs

Table 2. UDR – Outputs

UDR can be run with two entry points start.py or run_experiments.py.

Outputs	Description
If **start.py** is run: EDVs required, kilometric distance traversed by EDVs	If you run a single instance of the problem, then based on the batch of deliveries data selected, the result is the required EDVs amnd kilometric distance to cover demand for the specific batch of deliveries.

	The module returns two values and it has been developed in order to be integratable with other ICT systems. In start.py the user can select a single day in the delivery orders dataset.
If **run_experiments.py** is run, then UDR returns an Excel spreadsheet with EDVs required and kilometric distances for the date range under simulation	With this module you can select a subset of the deliveries data given as input based on a date range (e.g. 2022/ 5/ 24 up to 2022/ 5/ 29, or 2022/1/2 up to 2022/4/30) and get results from a wide range of dates (e.g. monthly, weekly, etc.)

3 Model Description

This section describes the different files and scripts present in the model

File name	Location	Description
start.py	Root folder	One of the two entry points for the software. It is used when we want to run a selected day in the dataset, or a specific batch of orders (DT functionality).
run_experiments.py	Root folder	One of the two entry points for the software. It is used when we want to run a simulation over past data for an extended range of days.
read_file.py	Root folder	Reads file from hard drive. Also serializes it to hard drive, for faster loading times.

vrp_scenarios (folder)	Root folder/vrp_scenarios	Implementations of e-C-VRP-TW in the Google-OR library.
utilities (folder)	Root folder	Includes several python modules for data manipulation.
routing (folder)	Root folder	Includes all Python modules responsible for real-world routing results based on OpenTripPlanner and haversine distance if OpenTripPlanner fails.
pre_optimization	Root folder	Includes spatial clustering and temporal classification modules.
opentripplanner (folder)	Root folder	Includes all data and software related to the re-use of OpenTripPlanner
input (folder)	Root folder	This is the folder where all input files should be placed.
impact_assessment (folder)	Root folder	Includes all impact assessment modules or python script with API calls to impact assessment tools.
excel_results (folder)	Root folder	This is where the results of run_experiments.py are places after the software is run.

4 Instructions to run the model

4.1 Command line execution of the model

4.1.1 Instructions and commands

To run the model you need to setup two environments:

- Python 3.7 environment (e.g., based on Anaconda) and Python packages based on requirements.txt file,
- Java 8 Runtime Environment in order to run OpenTripPlanner routing engine through the .bat file located in root_folder/opentripplanner

4.1.2 Arguments

UDR takes no argument through the cmd, the right input files need to be places in the root_folder/input folder.

If the OpenTripPlanner Grizzly server instance is running (Java runtime environment is setup), if all of the input files are placed at the right folders, and if the Python 3.7 environment is setup, then you can just run UDR model by the command line.

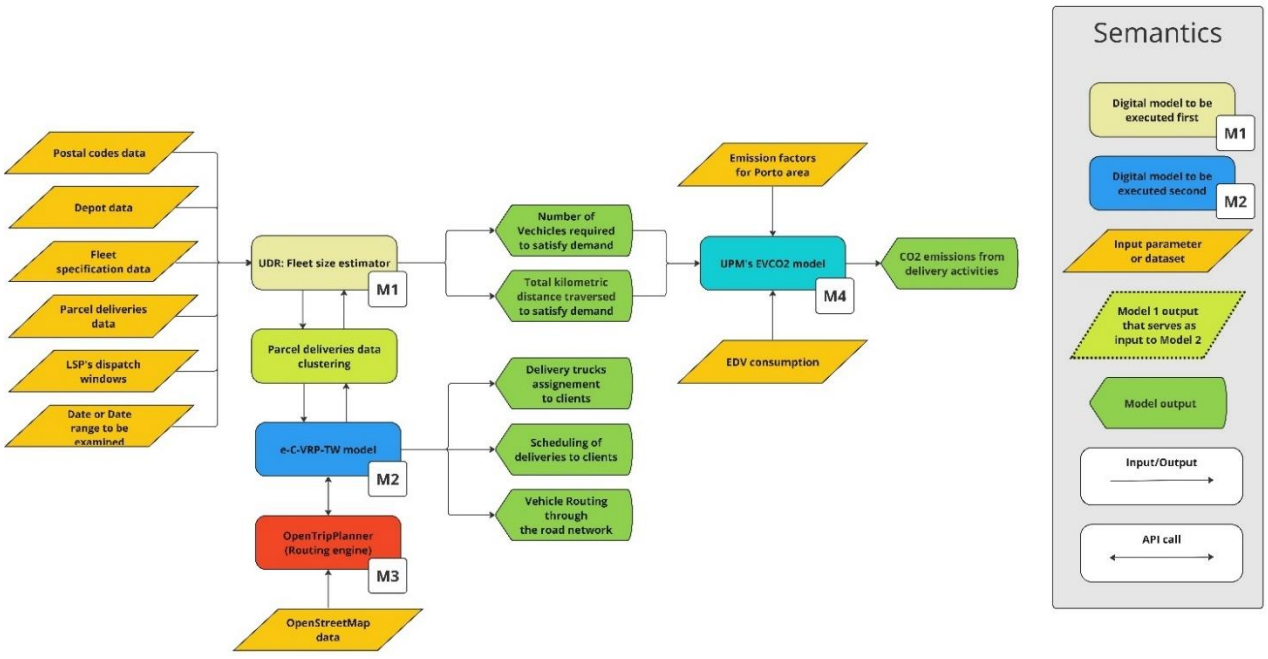


Figure 2: Model execution workflow of UDR as it has been integrated under a platform scenario

IRTX Synthetic population model

Introduction

The present model generates a synthetic representation of households, persons and their daily activity chains. While it can be applied to any region in France, it is configured to create a synthetic population for the Rhône-Alpes region around Lyon. In the project lead, the resulting synthetic population is used to (1) derive a demand data set for parcel deliveries in the region, and to (2) assess how delivering these parcels is affecting the local population using a detailed agent-based simulation. Both aspects are covered in downstream models.

The model is based on a methodology that has been published in

> Hörl, S., Balac, M., 2021. Synthetic population and travel demand for Paris and Île-de-France based on open and publicly available data. Transportation Research Part C: Emerging Technologies 130, 103291. https://doi.org/10.1016/j.trc.2021.103291

All code for the model is available on Github:

> https://github.com/eqasim-org/ile-de-france

The model is based entirely on open and publicly available data that can be downloaded from the respective distributing entities. The process of collecting all necessary data to create a synthetic population for Rhône-Alpes, as well as suggestions on configuring the model, are given in the same Github repository and users are expected to follow the detailed guidelines on where and how to obtain the individual data sets.

For the project LEAD and the special case of the Lyon Living Lab, adjustments have been made to the model. For instance, a module has been addded that scales up the population of the study area to the future expected resident count.

Requirements

All explanations in this document refer to the branch `lead` in the model's original Github repository and, specifically, the commit `51a0c37` is used throughout this document.

Software requirements

To run the model, the environment needs to be prepared:

- A `conda` environment needs to be set up in which the Python code of the model is run. The remote repository provides `environment.yml` which describes the `conda` environment and all dependencies:

```
conda env create -f environment.yml -n synpop
```

- Note that some of the dependencies installed via `conda > pip` need a recent compiler available on the system. Additionally, the MATSim instances that are run in the pipeline need to have access to the fonts available on the system. On an Ubunutu system, it suffices to `apt install build-essential fontconfig`.
- A `Java` runtime needs to be present on the executing machine. It is recommended to set up an **Adoptium OpenJDK 11** (https://adoptium.net).
- A recent version of `maven` needs to be installed, version `3.6.3` has been tested: https://maven.apache.org/
- A recent version of `git` needs to be installed on the system.
- Finally, `osmosis` needs to be installed, version `0.48.2` has been tested: https://github.com/openstreetmap/osmosis/releases

It is recommended to set up the environment on a Linux machine, the following executables should then be callable from the command line: `java`, `mvn`, `osmosis`, `git`.

A comprehensive example of setting up the environment for continuous integration can be found in the model repository at `.github/workflows/tests.yml`.

Input / Output

Input

How to collect all the necessary input data sets is described in detail in the model repository:

https://github.com/eqasim-org/ile-de-france/blob/lead/docs/cases/lyon.md

In brief the following data sets need to be collected:

- Census data (Rhône-Alpes cut-out)
- National origin/destination census data
- Population totals data
- Income tax data
- Service and facility census
- National household travel survey
- IRIS Zoning System
- Zoning registry
- Enterprise census (SIRENE)
- National Address Database (Rhône-Alpes cut-out)
- OpenStreetMap data (Rhoône-Alpes cut-out)
- GTFS data of the regional public transport operators

All data sets should be arranged as described in an arbitrary directory, from here on denoted as `/input`. Finally, `/input` should show the following structure:

```
Synthetic population:
/input/rp_2015/FD_INDCVIZE_2015.dbf
/input/rp_2015/FD_MOBPRO_2015.dbf
/input/rp_2015/FD_MOBSCO_2015.dbf
/input/rp_2015/base-ic-evol-struct-pop-2015.xls
/input/filosofi_2015/FILO_DISP_COM.xls
/input/filosofi_2015/FILO_DISP_REG.xls
/input/bpe_2021/bpe21_ensemble_xy.csv
/input/entd_2008/Q_individu.csv
/input/entd_2008/Q_tcm_individu.csv
/input/entd_2008/Q_menage.csv
/input/entd_2008/Q_tcm_menage_0.csv
/input/entd_2008/K_deploc.csv
/input/entd_2008/Q_ind_lieu_teg.csv
/input/iris_2017/CONTOURS-IRIS.cpg
/input/iris_2017/CONTOURS-IRIS.dbf
/input/iris_2017/CONTOURS-IRIS.prj
/input/iris_2017/CONTOURS-IRIS.shp
/input/iris_2017/CONTOURS-IRIS.shx
/input/codes_2017/reference_IRIS_geo2017.xls
/input/sirene/StockEtablissement_utf8.csv
/input/bdtopo_lyon/ADRESSE.shp
/input/bdtopo_lyon/ADRESSE.cpg
/input/bdtopo_lyon/ADRESSE.dbf²
/input/bdtopo_lyon/ADRESSE.prj
/input/bdtopo_lyon/ADRESSE.shx

Transport system:
/input/osm/rhone-alpes-latest.osm.pbf
/input/gtfs/lyon/GTFS_TCL.ZIP
/input/gtfs/lyon/CAPI.GTFS.zip
/input/gtfs/lyon/GTFS_RX.ZIP
/input/gtfs/lyon/SEM-GTFS.zip
/input/gtfs/lyon/stas.gtfs.zip
/input/gtfs/lyon/VIENNE.GTFS.zip
/input/gtfs/lyon/export_gtfs_voyages.zip
/input/gtfs/lyon/export-intercites-gtfs-last.zip
/input/gtfs/lyon/export-ter-gtfs-last.zip
```

For testing purposes, a prepackaged collection of those data sets can be obtained from IRT SystemX.

Configuration

To run the pipeline, a configuration file needs to be provided. An example is given in the LEAD repository as `data/template_lyon.yml`. The configuration file is in YAML format.

There are some **mandatory** settings that need to be updated in the configuration file. Those concern mostly paths in the file system. It is recommended to put absolute paths in the configuration file:

- `working_directory`: This option needs to point to an *existing* directory (otherwise the model will refuse to run) in which the pipeline will save all caching information as it will run a considerable number of models one after another. **Note that cached results in this working directory can be reused**: If the pipeline is run multiple times, possibly with varying optional configuration parameters, only those parts that are affected by the parameter changes are re-run and if, later, old parameter configuration should be re-run, the model will use cached information.
- `config.data_path`: This parameter should point to the directory in which all the data has been put, i.e. `/data` from the previous section.
- `config.output_path`: The option should point to an *existing* directory in which the model output will be written. From this directory, the results can be obtained later on.
- `config.lead_path`: The option should point to the `data` directory of the LEAD repository for the synthetic population model. It contains additional data for the Lyon living lab.

The following table covers **optional** attributes related to executing the model:

Parameter	Values	Description
`processes`	Integer (default `12`)	Number of parallel processes to use. Note that Python gets unstable on some systems for more that 12 processes.
`java_memory`	Integer + `G` (default `40G`)	Java is only needed when generating the inputs for the agent-based simulation. The pipeline will run two test iterations of the simulation, which may need up to 120GB of memory, but can be scaled down (see parameters below).
`output_prefix`	String (`lead_`)	All output files for a specific run that are put into the `output_path` will be prepended by `output_prefix`. This allows to have different scenarios / versions of the output.

Some parameters are related to the different scenarios that are examined using the LEAD platform:

Parameter	Values	Description
`random_seed`	Integer (default `1234`)	Allows to run the model with different random variations
`sampling_rate`	Real (default `1.0`)	Share of households that will be generated (lower numbers mean quicker run times)
`lead_year`	`2022` or `2030`	Defines for which year the synthetic population is generated.

Finally, the configuration can be adjusted using the `run` option. It describes a list of pipeline components that will be executed. For the Lyon case, only two components are relevant:

- `synthesis.output`: If this component is chosen, output for the synthetic population will be created in the `output_path`. Usually, it should be executed with a `sampling_rate` of `1.0`, because it will prepare data for the downstream parcel models.
- `matsim.output`: If this component is chosen, inputs for an agent-based transport simulation will be created in the `output_path`. Note that in this case the pipeline will automatically run two iterations of the simulation to verify the output. This requires substantial memory (up to 120GB RAM) if the full sampling rate is used. For the simulations, hence, smaller values for `sampling_rate` such as `0.05` are recommended.

To ease configuration for the project LEAD, `prepare_config.py` is provided with the synthetic population model in the LEAD repository. It allows to create a configuration file taking into account the relevant parameters for the project:

```
python3 prepare_config.py \
  --template-path /path/to/irtx-synpop/data/template_lyon.yml \
  --target-path config_XYZ.xml \
  --working-directory /path/to/working_directory \
  --data-path /path/to/data \
  --output-path /path/to/output \
  --lead-path /path/to/irtx-synpop/data \
  --processes 12 \
  --memory 40G \
  --output-prefix XYZ \
  --random-seed 1234 \
  --sampling-rate 1.0 \
  --lead-year 2022 \
  --generate-synthetic-population true \
  --generate-agent-based-simulation false
```

Note that, here, a specific configuration for the scenario `XYZ` is created. The option `template-path` corresponds to the template the is provided in the LEAD repository and `target-path` corresponds to where the prepared configuration file should be written. Also note the last two options that define which output components to active.

The resulting configuration file should be put in the root folder of the cloned model repository to execute it.

Output

The output depends on whether the output for the synthetic population and/or the input for the agent-based transport simulation are chosen to be created in the configuration (see above).

For the *synthetic population*, the following files are created (assuming `lead_` being the chosen prefix in the configuration):

- `lead_meta.json`: Meta information on when and how the synthetic population has been created
- `lead_households.csv`: Table containing socio-demographic information about all generated households
- `lead_homes.gpkg`: Geographic file containing the place of residence for all generated households
- `lead_persons.csv`: Table containing socio-demographic information about all generated persons
- `lead_activities.csv`: Table containing all activities performed during one day for each synthetic person
- `lead_activities.gpkg`: Same with geographic coordinate information
- `lead_trips.csv`: Table containing all trips performed during one day by each synthetic person
- `lead_trips.gpkg`: Same with geographic coordinate information
- `lead_commutes.gpkg`: Geographic information on all commutes of the synthetic population

For the *agent-based simulation*, the following files are created:

- `lead_population.xml.gz`: Information on the daily movements of the synthetic population
- `lead_network.xml.gz`: Describes the road and transit network
- `lead_facilities.xml.gz`: Describes facilities (appartments, stops, …)
- `lead_transit_schedule.xml.gz`: Describes the public transport schedules
- `lead_transit_vehicles.xml.gz`: Describes the public transport vehicles
- `lead_config.xml`: Pre-generated configuration file for the simulation

Running the model

First, the environment needs to be set up. All relevant dependencies should then be present, and the user can enter the generated `conda` environment:

```
conda activate synpop
```

As described above, the configuration process has been simplified. One can now call `prepare_config.xml` to create a configuration file, for instance, `lead_config.xml`.

The pipeline should be called from its root directory (the cloned model repository) and can be started by calling

```
cd /path/to/model
python3 -m synpp lead_config.xml
```

Depending on the configuration settings, the pipeline will run quite quickly or for a long time. Also remember that results in `working_directory` are cached, so subsequent calls will be much faster if not configuration parameters have been changed. In fact, if no changes are present at all, the pipeline will just return the previous obtained output.

Standard scenarios

For the Lyon living lab, some standard scenarios can be run:

- A full synthetic population (100%) for the years 2022 and 2030 that is used as input for the downstream parcel synthesis model.
- A reduced synthetic population (5%) for the year 2022 and 2030 that is used for the agent-based simulation.

Full synthetic population for 2022

```
--target-path config_2022_100pct.xml \
--output-prefix lead_2022_100pct_ \
--sampling-rate 1.0 \
--lead-year 2022 \
--generate-synthetic-population true \
--generate-agent-based-simulation false
```

Full synthetic population for 2030

```
--target-path config_2030_100pct.xml \
--output-prefix lead_2030_100pct_ \
--sampling-rate 1.0 \
--lead-year 2030 \
--generate-synthetic-population true \
--generate-agent-based-simulation false
```

Reduced synthetic population for 2022

```
--target-path config_2022_5pct.xml \
--output-prefix lead_2022_5pct_ \
--sampling-rate 0.05 \
--lead-year 2022 \
--generate-synthetic-population true \
--generate-agent-based-simulation true
```

Reduced synthetic population for 2030

```
--target-path config_2030_5pct.xml \
--output-prefix lead_2030_5pct_ \
--sampling-rate 0.05 \
--lead-year 2030 \
--generate-synthetic-population true \
--generate-agent-based-simulation true
```

IRTX Synthetic parcel model

Introduction

The present model uses a synthetic population to generate the daily parcel demand of a region. While it is based on marginal survey data from France, it can theoretically be applied to other use cases as well.

The basic idea is to take a list of synthetic (or real) households and persons including their sociodemographic attributes. Those should include *household size*, *socioprofessional category* (following the French defintiion), and *age* per person.

After, an Iterative Proportional Fitting procedure is applied to the data to attach an average number of parcels per year to each household, based on the sociodemographic attributes. This value is then transformed into an average number of parcels per day for each household, which, in a final step, is transformed into a discrete value using a Poisson sampling process. The methodology is described in detail in

> Hörl, S., Puchinger, J., 2022. From synthetic population to parcel demand: A modeling pipeline and case study for last-mile deliveries in Lyon. Paper accepted for presentation at the Transport Research Arena (TRA) 2022, November 2022, Lisbon.

Requirements

Software requirements

The model is packaged as a `Jupyter` notebook. All dependencies to run the model have been collected in a `conda` environment, which is available in the LEAD repository as `environment.yml`:

```
conda env create -n parcels -f environment.yml
```

Input / Output

Input

To run the model, a synthetic population (for instance, as created by the IRTX population synthesis model), needs to be located in an arbitrary directory denoted by `/irtx-synpop/output`. The structure of this directory should look as follows:

```
/irtx-synpop/output/lead_homes.gpkg
/irtx-synpop/output/lead_persons.csv
/irtx-synpop/output/lead_activities.csv
```

Output

The output of the model is one geographic file that contains for each household with at least one parcel that has been generated for the synthetic day:

- Coordinates (location) of the household
- Number of parcels to be delivered
- Identifier of the household to link back to the synthetic population

Assuming that `/irtx-parcels/output` has been defined as the output path of the model (see below), the resulting file will be created as `/irtx-parcels/output/lead_parcels.gpkg`.

Running the model

To run the model, the `conda` environment needs to be prepared and entered. After, the model is packaged as a `jupyter` notebook which can be run programmatically using `papermill`, which is part of the environment dependencies. Including the full set of command line options, the model can be run as follows:

```
papermill "Generate Parcels.ipynb" /dev/null \
  -pinput_path /irtx-synpop/output \
  -poutput_path /irtx-parcels/output \
  -pinput_prefix lead_ \
  -poutput_prefix lead_ \
  -prandom_seed 0 \
  -pscaling 1.0
```

The **mandatory** parameters are detailed in the following table:

Parameter	Values	Description
`input_path`	String	Path to the input directory containing the synthetic population
`output_path`	String	Path to the output directory into which the parcel data will be written (can be the same)

The following **technical** parameters are available:

Parameter	Values	Description
`input_prefix`	String (default `lead_`)	Defines which prefix the population input files have
`output_prefix`	String (default `lead_`)	Defines which prefix the parcel output file will have

Using these parameters, one can, for instance, process `xyz_persons.csv` and write `abc_parcels.gpkg` in the same directory, depending on which population scenarios should be read and which parcel scenario should be written.

Finally, **scenario** parameters exist that can be configured:

Parameter	Values	Description
`random_seed`	Integer (default `0`)	Allows to generate instances with different random variation
`scaling`	Real (default `1.0`)	Allows to uniformly scale up or down the total parcel demand

Standard scenarios

For the Lyon living lab, some standard scenarios can be run:

- Using an unscaled output from the population synthesis model to create the baseline parcel demand for 2022.
- Using an unscaled output from the population synthesis model to create the parcel demand for 2030, with an increase of parcel by a factor of two.

The scaling factor could also be an input to create multiple different scenarios on the LEAD platform.

Baseline demand 2022

```
-pinput_path /irtx-synpop/output \
-poutput_path output \
-pinput_prefix lead_2022_100pct_ \
-poutput_prefix lead_2022_ \
-pscaling 1.0
```

Future demand 2030

```
-pinput_path /irtx-synpop/output \
-poutput_path output \
-pinput_prefix lead_2030_100pct_ \
-poutput_prefix lead_2030_ \
-pscaling 2.0
```

IRTX MATSim Implementation

Introduction

This model is a wrapper around the multi-agent transport simulation framework MATSim:

https://matsim.org/

MATSim is a framework used by more than 50 research institutes world-wide to set up a wide variety of transport simulations that are able to model the dynamic interaction between travelers and mobility service providers. There are extensions to simulate car-sharing services, on-demand mobility, micromobility and standardized analysis tools for emissions, noise and externality-based traffic control.

A MATSim simulation consists of multiple iterations. In each iteration, a mobility simulation with all agents is performed, leading to phenomena such as congestion. In a second step, agents make decisions based on the observed traffic characteristics, for instance, they might choose a different mode of transport for certain trips in their daily scheduled. This way, agents adapt to the traffic conditions. By running this loop between mobility simulation and decision-making, the decisions and daily mobility pattern go into equilibrium with the traffic conditions. The final iteration can be observed and indicators and visualisations can be derived from it.

For the LEAD project, only a part of the overall functionality has been packaged and customized to be used mainly in the Lyon living lab. However, the model remains sufficiently generic and allows to simulate a city (given that the relevant input data is provided) with the mobility traces of all its inhabitants. Additionally, the LEAD repository adds the functionality to add commercial vehicles that have been generated in an upstream model such as JSprit.

Specifically, the model for Lyon is based on the synthetic population data generated by the synthetic population model and the vehicles traces generated by JSprit. In order to generate those inputs, see the documentation of the upstream models.

The MATSim model prepares the output for downstream emissions and noise analysis. Furthermore, indicators on congestion can be obtained.

Requirements

Software requirements

To run the model, the environment needs to be prepared:

- A `conda` or `mamba` environment needs to be set up in which the Python code of the model is run. The LEAD repository provides `environment.yml` which describes the `conda` environment and all dependencies.

- Note that some of the dependencies installed via `conda > pip` need a recent compiler available on the system. Additionally, the MATSim needs to have access to the fonts available on the system. On an Ubunutu system, it suffices to `apt install build-essential fontconfig`.
- A `Java` runtime needs to be present on the executing machine. It is recommended to set up an **Adoptium OpenJDK 11** (https://adoptium.net).
- A recent version of `maven` needs to be installed, version `3.6.3` has been tested: https://maven.apache.org/

It is recommended to set up the environment on a Linux machine, the following executables should then be callable from the command line: `java`, `mvn`.

Input / Output

The model can be used in two different setups:

- Running a baseline simulation for a larger region (Rhône-Alpes around Lyon in the case of the Lyon living lab). Such a simulation is run to establish a baseline congestion scenario where all agents adapt their itineraries according to the endogoenous travel times.
- Running a cut simulation for a focus area (Confluence inside Lyon in the case of the Lyon living lab). To do so, the converged regional simulation can be cut such that subsequent policy scenarios only need to be simulated on the smaller study area. In such a simulation, modifications can be added to simulate policy cases, e.g., including additional commercial vehicle traces as in the present case.

Input

Baseline simulation

In order to perform the regional baseline simulation, MATSim-compatible input data needs to be provided. This input data contains the following files (usually with an added prefix to distinguish between scenarios). They are provided as plain or compressed XML files:

- `config.xml`: Provides configuration information to the simulation and references all the other files mentioned below.
- `network.xml.gz`: Describes the network topology
- `households.xml.gz`: Describes all households that are present in the simulation including sociodemographic attributes
- `population.xml.gz`: Contains information on all the individual persons that are simulated including their daily mobility patterns with activities and trips in between
- `facilities.xml.gz`: Localizes addresses and other private and public facilities that are referenced in the mobility chains
- `transit_schedule.xml.gz`: Describes the public transport offer in the simulation area

- `transit_vehicles.xml.gz`: Describes the public transport vehicles that are offering the defined transit services

These input files can be generated, for instance, using the population synthesis model provided as a downstream stage in the LEAD modeling library. A detailed description of their contains is out of scope at this place as they are rather complex. Detailed information can be found in the MATSim Book:

> Horni, A., Nagel, K., Axhausen, K.W. (Eds.), 2016. The Multi-Agent Transport Simulation MATSim. Ubiquity Press. https://doi.org/10.5334/baw

Note that the regional simulation usually only needs to be run once and all detailed analyses are performed on a smaller cut-out. Further below it is described how such a cut out can be created using the wrapped model. In case such a cut-out is to be created, a geographic perimeter needs to be defined using an input file in Shapefile format. For the Lyon use case, as respective file is provided in `data/perimeter_lyon.shp`.

Commercial vehicle simulation

To run the local simulations and include the commercial vehicles from the LEAD pipeline, output data from the JSprit - MATSim connector needs to be provided in JSON format, e.g. `traces.json`. This input file is structured as follows:

```
{
  "vehicle_types": [
    { "id": "van", "speed_km_h": 40.0 }
  ],
  "vehicles": [
    {
      "vehicle_type": "van",
      "stops": []
    }
  ]
}
```

First, a list of vehicle types is given, after a list of individual vehicles with their stops. Each vehicle refers to a vehicle type which is defined by its maximum road speed. Furthermore, each vehicle has a list of stops, which are defined as follows:

```
{
  "type": "start",
  "arrival_time": 28800.0,
  "departure_time": 28800.0,
  "location": {
    "x": 841484.6518412721,
    "y": 6517523.922895046
  }
}
```

Each stop has a type that can either be `start` (for the initial stop at the depot), `end` (for the final stop at the depot), `pickup` (when picking up an item), `delivery` (when delivering an item). For each stop, an arrival time and departure time is given, which is defined by the upstream JSprit and connector models. Finally, the location of the stop is defined using coordinates `x` and `y`. Not that these are not longitude and latitude as in the upstream models, but projected coordinates using a coordinate reference system that needs to be the same as the one in which all other model files (`population.xml.gz`, `network.xml.gz`) are described. For the specific use case of Lyon the French standard projection `EPSG:2154` is used.

Output

The output of a MATSim simulation is a directory with various information. The most important output is a file called `events.xml.gz` which contains all the individual events that happened in the simulation (agent enters/leaves link, agent starts/ends a trip/activity, …). It is the major output file from which second-order analyses can be derived. The relevant files for the use of the MATSim model in LEAD are:

- `trips.csv`: A file containing all trips that happened during the simulation including their mode of transport and the covered distance. Note that each commercial vehicle type defined in the input files is represented as one individual mode identified as `freight:{vehicle_type_id}`. The trips file is also used to generate the input for the downstream noise and emission models using the respective connectors.
- `congestion.csv`: A file specifically develope in the MATSim implementation for LEAD which compares all `car` trips in terms of their recorded travel time in the simulation and the direct freeflow travel time of the same trip. The LEAD repository contains a script that allows to generate high-level congestion KPIs based on this file (see below).
- `output_plans.xml.gz`: Contains the final daily mobility plans of all agents. This file is used as a basis for cutting a smaller scope simulation from a larger one.

To generate the aggregated congestion information, the notebook `Congestion Analysis.ipynb` can be used (see below). The output of this notebook is a `json` file that is structued as follows:

```
{
  "total_delay_min": 114752.59,
  "delay_per_driver_min": 14.66
}
```

The first slot, `total_delay_min` gives the total delay that has been accumulated by the population compared to a freeflow travel time for their car trips in minutes. The second slot `delay_per_driver_min` divides this value by the number of people that have used a car during the day.

Building the model

The MATSim model is provided as Java code. To run it, it first needs to be built using the Maven build system. For that purpose, one needs to enter the `java` directory of the LEAD repository and call `mvn package`:

```
cd /irtx-matsim/java
mvn package
```

The build process should download all necessary Maven dependencies including the MATSim and finish without errors. After, the built model should be present in

```
/irtx-matsim/java/target/lead-matsim-1.0.0.jar
```

The `jar` file can be saved in a fixed location. As long as the model is not changed, it can be reused for multiple model runs. To test whether the `jar` has been build successfully, call

```
java -cp /irtx-matsim/java/target/lead-jsprit-1.0.0.jar
fr.irtx.lead.matsim.RunVerification
```

which should respond by the message `It works!`.

Running the model

The model repository provides the code to run the MATSim model itself and to generate congestion KPIs. Note that in the proposed LEAD use case for Lyon, the MATSim model is called twice (see above): Once to create a converged simulation state for the Rhône-Alpes region around Lyon, and once (or multiple times) to run a much smaller cut-out of the Confluence half-island with varying simulation parameters.

The model, hence, provides three functionalities: - Run a MATSim simulation - Run a simulation perimeter - Run congestion analysis

These are described in detail in the following.

MATSim simulation

To run a MATSim simulation, the respective jar needs to be built first. We assume that the configuration file of the underlying MATSim scenario is located at `/path/to/config.xml`. The simulation can then be started in the following way:

```
java -Xmx20g -cp /irtx-matsim/java/target/lead-matsim-1.0.0.jar
fr.irtx.lead.matsim.RunSimulation \
  --config-path /path/to/config.xml \
  --output-path /path/to/output_directory \
  --threads 8 \
  --iterations 120 \
  --freight-path /path/to/traces.json
```

The first line is mandatory with the path to the built `jar` file that needs to be adapted. The following lines represent parameters. The **mandatory** parameters are detailed in the following table:

Parameter	Values	Description
`--config-path`	String	Path to the configuration file
`--output-path`	String	Path to where the result will be saved

The following **optional** parameter is available:

Parameter	Values	Description
`--freight-path`	String	Path to the vehicle traces of the additional commercial vehicles

Finally, **technical** parameters exist that can be configured:

Parameter	Values	Description
`--random-seed`	Integer (default `1234`)	Allows to perform ensemble runs by providing different initialization seeds of the optimization
`--iterations`	Integer (default `120`)	Sets the number of iterations per operator with higher accuracy with increasing values (but also higher runtime)
`--threads`	Integer (default `12`)	Allows to set the number of threads that are used for the optimization

Note that the memory available to Java can be configured using the `-Xmx` option and by appending a size of the format `1024M` to define the amount in megabytes or `20G` to define the amount in gigabytes.

Cutting a simulation

Using another run class in from the same `jar` file, one can cut a generated simulation as follows:

```
java -Xmx20G -cp /irtx-matsim/java/target/lead-matsim-1.0.0.jar
org.eqasim.core.scenario.cutter.RunScenarioCutter \
  --config-path /path/to/config.xml \
  --extent-path /path/to/perimeter.shp \
  --output-path /path/to/output_directory \
  --prefix perimeter_2022_5pct_ \
  --config:plans.inputPlansFile /path/to/output_plans.xml.gz \
  --threads 12
```

The **mandatory** parameters are detailed in the following table:

Parameter	Values	Description
`--config-path`	String	Path to the configuration file of the original simulation
`--extent-`	String	Path to the geographic file describing the extent of the cut-out

Parameter	Values	Description
`path`		(must be in the same CRS as the simulation data)
`--output-path`	String	Path to a directory into which the cut simulation data will be saved. *Must exist.*

The following **optional** parameter is available:

Parameter	Values	Description
`--prefix`	String	All generated files (network, population, …) will be prepended by this prefix
`--threads`	Integer (default `12`)	Number of threads to be used for cuting
`--config:plans.inputPlansFile`	String	Path to the `output_plans.xml.gz` file of the output of the converged simulation that should be used as a basis for cutting. *Attention: It is recommended to provide an absolute path here, as any relative path is interpreted as relative to the configuration file*.

After running the cutter, you'll find the exact same input files for a MATSim simulation as described above in the `output-path`, possibly with a custom prefix according to the command line parameters. This simulation can then be restarted as a standard MATSim simulation (see above).

Congestion analysis

Finally, the repository contains `Congestion.ipynb`, a notebook which allows to generate aggregated congestion indicators from the MATSim simulation. To run it, call it through the `papermill` command line utility (which is installed as a dependency in the `conda` environment) as described below:

```
papermill "Congestion Analysis.ipynb" /dev/null \
  -psimulation_output_path /path/to/irtx-matsim/output \
  -pkpi_path /path/to/congestion_kpi.json \
  -pcutoff_min 60.0
```

The **mandatory** parameters are as follows:

Parameter	Values	Description
`simulation_output_path`	String	Path to the output directory of a MATSim simulation
`kpi_path`	String	Path where to save the aggregated congestion information

There is one **optional** parameter:

Parameter	Values	Description

Parameter	Values	Description
`cutoff_min`	Real (default `60`)	To avoid outliers, it defines a cutoff value for the observed delays

Standard scenarios

For the Lyon living lab, some standard simulations can be run. They are based on the output of the synthetic population model (Population 2022, Population 2030) and the output of the JSprit scenarios (Baseline 2022, UCC 2022, UCC 2030).

Baseline

First, the baseline simulations can be run, based on the synthetic population output. The command to do so is:

```
java -Xmx20g -cp /irtx-matsim/java/target/lead-matsim-1.0.0.jar
fr.irtx.lead.matsim.RunSimulation \
  --config-path /irtx-synpop/output/lead_{year}_5pct_config.xml \
  --output-path /irtx-matsim/output/output_lead_{year}
```

Here, `year` can be replaced by `2022` or `2030`.

Cutting

Both cases can be cut to the Lyon Confluence area. For that, the perimeter is provided in the repository as `scenario_data/perimeter.shp`. The command is as follows:

```
java -Xmx20G -cp java/target/lead-matsim-1.0.0.jar
org.eqasim.core.scenario.cutter.RunScenarioCutter \
  --config-path /irtx-synpop/output/lead_{year}_5pct_config.xml \
  --extent-path data/perimeter_lyon.shp \
  --output-path /irtx-matsim/output \
  --prefix perimeter_{year}_ \
  --config:plans.inputPlansFile /irtx-
matsim/output/output_lead_{year}/output_plans.xml.gz
```

Again, `year` can be replaced by `2022` or `2030`. Note that the baseline simulation and cutting only needs to be redone when the upstream synthetic population is changed.

Scenario simulation

Based on the cut perimeters, the local simulations for Confluence with the three scenarios from the JSprit model can be run. First for *Baseline 2022*:

```
java -Xmx20g -cp /irtx-matsim/java/target/lead-matsim-1.0.0.jar
fr.irtx.lead.matsim.RunSimulation \
  --config-path /irtx-matsim/output/perimeter_2022_config.xml \
  --output-path /irtx-matsim/output/output_baseline_2022 \
  --freight-path /irtx-jsprit-matsim-
connector/output/traces_baseline_2022.json
```

Then for *UCC 2022* and *UCC 2030* with {year} replaces by the respective year:

```
java -Xmx20g -cp /irtx-matsim/java/target/lead-matsim-1.0.0.jar
fr.irtx.lead.matsim.RunSimulation \
  --config-path /irtx-matsim/output/perimeter_{year}_config.xml \
  --output-path /irtx-matsim/output/output_ucc_{year} \
  --freight-path /irtx-jsprit-matsim-connector/output/traces_ucc_{year}.json
```

Congestion

For each scenario, the congestion KPIs can be calculated:

```
papermill "Congestion Analysis.ipynb" /dev/null \
  -psimulation_output_path /irtx-matsim/output/output_{scenario} \
  -pkpi_path /irtx-matsim/output/congestion_{scenario}.json
```

Replace {scenario} = baseline_2022 | ucc_2022 | ucc_2030.

IRTX JSprit Implementation

Introduction

The JSprit model makes use of the open-source route optimization library JSprit:

> https://jsprit.github.io/

During the LEAD project it has been adapted to the specific use case of the Lyon living lab, but with the ambition to provide a generic route and fleet size optimization tool that can be applied to other cases. The individual uses and new developments around JSprit in LEAD have been documented in

> Mahmoud, A., Chouaki, T., Hörl, S., Puchinger, J., 2022. Adapting JSprit for the Electric Vehicle Routing Problem with Recharging: Implementation and Benchmark. International Journal for Traffic and Transport Engineering 12 (3), 340-351. http://dx.doi.org/10.7708/ijtte2022.12(3).04

The model has been streamlined to work on configurable scenarios that define:

- Vehicle types with their individual properties such as daily and per-distance costs, speeds, energy consumption and emissions
- Operators with their individual distribution centers (by coordinate) and demand to fulfill during one day (by coordinate) and a list of vehicle types that are available at their depots

Optionally, an Urban Consolidation Center (UCC) can be defined with its location (by coordinate) and its available vehicle types. For each conventional operator (see above), it can be defined whether deliveries should be shipped through the UCC if it is defined.

The model, hence, allows to build a rich set of shipment scenarios. In the baseline case, various operators can be defined. The model will then optimize the fleet composition (based on the available vehicle types) and the driven distance by minimizing the total cost of each operator individually. After, a consolidation scenario can be developed in which all or some operators need to pass their deliveries through the UCC, which, analogously to the operators, minimizes its total cost.

Additional scenarios can be constructed, for instance, by changing the cost structures and observing trade-offs between vehicle types.

Routing-based costs are either approximated using Euclidean distance or calculated directly based on OpenStreetMap road network data.

Requirements

Software requirements

To run the model, the environment needs to be prepared:

- A `conda` or `mamba` environment needs to be set up in which the Python code of the model is run. The LEAD repository provides `environment.yml` which describes the `conda` environment and all dependencies.
- Note that some of the dependencies installed via `conda > pip` need a recent compiler available on the system. Additionally, the MATSim instances that are run in the pipeline need to have access to the fonts available on the system. On an Ubunutu system, it suffices to `apt install build-essential fontconfig`.
- A `Java` runtime needs to be present on the executing machine. It is recommended to set up an **Adoptium OpenJDK 11** (https://adoptium.net).
- A recent version of `maven` needs to be installed, version `3.6.3` has been tested: https://maven.apache.org/
- Finally, `osmosis` needs to be installed, version `0.48.2` has been tested: https://github.com/openstreetmap/osmosis/releases

It is recommended to set up the environment on a Linux machine, the following executables should then be callable from the command line: `java`, `mvn`, `osmosis`.

Input / Output

Input

The model needs various input data sets. First, network data needs to be provided to perform the distance routing, this includes defining the perimeter of the study area. Second, information on the operators needs to be provided including their distribution center locations and demand data. Finally, the operator data is merged into a scenario configuration. To ease the process of setting up a scenario, we provide a command line tool that makes it easy to join this operator information and change values in the scenario configuration using command line parameters.

Network

In order to define the network topology on which the deliveries will be routed, data from OpenStreetMap needs to be provided in `pbf` format. Regular snapshots of OpenStreetMap data are available publicly from Geofabrik.

Processing the whole OpenStreetMap each time the model is run would take too much time, so it is recommended to trim the data to the specific use case area. This task can be achieved by the `osmosis` command line tool, which, however needs a specific non-standard format that describes the perimeter to cut. For that purpose, we provide the `prepare_perimeter.py` script, which can be called as follows once one has entered the `conda` environment:

```
python3 prepare_perimeter.py \
  --input-path /path/to/perimeter.gpkg \
  --output-path /path/to/perimeter.poly
```

The input perimeter can be provided in any common geographic data format that is understood by `geopandas` / `fiona` such as Shape file (`shp`) or GeoPackgage (`gpkg`).

The resulting `poly` file can then be passed to `osmosis` to cut the case study area. The command is packaged in `prepare_osm.sh` which can be called as follows:

```
sh prepare_osm.sh \
  /path/to/openstreetmap.osm.pbf \
  /path/to/perimeter.poly \
  /path/to/reduced.osm.pbf
```

Note that these steps only need to be performed once per use case. Once the OpenStreetMap data has been converted they serve as input to any subsequent model execution. Concrete examples and data for the Lyon living lab of the LEAD project are given further below.

Operators

Data for multiple operators that are active on the territory can be provided. Each operator is represented in a separate `json` file with the following exemplary format:

```
{
  "id": "my_operator",
  "center": { "lat" : 45.7327, "lng" : 4.8245 },
  "vehicle_types": ["van"],
  "shipment_type": "delivery",
  "consolidation_type": "none",
  "demand": [
    { "lat" : 45.741532014731064, "lng" : 4.822740554809571 },
    { "lat" : 45.74407774585035, "lng" : 4.816946983337403 },
    { "lat" : 45.73954027356843, "lng" : 4.818427562713624 }
  ]
}
```

First, an internal identifier is given for every opeartor (`id`), second, the location of its distribution center is given as latitude and longitude. After, a list of *available* vehicle types is defined (see below). The file ends with the `demand` section which describes the individual shipments that need to be fulfilled by the operator during one day by coordinate.

An important part is the definition of the *shipment type* and the *delivery type*. The *shipment type* describes how goods are moved between the distribution center and the final destinations. If it is set to `delivery`, the operator will deliver (using the available vehicle types) all goods to the destinations. In case it is set to `pickup`, each destination will use one of the available vehicle types to pick up the delivery. Usually, this clearly leads to a much higher number of vehicles and movements (but allows, for instance, the representation of common logistics schemes for construction sites today). Finally, the value can be set to `none` indicating that no direct movements from the distribution center. This allows, for instance, to construct scenarios in which all operations are entirely moved to the a third-party consolidation center.

The *consolidation type* is set to `none` by default. This means that the relations between the distribution center and the destinations are handled directly by the receivers or the operator. In case it is set to `delivery`, the goods will be delivered to the final destinations from the UCC using its defined vehicle types. In case it is set to `pickup`, the receivers will use their defined vehicle types to pick up the goods at the UCC. In any case, if any UCC *consolidation type* is chosen, the *shipment type* of the operator defines how the goods arrive at the UCC in the first place. This means that one can define situations in which the operator either delivers the goods to the UCC, the UCC picks up the goods at the operator's distribution center, or no interaction between the operator center and the UCC is happening.

Scenario

A scenario is described through a `json` file of the following format:

```
{
  "vehicle_types": [
    {
      "id": "van",
      "capacity" : 20,
      "cost_per_day_EUR" : 85.0,
      "cost_per_km_EUR" : 0.15,
      "co2_per_km_g" : 130.0,
      "energy_per_km_Wh" : 120.0,
      "speed_km_h": 40.0,
      "euclidean_distance_factor": 1.3,
      "active_time": 28800.0
    }
  ],
  "ucc": {
    "location": { "lat" : 45.74243052132232, "lng" : 4.824800491333009 },
    "vehicle_types": [ "cargobike", "van" ]
  },
  "operators": []
}
```

On top, multiple vehicle types with individual identifiers can be defined that differ in their capacity (number of items that can be carried), their cost structure, emissions, energy consumption, and speed. The `euclidean_distance_factor` is added to any distance in case the model is run *without* specific network data (see below). The `active_time` parameter defines within which time limit (usually one work day, the vehicle can operatate). After, the characteristics of the `ucc` are defined by defining a location in latitude and longitude and its available vehicle types.

Finally, a list of operators is given. Each operator follows exactly the configuration scheme as presented above. One could, hence, manually build a complete scenario by copy-pasting the operator information directly into the scenario file. To ease the process, a command line-based utility is provided alongside the model (see below).

Configuration

To configure a scenario, the provided script `prepare_scenario.py` can be used which is called as follows:

```
python3 prepare_scenario.py \
  --scenario-path /path/to/base_scenario.json \
  --output-path /path/to/output_scenario.json \
  --operator-path /path/to/operator1.json \
  --operator-path /path/to/operator2.json \
  --operator-path /path/to/operator3.json \
  --shipment-type:operator1 pickup \
  --consolidation-type:operator2 delivery \
  --vehicle-type:van:cost_per_km_EUR 0.3
```

The scenario preparation script has the following **mandatory** parameters:

The **mandatory** parameters are detailed in the following table:

Parameter	Values	Description
`--scenario-path`	String	Path to the baseline scenario
`--output-path`	String	Path to the updated output scenario

The following **optional** parameters exist that can be configured.:

Parameter	Values	Description
`--operator-path`	String	Integrates a new operator defined in a `json` file into the scenario. *Can be set multiple times to integrate multiple operators.*
`--shipment-type:{operator}`	`delivery`* or `pickup`	Sets the shipment type for operator `{operator}` (see above)
`--consolidation-type:{operator}`	`none`* or `delivery` or `pickup`	Sets the consolidation type for operator `{operator}` (see above)
`--driver-salary:{operator}`	Real	Sets the daily salary per driver in EUR
`--vehicle-type:{vt}:{property}`	Any	Sets a property of vehicle type `{vt}`, for instance `speed_km_h`

Concrete use cases for the utility in the context of the Lyon living lab will be provided further down in the *Standard scenarios* section.

Output

The output of the model is provided as a `json` file of the following format:

```
{
  "cost_EUR" : 235.27880397887634,
  "energy_kWh" : 0.2788039788763593,
  "co2_kg" : 0.2788039788763593,
  "distance_km" : 0.2788039788763594,
  "runtime_s" : 7.006087272,
  "routes" : []
}
```

It contains various KPIs such as the total cost of the system, the energy consumed, the CO2 emitted, and the distance driven. Furthermore, the runtime of the model is given.

The `routes` part gives a more detailed analysis per vehicle, in the following format:

```
{
  "cost_EUR" : 65.27,
  "energy_kWh" : 0.27,
  "co2_kg" : 0.27,
  "distance_km" : 0.27,
  "vehicle_type" : "cargobike",
  "carrier" : "operator_id",
  "trajectory" : [
    { "lat" : 45.74243052132232, "lng" : 4.824800491333009, "t0" : 0.0, "t1":
60.0 },
    { "lat" : 45.74199624494206, "lng" : 4.820101261138917, "t0" : 900.0,
"t1": 960.0 }
  ]
}
```

The first fields give the same information as the overall KPI analysis, but specifically for an individual vehicle. The vehicle type of that vehicle is given after, as well as the operator to which it belongs. The `carrier` may be `$ucc$` to indicate that the vehicle is operated by the UCC. Finally, a detailed trajectory of the vehicle including coordinates and timestamps is given.

Building the model

The model is provided as Java code. To run it, it first needs to be built using the Maven build system. For that purpose, one needs to enter the `java` directory of the LEAD repository and package and call `mvn package`:

```
cd /irtx-jsprit/java
mvn package
```

The build process should download all necessary Maven dependencies including the JSprit library and finish without errors. After, the built model should be present in

```
/irtx-jsprit/java/target/lead-jsprit-1.0.0.jar
```

The `jar` file can be saved in a fixed location. As long as the model is not changed, it can be reused for multiple model runs. To test whether the `jar` has been build successfully, call

```
java -cp /irtx-jsprit/java/target/lead-jsprit-1.0.0.jar
fr.irtx.lead.jsprit.RunVerification
```

which should respond by the message `It works!`.

Running the model

Once the model is built, it can be called the following way:

```
java -Xmx12g -cp /irtx-jsprit/java/target/lead-jsprit-1.0.0.jar
fr.irtx.lead.jsprit.RunSolver \
  --problem-path /path/to/scenario.json \
  --solution-path /path/to/solution.json \
  --crs EPSG:2154 \
  --osm-path /path/to/scenario.osm.pbf
```

The first line is mandatory with the path to the built `jar` file that needs to be adapted. The following lines represent parameters. The **mandatory** parameters are detailed in the following table:

Parameter	Values	Description
`--problem-path`	String	Path to the scenario file
`--solution-path`	String	Path to where the result will be saved
`--crs`	String	A geographic projection

The coordinate reference system (CRS) must be provided such that the model can performed Euclidean distance calculations on the coordinates and network data. The projection is use-case specific. For France, the standard geographic projection is `EPSG:2154`.

The following **optional** parameters are available:

Parameter	Values	Description
`--osm-path`	String	Path to the filtered OSM data.
`--freespeed-factor`	Real (default `0.7`)	A factor that is added to the nominative road speeds to simulate congestion

Note that if no path to the OpenStreetMap data is given, the model will perform Euclidean distance calculations with a per-vehicle-type factor.

Finally, **technical** parameters exist that can be configured:

Parameter	Values	Description
`--random-seed`	Integer (default `1234`)	Allows to perform ensemble runs by providing different initialization seeds of the optimization
`--iterations`	Integer (default `10000`)	Sets the number of iterations per operator with higher accuracy with increasing values (but also higher runtime)
`--threads`	Integer (default `1`)	Allows to set the number of threads that are used for the optimization

Note that the memory available to Java can be configured using the `-Xmx` option and by appending a size of the format `1024M` to define the amount in megabytes or `20G` to define the amount in gigabytes.

For test runs, it is recommended to divert from the standard value of `10000` iterations that is configured. Values like `1000` or `2000` will give results much quicker.

Standard scenarios

For the Lyon living lab, some standard scenarios can be run. Initially, the preparation steps need to be performed. After, individual scenarios can be evaluated/

Preparation

Network

In the specific case for Lyon, the latest snapshot of the Rhône-Alpes region provided by Geofabrik can be used:

> https://download.geofabrik.de/europe/france/rhone-alpes.html

For the specific case of Lyon, the perimeter is provided in the `data` directory in the LEAD repository. Hence, the correct `poly` file can be created by calling

```
python3 prepare_perimeter.py \
  --input-path /irtx-jsprit/data/perimeter_lyon.gpkg \
  --output-path /irtx-jsprit/output/perimeter_lyon.poly
```

This file can be used with `osmosis` to cut the relevant perimter for the Confluence study area in Lyon with the resulting file `scenario.osm.pbf`:

```
sh prepare_osm.sh \
  /irtx-jsprit/input/rhone-alpes-latest.osm.pbf \
  /irtx-jsprit/output/perimeter.poly \
  /irtx-jsprit/output/scenario.osm.pbf
```

Operator For the Lyon living lab, an operator file is available in `data` as `rexel_lyon.json`. This file does not exactly describe the actual demand of the operator. Respective data will be integrated in the final platform.

Additionally, the downstream parcel generation model and its respective connector model can be used to generate a parcel shipment operator (`laposte_*.json`) based on synthetic population data.

Scenario execution

For the Lyon living lab, a scenario template is provided in the LEAD repository as `data/template_lyon.json`. Different configurations based on this template are described further below. The resulting configured scenario files can then be run using the following command:

```
java -cp /irtx-jsprit/java/target/lead-jsprit-1.0.0.jar
fr.irtx.lead.jsprit.RunSolver \
  --problem-path /irtx-jsprit/output/scenario_{scenario}.json \
  --solution-path /irtx-jsprit/output/solution_{scenario}.json \
  --crs EPSG:2154 \
  --osm-path /irtx-jsprit/output/scenario.osm.pbf
```

Here, `{scenario}` should be replaced by `{scenario} = baseline_2022 | ucc_2022 | ucc_2030`. In each case, a different `/irtx-jsprit/output/scenario_{scenario}.json` can be generated using the configuration tool:

Baseline 2022

```
python3 prepare_scenario.py \
  --scenario-path data/template_lyon.json \
  --output-path /irtx-jsprit/output/scenario_baseline_2022.json \
  --operator-path data/rexel_lyon.json \
  --operator-path /irtx-parcels-jsprit-connector/output/laposte_2022.json \
  --shipment-type:rexel pickup \
  --shipment-type:laposte delivery \
  --consolidation-type:rexel none \
  --consolidation-type:laposte none \
  --driver-salary:rexel 0.0
```

The driver salary for Rexel is put to zero, because it is a pick-up service, which means that employees are employed independent of their pick-up tasks.

UCC 2022

In this scenario consolidation is integrated by forcing the deliveries to be routed through the UCC. The postal distribution center is relcoated to the UCC. This is done by changing the last four lines.

```
python3 prepare_scenario.py \
  --scenario-path data/template_lyon.json \
  --output-path /irtx-jsprit/output/scenario_ucc_2022.json \
```

```
  --operator-path data/rexel_lyon.json \
  --operator-path /irtx-parcels-jsprit-connector/output/laposte_2022.json \
  --shipment-type:rexel delivery \
  --shipment-type:laposte none \
  --consolidation-type:rexel delivery \
  --consolidation-type:laposte delivery
```

In this example, Rexel needs a driver, so the salary is kept at its default value.

UCC 2030

In this scenario the parcel demand for 2030 is used.

```
python3 prepare_scenario.py \
  --scenario-path data/template_lyon.json \
  --output-path /irtx-jsprit/output/scenario_ucc_2030.json \
  --operator-path data/rexel_lyon.json \
  --operator-path /irtx-parcels-jsprit-connector/output/laposte_2030.json \
  --shipment-type:rexel delivery \
  --shipment-type:laposte none \
  --consolidation-type:rexel delivery \
  --consolidation-type:laposte delivery
```

Other scenarios

Other scenarios can be constructed by changing parameters in the upstream models (total number of parcels, for instance), by replacing the operator data sets for other years or even adding new operators, or by changing the vehicle parameters.

In the platform a user could, for instance, a list of prepared operator data sets that can then be tested in combination using the model.

Route Optimisation Last Mile Team

Configuration Documentation for a LEAD Living Lab

Author(s): Angel Batalla

Author'(s') affiliation (Partner short name): LMT

Revision History

Version No.	Date	Details
1.0	13/02/22	First draft
2.0	28/04/22	Expanded output tables
3.0	30/10/22	Updated with refactored input, output and operational tables

Contents

List of tables

1 Introduction

1.1 Scope and objectives

Madrid Living Lab ambition is to demonstrate the better economic and environmental efficiencies in using an Urban Consolidation Centre (UCC) connected to the Trans-European Transport Network (TEN-T), to deliver goods to the city centre.

The UCC is located in an underground parking in the center of 'Madrid Distrito Centro' Special Protection Low Emission Zone represented in the map below. The area main characteristics are:

- 5 zip codes – 5,25 Km²
- 150.000 permanent residents
- ± 20.000 tourists/day

Madrid Living Lab addresses real freight movement problems deriving from potential traffic restrictions to be enforced, by simulating and demonstrating the use of the urban consolidation centre to deliver/pick up freight, using two and three-wheel electric cargo scooters and vans.

To achieve this, it uses a Digital Twin that integrates and orchestrates different specialized simulation models, whose relationships and workflow can be seen in figure 1.1.

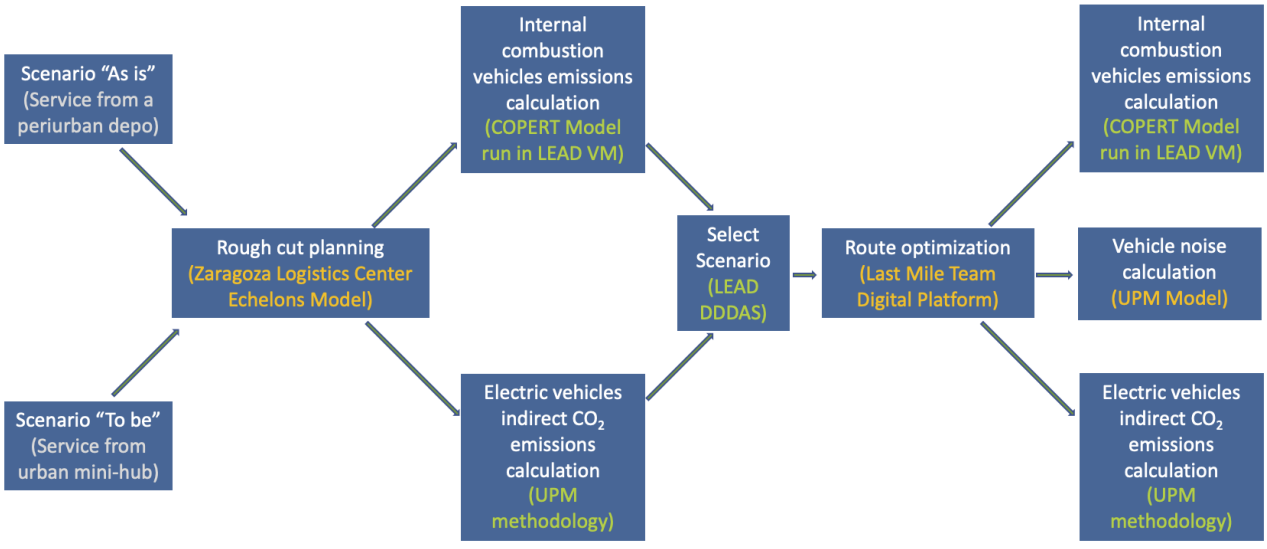


Figure 1.1 Madrid Living Lab Digital Twin models and workflow

The Route Optimisation model conducts the routing modelling and provides the outputs needed to assess the improvement of environmental indicators and the impact of alternative strategies for the clients in terms of cost and reliability, supporting the decision-making process.

2 Requirements.

2.1 Software requirements

The LMT Route Optimisation model is developed in C#, an object-oriented programming language created by Microsoft that runs on the .NET Framework. Is dependent on Windows operating system, Microsoft SQL, LMT Active Directory and Azure Active Directory.

Is executed on a Windows Azure Cloud environment, in the following Windows Server Virtual Machines: 2 Domain Controllers, 2 Web Servers, 1 Windows Services Server, 1 Build Server, 1 Jump Server. The database is Microsoft SQL, running in a Microsoft SQL Server.

2.2 Input/Outputs

2.2.1 Inputs

The Route Optimisation model is a proprietary model of LMT. It is an integral part of the Last Mile Digital Platform, a commercial asset used in LEAD as the underlying technology of Madrid Living Lab.

Receives as operational input only the Services file from the scenario selected by the user in the DDDAS.

Being an enterprise-certified commercial asset configuration is a critical process. It has to follow a specific order to populate the tables, establish relationships, create users, assign permissions, profiles, roles etc. to guarantee the expected levels of data integrity and operational security.

The configuration of the model is customer-specific. Currently there are twenty tables with over two hundred fields to configure a systemic Digital Twin, the aggregation of all individual Digital Twins of each and every one of customer owned, leased or managed physical assets used to perform its operations in the real world.

In real-life operations all individual physical assets configuration parameters are dynamically updated between optimization runs. The Route Optimization model uploads the snapshot of all individual physical asset’s configuration parameters at the time of execution, to ensure the best possible optimization to the specific services demand, balancing costs, environmental sustainability and driver working conditions and well-being.

Most of the configuration inputs for Madrid Living Lab scenarios is carried out by LMT, leaving the business partners to provide the minimal inputs specific to them as type of vehicles, working hours, etc.

LMT has completely redesigned for LEAD Madrid Living Lab a simplified, eight tables partial input configuration data model respecting the underlying database structure integrity, reducing the need of inputs to the minimum possible extent, following the once-only principle, to reduce administrative burden and enable fast scenario setting for Madrid Living Lab partners.

For the operational inputs the Living Lab technical partners (ZLC, LMT, UPM) and the business partners (CLOGIN, EMT) have co-created an anonymous-by-design single-table data collection model to capture the strictly minimum information of the physical activities necessary to feed the Route Optimization model.

The configuration inputs are described in Tables 1 to 8, the operational inputs in Table 9

Table 1 - Step04_CityHub - Configuration Inputs

Inputs	Description
CustomerLevelTwoId	Leave **BLANK**
CustomerLevelThreeId	Leave **BLANK**
Code	Assigned by LMT
Name	Assigned by LMT
Address	CityHub address
Number	CityHub address
City	CityHub city
ZipCodeId	CityHub zip code
Region	Assigned by LMT
Country	Assigned by LMT
Coordinates	Coordinates of the CityHub location, expressed as latitude and longitude separated by comma (,). Longitude preceded by a minus sign (-) if west, or without sign if east. Decimal separator is a dot (.) Coordinate system EPSG:4326 (WGS84).

LandLine	Assigned by LMT
Mobile	Leave **BLANK**
Email	Assigned by LMT
OpeningTime1	Assigned by LMT, to ensure City Hub opening times are always within drivers working shifts
ClosingTime1	Assigned by LMT, to ensure City Hub opening times are always within drivers working shifts
OpeningTime2	Leave **BLANK**
ClosingTime2	Leave **BLANK**

Table 2 - Step05_ServiceTypeCustomer – Configuration Inputs

Inputs	Description
Name	Assigned by LMT
ShortName	Assigned by LMT
ServiceTypeId	Assigned by LMT
VehicleTypeId	Assigned by LMT. If possible, we will use the vehicle-equivalent designation in COPERT DB, to enable vehicle emissions and energy consumption calculation.
WindowStart1	ServiceType time window start in hh:mm format
WindowEnd1	ServiceType time window end in hh:mm format
WindowStart2	Leave **BLANK**
WindowEnd2	Leave **BLANK**
Price	Leave **BLANK**
Requirement	Assigned by LMT
Capability	Assigned by LMT

Priority	Assigned by LMT
Slot Duration	Assigned by LMT
AvailableForHolidays	Assigned by LMT
Activity	Assigned by LMT

Table 3 - Step06_ZipCodeCityHub – Configuration Inputs

Inputs	Description
ZipCodeId	All zip codes served by the CityHub
CityHubId	Assigned by LMT
ServiceTypeCustomerId	Assigned by LMT
CountryId	Assigned by LMT

Table 4 - Step10_User – Configuration Inputs

Inputs	Description
Name	Assigned by LMT
Surname	Leave **BLANK**
Email	Leave **BLANK**
UserName	UNIQUE – assigned by LMT
Password	Assigned by LMT
Mobile	Leave **BLANK**
LandLine	Leave **BLANK**
CustomerLevelTwoId	Leave **BLANK**

CustomerLevelThreeId	Leave **BLANK**
CityHubId	Assigned by LMT
PudoId	Leave **BLANK**
Type	Leave **BLANK**

Table 5 - Step11_ServiceTime – Configuration Inputs

Inputs	Description
ServiceTypeCustomerId	Assigned by LMT
ZipCodeId	Assigned by LMT
Time	Average time for the ServiceType in hh:mm format
Province	Assigned by LMT

Table 6 - Step14_PlatformCityHub – Configuration Inputs

Inputs	Description
Platform	Assigned by LMT
ZipCodeCityHubId	Assigned by LMT
CityHubId	Assigned by LMT
Country	Assigned by LMT

Table 7 - Step17_Driver – Configuration Inputs

Inputs	Description
CityHubId	Assigned by LMT
Name	Assigned by LMT
Dni	Assigned by LMT
Email	Assigned by LMT
Mobile	Assigned by LMT
Imei	Assigned by LMT
StartHour	Working day start time in hh:mm format
EndHour	Working day end time in hh:mm format
Address	Assigned by LMT
Number	Assigned by LMT
City	Assigned by LMT
ZipCode	Assigned by LMT
Region	Assigned by LMT
Country	Assigned by LMT
Coordinates	Coordinates of the location where the driver working day starts, expressed as latitude and longitude separated by comma (,). Longitude preceded by a minus sign (-) if west, or without sign if east. Decimal separator is a dot (.) Coordinate system EPSG:4326 (WGS84). By default LMT assigns to every driver the coordinates of the CityHub
ActiveforPlanification	Assigned by LMT
ControlWorkday	Assigned by LMT

AverageWorkday	Assigned by LMT
UserId	Assigned by LMT

Table 8 - Step18_Vehicle – Configuration Inputs

Inputs	Description
NumberPlate	Assigned by LMT
VehicleTypeId	Assigned by LMT. If possible we will use the vehicle-equivalent designation in COPERT DB, to enable vehicle emissions and energy consumption calculation.
CityHubId	Assigned by LMT
EnvironmentalHallmark	Assigned by LMT
Capability	Assigned by LMT
ReturnDepartureLocation	Assigned by LMT
ArrivalCityHubId	Leave **BLANK**
ArrivalDriverId	Leave **BLANK**
	Leave **BLANK**
Capacity	Vehicle capacity in parcels, baskets, crates etc.
InitialUnits	Assigned by LMT
CapacityKg	Vehicle capacity in Kg.
InitialKg	Assigned by LMT
CapacityM3	Vehicle capacity in volume
InitialM3	Assigned by LMT
FixedCost	Daily cost

KilometerCost	Leave **BLANK**
HourCost	Leave **BLANK**
OvertimeCost	Leave **BLANK**
OvertimeUnits	Leave **BLANK**
LimitOvertimeBefore	Leave **BLANK**
LimitOvertimeAfter	Leave **BLANK**
MaxKilometersDay	Assigned by LMT
MaxContDrivingTime	Leave **BLANK**
Priority	Assigned by LMT
Template	Assigned by LMT
StartTime	Assigned by LMT
EndTime	Assigned by LMT
WorkBreakStart	Work break window start time in hh:mm format
WorkBreakEnd	Work break window end time in hh:mm format
WorkBreakDuration	Work break duration, in hh:mm format
Notes	Leave **BLANK**
ActiveForPlanification	Assigned by LMT
DriverId	Assigned by LMT

Table 9 - Service – Operational Inputs

Inputs	Description
WayBillCustomer	UNIQUE identifier for a service
RetailerCustomerId	Assigned by LMT. The same value for all services of all shippers/retailers
WayBillOriginator	Leave **BLANK**
RecipientName	Leave **BLANK or include a consecutive anonimous designator meaningful for you**, as "Delivery Point X" or "Customer X"
RecipientAddress	Insert actual address(only street name and house number) **if allowed by Data Protection**. If not, fill it with any character or string.
RecipientCity	City name
RecipientZipCode	Has to be one of the zip codes served by the City Hub
RecipientCountry	Receiver country code in ISO format 3166-1-alpha-2
RecipientMobile	Leave **BLANK**
RecipientLandLine	Leave **BLANK**
RecipientEmail	Leave **BLANK**
ServiceType	Name of the service/s configured by LMT
Coordinates	Coordinates of the location to drop off or pick up parcels, expressed as latitude and longitude separated by comma (,). Longitude preceded by a minus sign (-) if west, or without sign if east. Decimal separator is a dot (.) Coordinate system EPSG:4326 (WGS84).
AmountCashOnDelivery	Leave **BLANK**
AmountShippingCharge	Leave **BLANK**
Notes	Leave **BLANK**
InjectionZipCode	Assigned by LMT. Has to be one of the zip codes served by the City Hub.

ServiceTime	Leave **BLANK, except for a justified cause.** If left **BLANK** the optimization engine will take for all services the ServiceType ServiceTime default of all the zip codes served by the CityHub, **OR** the one of the ServiceType ServiceTime of the specific zip code if these were individually configured. **If NOT BLANK** the optimization engine will take the ServiceTime specified for the individual service, overriding the above.
LoadUnits	Units to Pick up (collect) at location. **If none, ensure UnloadUnits is > 0**
UnloadUnits	Units to Drop off (deliver) to location. **If none, ensure LoadUnits is > 0**

2.2.2 Outputs

Main model outputs are:

- Number and average distance driven per internal combustion engine vehicle type. This is the input to COPERT, the EU standard vehicle emissions and energy consumption calculator.
- Number and average distance driven per electric vehicle type. This is the input to EVCO2, a methodology adapted by UPM from Red Electrica Española, the Spanish power grid operator, to calculate the CO_2 equivalent emissions of the energy consumed by the electric vehicles operation.
- Step-by-step routes geospatial data per vehicle. Input to the noise calculation module.

The main Route Optimization model outputs for LEAD Living Lab are:

- Step-by-step routes per vehicle. Available to the logistics partner (CLOGIN), for them to use as the input to estimate some of the operational and economic KPIs.

- lmt_LEAD_input_to_COPERT.json. This is a file with the number of internal combustion engine vehicles per COPERT vehicle type and the average distance driven per vehicle type. These data elements are inserted in a specific COPERT-proprietary Microsoft Excel file that is sent to LEAD COPERT server in Amazon Web Services using and ad-hoc Python Jupyter notebook. The server calculates vehicle emissions and energy consumption, that are downloaded in LMT environment and sent to LEAD platform.

- lmt_LEAD_input_to_EVCO2.json. This is a file with the number of electric vehicles per type and the average distance driven. These data elements are inserted in specific working files that, using and ad-hoc Python Jupyter notebook, calculates in LMT environment the CO_2 equivalent emissions of the energy consumed by the electric vehicles operation, and sends the results to LEAD platform.

- lmt_LEAD_input_to_noise.json. This is a step-by-step routes GeoSpatial data set per vehicle, to be consumed by UPM as the input to their noise calculation module.

CAUTION: when there is more than one service in the exact same location, the GeoSpatial information of the route leg from the preceding location is only captured in the first service, having the other/s the value “LINESTRING EMPTY”. This dataset excludes the “LINESTRING EMPTY” data elements of the second, third, etc. services, to avoid the need of post-processing the data set to upload it in a GIS.

LMT Route Optimization operational outputs that could potentially be publicly shared are described in Tables 10 to 14.

Table 10 - ResponsePendingService – Operational Outputs

This table is blank most of the time. It captures the details of services that the optimization engine left unserved because they violate any of the set time, journey, cost or other constraints.

Output	Description
Id	UNIQUE, created when the record is inserted in the DB
ResponsePlanId	UNIQUE, created when the ResponsePlan record is inserted in the DB
ServiceId	UNIQUE, created when services are inserted in the DB, before creating the Request to the optimization engine.
Name	Always BLANK, to abide to GDPR requirements
Duration	Time in minutes assigned to perform the service
LocationCoordinates	Coordinates of the location to drop off or pick up parcels, expressed as latitude and longitude separated by comma (,). Longitude preceded by a minus sign (-) if west, or without sign if east. Decimal separator is a dot (.) Coordinate system EPSG:4326 (WGS84).
Priority	Always set to 1. The optimization engine will treat all services equally, will try to serve them all.
WindowStart	ServiceType time window start
WindowEnd	ServiceType time window end
Requirement	Hard constraint associated to the ServiceType or location that must be met (need a specific vehicle type, a driver with certain skills, can only be accessed at certain time windows, etc.
UnloadUnits	Units to Drop off (deliver) to location (parcels, crates, pallets)
LoadUnits	Units to Pick up (collect) from location (parcels, crates, pallets)
LoadKg	Weight to Drop off (deliver) to location
UnloadKg	Weight to Pick up (collect) from to location
LoadM3	Volume to Drop off (deliver) to location
UnloadM3	Volume to Pick up (collect) from location

Comments	Field to communicate special instructios or other details of the service to the driver. Not used, always blank.
RegisterDate	The date corresponds with the services date. Discard the time, LEAD optimizations are asynchronous and the stated time is irrelevant
DeletionDate	Always null

Table 11 - ResponsePlan - Operational Outputs

This table provides a summary of the main parameters and outputs of the Response to an optimization Request.

Output	Description
Id	UNIQUE, created when the record is inserted in the DB
RequestId	UNIQUE, created in the DB before sending the Request to the optimization engine
JobId	UNIQUE, created in the optimization engine when receives a valid Request. Value is inserted in the DB for reference
CityHubId	UNIQUE
ExecutionTime	Total elapsed time from uploading the services file to the FTP folder to the insertion of all the data contained in the Response
Iterations	Number of iterations allowed to the optimization engine
NumberOfServices	Total number of services sent to the optimization engine
ServicesNotProvided	Services that the optimization engine left unserved because they violate any of the set time, journey, cost or other constraints
NumberofVehicles	Number of vehicles available at the CityHub
VehiclesInUse	Number of vehicles used for the specific optimization
TotalDrivingTime	
TotalServiceTime	
TotalBreakTime	

TotalNotWorkTime	Provides an indication of the vehicle/delivery capacity available at the CityHub versus the vehicle/delivery capacity utilized. Disregard this field. It is not used in LEAD.
TotalOvertimeWork	Always 0 in LEAD. Overtime is not allowed
AverageWorkingTime	Of the vehicles used for the specific optimization
AverageWorkVehicle	Provides an indication of the average vehicle/delivery capacity utilized versus the available. Disregard this field. It is not used in LEAD.
OnTimeServices	
TotalKm	
TotalCosts	
TollCosts	Always 0 in LEAD
ServicesOutOfWindow	LEAD services time window is from 09:00 to 17:30. Always 0 because overtime is a not allowed, hard constraint.
TimeOutOfWindow	Same as above
RegisterDate	The date corresponds with the services date. Discard the time, LEAD optimizations are asynchronous and the stated time is irrelevant
DeletionDate	Always null
IsOverLimitHours	Value 1 if the optimization does not violate regular working hours. In LEAD always zero because overtime is a not allowed, hard constraint.

Table 12 - Response Service - Operational Outputs

This table provides data associated to each individual service of an optimization Request.

Output	Description
Id	UNIQUE, created when the record is inserted in the DB
ResponseVehiclePlanId	UNIQUE, created when each ResponseVehiclePlan record is inserted in the DB
Start	Service start time

End	Service end time
ServiceId	UNIQUE, created when services are inserted in the DB, before creating the Request to the optimization engine.
Name	Always BLANK, to abide to GDPR requirements
Duration	Time in minutes assigned to perform the service
LocationCoordinates	Coordinates of the location to drop off or pick up parcels, expressed as latitude and longitude separated by comma (,). Longitude preceded by a minus sign (-) if west, or without sign if east. Decimal separator is a dot (.) Coordinate system EPSG:4326 (WGS84).
Priority	Always set to 1. The optimization engine will treat all services equally, will try to serve them all.
WindowStart	ServiceType time window start
WindowEnd	ServiceType time window end
Requirement	Hard constraint associated to the ServiceType or location that must be met (need a specific vehicle type, a driver with certain skills, can only be accessed at certain time windows, etc.
UnloadUnits	Units to Drop off (deliver) to location
LoadUnits	Units to Pick up (collect) from location
LoadKg	Weight to Pick up (collect) from location
UnloadKg	Weight to Drop off (deliver) to location
LoadM3	Volume to Pick up (collect) from location
UnloadM3	Volume to Drop off (deliver) to location
Comments	Field to communicate special instructions or other details of the service to the driver. Not used, always blank.
RegisterDate	The date corresponds with the services date. Discard the time, LEAD optimizations are asynchronous and the stated time is irrelevant
TravelDistance	Driven distance from the former location

TravelTime	Driving time from the former location
DepartureDate	Date and time when the vehicle start driving to the next location

Table 13 - ResponseServicePath - Operational Outputs

This table provides GeoSpatial data of all legs of a specific route of a specific vehicle of a specific optimization Request.

Output	Description
Id	UNIQUE, created when the record is inserted in the DB
Time	Driving time from the former location
Distance	Distance driven from the former location
Geometry	**CAUTION**: LMT backend underlying DB is Microsoft SQL. All records in this json file have a LINESTRING instance as a Geometry variable. When there is more than one service in the exact same location, the GeoSpatial information of the route leg from the preceding location is only captured in the first service, having the other/s the value “LINESTRING EMPTY” If you plan to load this data into a GIS consider use the geojson version to avoid having to process this json file, if fits your purpose.
Order	Is the sequence of the service stop in the route
ResponseVehiclePlanId	UNIQUE, created when each ResponseVehiclePlan record is inserted in the DB
RegisterDate	The date corresponds with the services date. Discard the time, LEAD optimizations are asynchronous and the stated time is irrelevant
DeletionDate	Always null

Table 14 - ResponseVehiclePlan - Operational Outputs

This table provides the summary per vehicle of the routes of an optimization Request.

Output	Description
Id	UNIQUE, created when the record is inserted in the DB
ResponsePlanId	UNIQUE, created when the ResponsePlan record is inserted in the DB
TotalDrivingTime	
TotalServiceTime	
TotalBreakTime	
TotalOvertimeWork	
TotalKm	
TotalCosts	
TollCosts	
ServicesOutOfWindow	LEAD services time window is from 09:00 to 17:30. Always 0 because overtime is a not allowed, hard constraint.
TotalServices	Number of services by the vehicle
VehicleId	UNIQUE, assigned by LMT during configuration
Name	UNIQUE, assigned by LMT during configuration
StartTimeWorkday	
EndTimeWorkday	
CostKm	Not used in LEAD. Always 0
CostHour	Not used in LEAD. Always 0
LocationCoordinates	Coordinates of the location to drop off or pick up parcels, expressed as latitude and longitude separated by comma (,). Longitude preceded by a minus sign (-) if west, or without sign if east.

	Decimal separator is a dot (.) Coordinate system EPSG:4326 (WGS84).
WorkBreakStart	Work break window start time in hh:mm format
WorkBreakEnd	Work break window end time in hh:mm format
WorkBreakDuration	Work break duration, in hh:mm format
Barcode	UNIQUE for each vehicle. Not used in LEAD
RegisterDate	The date corresponds with the services date. Discard the time, LEAD optimizations are asynchronous and the stated time is irrelevant
DepartureDate	Date and time when the vehicle start the journey
ArrivalDate	Date and time when the vehicle end the journey
TravelDistance	Driven distance from the last route stop to the CityHub
TravelTime	Driving time from the last route stop to the CityHub
InitialBattery	Not used in LEAD. It does not appear in the file.
FinalBattery	Not used in LEAD. It does not appear in the file.
InitialKilometers	Not used in LEAD. It does not appear in the file.
FinalKilometers	Not used in LEAD. It does not appear in the file.

3 Model Description

At the core of the LMT Route Optimisation model are different road transport specific Artificial Intelligence algorithms plus a comprehensive, in-house developed, IP registered, technology stack.

The core enables to optimize urban logistics networks in cities, urban and peri-urban areas considering the road/streets existing network plus all available resource: people, their working hours, customer delivery windows, municipal, regional, national and European rules and legislation in diverse road-transport related regulated fields as vehicle types, low emission zones, other geo-fenced vehicle, time or otherwise restricted areas, specific transport regulations etc. to ensure the best possible optimization to any specific need not only from an economic point of view but also from an environmental sustainability and driver working conditions and well-being points of view.

4 Instructions to run the model

The Route Optimization model environment, libraries, system dependencies etc. are kept up to date by LMT. Can’t be executed through the command line by the user, is only run by LMT.

Discrete Choice Model V1.0

Documentation

Author(s): Carla de Oliveira Leite Nascimento, Edoardo Marcucci, Valerio Gatta

Author’(s’) affiliation (Partner short name): MOLDE

Revision History

Version No.	Date	Details
1.0	25/07/22	

Contents

List of tables

1 Introduction

1.1 Scope and objectives

Analytical tool to model agents' preferences, assess adoption/acceptance/reaction, optimise stimuli selection and identify compromise solutions. Discrete choice models (DCM) are econometric models aimed at analysing the behaviour of a decision-maker when choosing among different (discrete) options. This model can be used to investigate stakeholders' preference heterogeneity to forecast their choice behaviour related to some scenario, policy-making or decision.

Once the consumer preferences have been collected (through a well-designed stated preference model), then this information will be processed and organized in a structured way to support econometric analysis and experimentation (simulation). The DCM use random utility theory to model SP respondents' decision-making and is a valuable instrument in addressing, from a theoretically well-grounded perspective, stakeholders' heterogeneous preferences concerning alternative delivery configurations.

Key assumptions:

- The user has already collected the data to be analysed through the DCM model
- The data was collected via a Stated Preference (SP) questionnaire

2 Requirements.

2.1 Software requirements

The simulators have been built using RStudio version 1.4
The following R libraries need to be installed:

1. library(mlogit)
2. library(stargazer)
3. library(readxl)

2.2 Input/Outputs

2.2.1 Inputs

As shown in Table 1, every column describes respectively respondents' answers to the SP experiment, attributes levels and alternatives. Even in this case, columns order must be respected, and you can add new attributes by inserting new columns.

Table 1 - Example of input for the DCM model

Answer	Attribute 1	Attribute 2	Attribute 3	Attribute 4
	Price of Delivery	Delivery time	Date of the delivery	CO_2 emission
1	1	High	100	High
0	0	High	200	High
1	0	Low	300	Low
0	1	High	100	High
1	1	High	200	High
0	1	Low	200	Low
0	0	High	100	High
1	0	Low	300	Low
0	1	High	200	High
1	0	Low	100	Low

Table 2 below contain the description of the file.

Table 2 DCM – Inputs

Inputs	Description
answers.xlm	Excel file containing all answers and collected data

2.2.2 Outputs

Table 3 shows an example of an output obtained from the DCM model. The R code provides a csv file that can be easily transformed into an Excel file.

With these results, it is possible to formulate a utility function (equation 1). V_{in} is the systematic utility expressed as a function of observable variables and ε_{in} is the random utility component.

Based on idea that an individual n selects the alternative i with the highest utility U_{in} among those in the choice set C_n. Specifically, the goal of this choice model is to estimate the significance of the determinants of V_{in}.

$$U_{in} = V_{in} + \varepsilon_{in} \qquad \text{Equation 1}$$

The interpretation of the estimator depends on the nature of the attributes.

Table 3 - Example of input for de DCM model

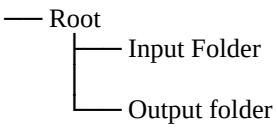
	Estimate	Std. Error	z-value	Pr(>|z|)
(intercept):2	-83,0993347	10527,133394	-0,007893823	0,993701706
Attribute 1	-49,55247018	7122,786155	-0,006956894	0,994449246
Attribute 2	66,75697207	8560,27969	0,007798457	0,003777795
Attribute 3	0,164785195	26,89874632	0,006126092	0,996112116
Attribute 4	0,852621457	56,25412876	0,008524731	0,001243387

Table 4 DCM – Outputs

Outputs	Description
dcm.csv	Attributes list with the estimation of the coefficients

2.3 Paths' structure

The directory where the model is located has the following structure:



3 Model Description

This section describes the different files and scripts present in the model

File name	Location	Description
DCM.R	Root	Main script
requirements.txt	Root	R packages required

4 Instructions to run the model

4.1 Environment preparation

4.1.1 Environment

4.1.2 Use of the code

Once created the file, r code can be easily run typing “name_of_the_file.xlsx”, namely the file path of the excel file, in the function called model.

```
z<-model(dataset="name_of_the_file2.xlsx")
```

4.2 Command line execution of the model

4.2.1 Execution command

4.2.2 Execution example

An example for the implementation

Experimental Design V1.0

Documentation

Author(s): Carla de Oliveira Leite Nascimento, Edoardo Marcucci, Valerio Gatta

Author’(s’) affiliation (Partner short name): MOLDE

Revision History

Version No.	Date	Details
1.0	25/07/22	

Contents

List of tables

1 Introduction

1.1 Scope and objectives

This model is an analytical tool that will supply the building blocks to design the choice experiments. A choice experiment is a survey approach designed to draw consumer preferences based on a hypothetical scenario. Thus, the respondents are required to choose between multiple options. This choice is expected by the researcher to occur by the trade-off of the individual attributes of the different goods available and choosing the good (or alternative) that provides the most utility.

The results will help in forecasting decisions, suggesting to respondents' questions about their possible choices in hypothetical situations given a specific set of conditions created thanks to experimental design. The level of distinctiveness of the alternatives is nothing more than the representation of goods or services that differ from each other. Alternatives are offered to individuals, and they are asked to express their choices by declaring their preference.

The code exemplified here allows to create an experimental design matrix, for then to be able build the blocks used in a stated preference data collection.

2 Requirements.

2.1 Software requirements

The simulators have been built using Rstudio version 1.4

The following R libraries need to be installed:

1. library(readxl)
2. library(choiceDes)

2.2 Input/Outputs

2.2.1 Inputs

An excel file with the following columns: attributes columns, choice sets, alternatives and blocks. The experiment can have as many attributes as is necessary, but the order of the columns should be the same as is shown in Table 1.

Table 1 Example of the excel input for the experimental design

Attribute 1	Attribute 2	Attribute 3	Attribute 4	Choice sets	Alts	Blocks
Price of Delivery	Delivery time	Date of the delivery	CO2 emission			

10	1	Yes	10	10	2	4
20	2	No	100			
30	3		1000			
40	4		10000			
	5					

Table 2 below contain the description of the file.

Table 2 Experimental design– Inputs

Inputs	Description
attributes.xml	Excel file containing the attributes to be tested

2.2.2 Outputs

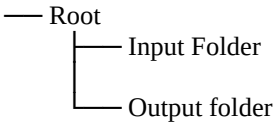
As an output for this model, the code will provide an csv file with the attribute's levels, structured via orthogonal coding

Table 3 – Experimental design output example

Version/Block	Task	Alternative	Attribute 1	Attribute 2	Attribute 3	Attribute 4
			Price of Delivery	Delivery time	Date of the delivery	CO2 emission
1	1	1	1	-2	2	-3
1	1	2	1	0	-2	-1
1	2	1	-3	2	1	3
1	2	2	3	-1	0	1
1	3	1	3	1	-1	-3
1	3	2	-1	-2	0	-3
1	4	1	3	-1	2	3
1	4	2	-3	-1	-1	3

2.3 Paths' structure

The directory where the model is located has the following structure:



3 Model Description

This section describes the different files and scripts present in the model

Table 4

File name	Location	Description
ED.R	Root	Main script
requirements.txt	Root	R packages required

4 Instructions to run the model

4.1 Environment preparation

4.1.1 Environment

4.1.2 Use of the code

Once created the file, r code can be easily run typing “name_of_the_file.xlsx”, namely the file path of the excel file, in the function called design:

```
z<-design(dataset="name_of_the_file1.xlsx")
```

4.2 Command line execution of the model

4.2.1 Execution command

Parcel Market V1.0

Documentation

Author(s): Rodrigo Tapia

Author’(s’) affiliation (Partner short name): TU DELFT

Revision History

Version No.	Date	Details
1.0	19/01/22	

Contents

List of tables

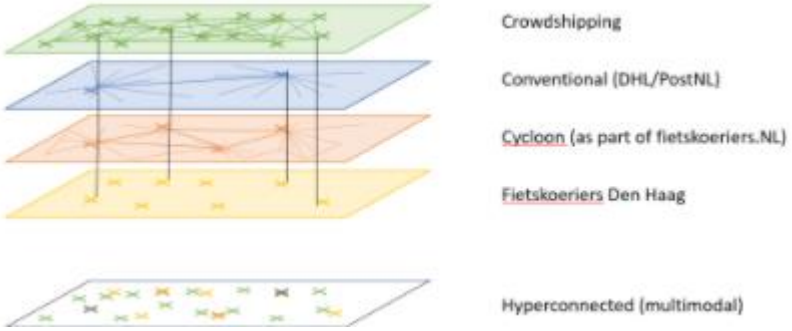
1 Introduction

1.1 Scope and objectives

The parcel market module is separated into 2 parts. The first one generates the networks and, if selected, connects couriers' networks or adds crowdshipping links between particular O-Ds.

1.1.1.1 Network Generation

This section generates the different networks for different delivery methods, including crowdshipping. Each of the networks is a separate entity generated with the package NetworkX and contains information regarding the distance links and nodes. For the final delivery (i.e. the node that connects the network to the delivery zone) a fictional node is generated that can different delivery costs (such as drop-off times) can be included. For couriers, additional nodes are generated in the hubs, according to the location in the shapefile. New hubs or Xdocking places can be generated by creating a fictional node in the hub list.



To connect networks, it is as simple as to generate nodes where two or more networks coincide. This can be done if a particular parcel can be crowdshipped, or if couriers share networks. It is worth noting that one and only one link connects two nodes. Once these links are created, we have created a hyperconnected network.

1.1.1.2 Service Network fulfilment

Once the networks are created, the parcels_hyperconnected go through the network using the dijkstra algorithm using a weighted path. The weights of each link is estimated with a score function that can estimate the cost depending on: Distance, time, transhipments, type of connection (delivery or between hubs). As a result, a path (sequence of nodes) for each parcel is determined.

With the paths estimated, we proceed to decompose it into individual trips between nodes. The parcels eligible for crowdshipping are sent the crowdshipping module. For the conventional trips (legs of traditional network + unmatched crowdshipping), a pickup trip is generated by connecting the origin from the consumer to the first depot in the chain. For the crowdshipping parcels, a matching algorithm that gives "the best parcel to the best trip" is used. This algorithm simulates the utility of all travellers to pick up all parcels and iteratively allocates the combination parcel-traveller that has the highest utility.

Finally, the individual trips are combined with the ones generated in the parcel generation module that did not go through the hyperconnected network (i.e. the parcels that are not L2L nor crowdshipped). The new list of parcels (pickup and delivery) is going to be fed to the schedule delivery.

Key assumptions:

- Conventional courier
  - All parcels are delivered and picked up from their closest depot
  - Hub/spoke structure
  - Zones are only connected to the closest depot of each courier
- Crowdshipping
  - Maximum utility framework assumptions
- Combination of logistic providers
  - Combinations are possible via transhipments points

2 Requirements.

2.1 Software requirements

The simulators have been built using Python version 3.8.8.
The following Python libraries need to be installed:

- networkx==2.6.3
- numpy==1.23.3
- scipy==1.9.1
- pandas==1.3.4
- pyshp==2.1.3
- python-dotenv==0.19.2

2.2 Input/Outputs

2.2.1 Inputs

The inputs of the parcel generation module are described in Table 1.

Table 1 Parcel Market– Inputs

Inputs	Description
Parameters.txt	Text file containing the parameters of the model

DEMANDPARCELS.csv	Parcel demand with the fulfilment option, CEP, L2L segment, Crowdshipping eligibility and whether they go to a Parcel Locker or not.
TimeSkimMatrix.mtx	Matrix that contains the time (in seconds) between each pair OD
DistanceSkimMatrix.mtx	Matrix that contains the distance (in kilometres) between each pair OD
StudyAreaShapefile	Shapefile of the city. The areas delimited should be linked with the areas in the Socioeconomic Data file.
SocioeconomicData	CSV with socioeconomic data per area within the city. The mandatory fields are “zone”; "1: woningen"; "9: arbeidspl_totaal"
ParcelNodes	Shape with the location of the distribution nodes of the couriers
Trips.csv	Passenger trips with origin, destination, mode and socioeconomic characteristics.

2.2.2 Outputs

The outputs of the parcel generation module are described in Table 2.

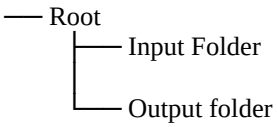
Table 2 Parcel Market – Outputs

Outputs	Description
ParcelDemand_HS_delivery	Parcel list with Origin, Destination and Courier
KPIs.json	File containing key KPIs estimated in the model
Log.txt	Logs of the run
ParcelDemand_HS_pickup	List of parcels to be picked up with Origin and Destination
ParcelDemand_ParcelHubSpoke	List of parcels with Origin and Destination that have Hub-Spoke delivery configuration
ParcelDemand_ParcelTrips	Local to Local parcel trips
Parcels_CS	List of parcels that considered using crowdshipping
Parcels_CS_matched	List of parcels that were delivered through crowdshipping

TripsCS	Passenger trips that deliver parcels with the parcel ID

2.3 Paths structure

The directory where the model is located has the following structure:



3 Model Description

This section describes the different files and scripts present in the model

File name	Location	Description
Parcel_Market.py	Root	Main script
LEAD_CS.py	Root	Crowdshipping module
__functions__.py	Root	External functions
requirements.txt	Root	Python packages required

4 Instructions to run the model

4.1 Command line execution of the model

4.1.1 Instructions and commands

The instruction to install the packages needed:

- pip install -r requirements.txt

The instruction to run the model

Parcel_Market.py Label Input Output Params_ParcelMarket.txt DEMANDPARCELS.csv TimeSkimMatrix.mtx DistanceSkimMatrix.mtx StudyAreaShapefile.shp SocioeconomicData.csv ParcelNodes.shp TRIPS.csv

4.1.2 Arguments

The arguments in the instructions to run the model are:

Table 3 Parcel Market – Input arguments

Arg[0]	Script name
Arg[1]	Lable (name of scenario)
Arg[2]	Input folder name
Arg[3]	Output folder name
Arg[4]	Parameter text file
Arg[5]	Parcel Demand
Arg[6]	Time skim matrx
Arg[7]	Distance skim matrix
Arg[8]	Zone shapefile
Arg[9]	Socioec data
Arg[10]	Parcel Nodes shapefile
Arg[11]	Passenger trips

4.2 Requirements

4.2.1 Testing requirements

pip install -r requirements.txt

4.2.2 Input folder (Arg[2])

Folder 1(e.g. Input)

Which files are in the folder, what do they do, what type of file (csv, etc) what requirements do they have (table with headers, what each row represent, etc)

Table 4 Parcel Market – Inputs

Inputs	Type	Requirements
Parameters.txt	.txt	
DEMANDPARCELS.csv	.csv	Required cols: “Parcel_ID“

		“O_zone“ “D_zone“ “DepotNumber“ “CEP“ “PL“ “L2L“ “Fulfilment” “CS_eligible“
TimeSkimMatrix.mtx	.mtx	Id of areas ordered increasingly
DistanceSkimMatrix.mtx	.mtx	Id of areas ordered increasingly
StudyAreaShapefile	.shp	.dbf with the same name
SocioeconomicData	.csv	Required cols: “zone”; "1: woningen"; "9: arbeidspl_totaal"
ParcelNodes	.shp	.dbf with the same name
Trips	.csv	Required cols: "areaOrigin" "areaDest" "age" "gender" "income" "work" "Mode"

Parcel Demand Generation V1.0

Documentation

Author(s): Rodrigo Tapia, Ioanna Kourounioti

Author’(s’) affiliation (Partner short name): TU DELFT

Revision History

Version No.	Date	Details
1.0	19/01/22	

Contents

List of tables

1 Introduction

1.1 Scope and objectives

The parcel demand module estimates the demand for B2C and B2B parcels using household and employment populations on the one hand, and the networks of parcel and express carriers (CEP) in the study area on the other hand. It considers only one parcel type and takes into account general demand parameters for the B2C and B2B market segments. Figure 1 shows the simulation procedure.

The module first calculates the B2B parcel demand by multiplying zonal employment and a parameter for the daily number of parcels per employee. The latter is derived from the statistics of the total B2B parcel market size which is available in the official Market Monitor data[1] and publicly available sociodemographic data.

Next, the module calculates zonal B2C parcel demand. It is fair to assume that the frequency with which a person receives parcels greatly differs between individuals and is among other factors dependent on their age, income, location of residence and personal preferences. To capture this, we developed an ordered logit model and estimated it on data from the MPN (Mobility Panel Netherlands). The model explains the online shopping frequency of a person as a function of their personal and household characteristics (age, household income, urbanization level at their location of residence). As the household composition differs between zones in the study area, the obtained parcel demand also differs between zones, in absolute numbers as well as in terms of the number of parcels per person per day.

To correct for the year of observations in the MPN and the base year of MASS-GT, the B2C demand is then calibrated to match the observed parcel market size from the official Market Monitor data for the base year of MASS-GT. This procedure preserves the differences in demand between zones while ensuring that the total demand is accurate.

[1] https://www.acm.nl/nl/publicaties/post-en-pakkettenmonitor-2019

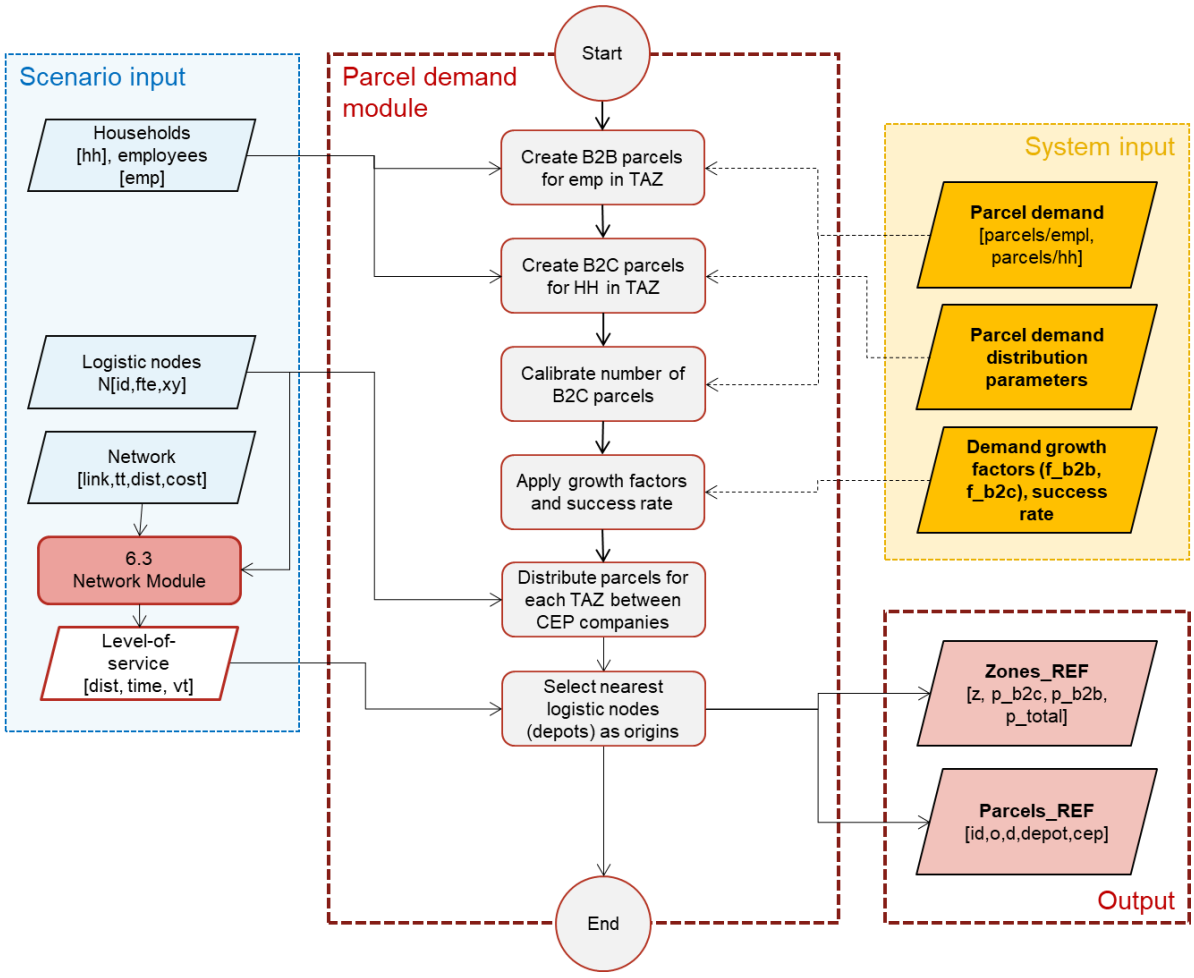


Figure 1: The parcel demand module structure

After this step, growth factors and delivery success rates can be applied. The delivery success rate (B2C and B2B separately) is a number between 0 and 1 and reflects the share of parcels for which the delivery attempt fails. Failed deliveries need to be repeated and thus increase the number of tours. The growth factors (again B2C and B2B) are applied to the market size and can be used to test future growth assumptions. Both can be set to 1 in the control file if no effect is desired.

Next, parcel demand (for both segments combined) is allocated to parcel and express companies (CEP). The CEP's observed market shares in the domestic parcel market are applied here. This data is available from open market data publications.

In the final step, the parcels are assigned to a depot from the corresponding CEP. It is assumed that each CEP has optimized its operations to deliver each parcel from the nearest depot. The output is a set of parcels with an origin (a depot) and a destination (a zone in which households and/or businesses are located).

Key assumptions:

- C2C parcels destination & origin is based on households distribution
- B2C parcels destination based on households distribution, origin on retail jobs

- Certain percentage of parcel being local to local. (The percentage is taken from the parcelmonitor)

2 Requirements.

2.1 Software requirements

The simulators have been built using Python version 3.8.8.
The following Python libraries need to be installed:

1. networkx==2.6.3
2. pandas==1.3.4
3. pyshp==2.1.3
4. tk==0.1.0
5. Built-in modules: itertools, math, sys, os

2.2 Input/Outputs

2.2.1 Inputs

The inputs of the parcel generation module are described in Table 1.

Table 1 Parcel Generation– Inputs

Inputs	Description
Parameters.txt	Text file containing the paramenters of the model
TimeSkimMatrix.mtx	Matrix that contains the time (in seconds) between each pair OD
DistanceSkimMatrix.mtx	Matrix that contains the distance (in kilometres) between each pair OD
tudyAreaShapefile	Shapefile of the city. The areas delimited should be linked with the areas in the Socioeconomic Data file.
SocioeconomicData	CSV with socioeconomic data per area within the city. The mandatory fields are “zone”; "1: woningen"; "9: arbeidspl_totaal"

ParcelNodes	Shape with the location of the distribution nodes of the couriers
CourierMarketShares	Couriers table with market shares
ExternalZones	Coordinates of the external (super) zones that lie outside of the study area.

2.2.2 Outputs

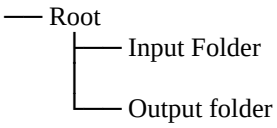
The outputs of the parcel generation module are described in Table 2.

Table 2 Parcel Generation– Outputs

Outputs	Description
ParcelDemand.csv	Parcel list with Origin, Destination and Courier
KPIs.json	File containing key KPIs estimated in the model
Log.txt	Logs of the run

2.3 Paths structure

The directory where the model is located has the following structure:



3 Model Description

This section describes the different files and scripts present in the model

File name	Location	Description
Parcel_Generation.py	Root	Main script
__functions__.py	Root	External functions
requirements.txt	Root	Python packages required

4 Instructions to run the model

4.1 Command line execution of the model

4.1.1 Instructions and commands

The instruction to install the packages needed:

- pip install -r requirements.txt

The instruction to run the model

python3 Parcel_Generation.py Label Input Output Params_ParcelGen.txt TimeSkimMatrix.mtx DistanceSkimMatrix.mtx StudyAreaShapefile.shp SocioeconomicData.csv ParcelNodes.shp CourierMarketShares.csv

4.1.2 Arguments

The arguments in the instructions to run the model are:

Table 3 Parcel Generation– Inputs

Arg[0]	Script name
Arg[1]	Lable (name of scenario)
Arg[2]	Input folder name
Arg[3]	Output folder name
Arg[4]	Parameter text file
Arg[5]	Time skim matrx
Arg[6]	Distance skim matrix
Arg[7]	Zone shapefile
Arg[8]	Socioec data
Arg[9]	Parcel Nodes shapefile
Arg[10]	Courier shares

4.2 Requirements

4.2.1 Testing requirements

pip install -r requirements.txt

4.2.2 Input folder (Arg[2])

Folder 1(e.g. Input)

Which files are in the folder, what do they do, what type of file (csv, etc) what requirements do they have (table with headers, what each row represent, etC)

Table 4 Parcel Generation– Inputs

Inputs	Type	Requirements
Parameters.txt	.txt	
TimeSkimMatrix.mtx	.mtm	Id of areas ordered increasingly
DistanceSkimMatrix.mtx	.mtm	Id of areas ordered increasingly
StudyAreaShapefile	.shp	.dbf with the same name
SocioeconomicData	.csv	Required cols: “zone”; "1: woningen"; "9: arbeidspl_totaal"
ParcelNodes	.shp	.dbf with the same name
CourierMarketShares	.csv	Required cols: “CEP”; "ShareTotal";
SupCoordinatesID	.csv	Required cols: “COROP”; " Xcoor "; “Ycoor”; " AREANR ";

Parcel Tour Formation V1.0

Documentation

Author(s): Rodrigo Tapia

Author’(s’) affiliation (Partner short name): TU DELFT

Revision History

Version No.	Date	Details
1.0	19/01/22	

Contents

List of tables

1 Introduction

1.1 Scope and objectives

The parcel scheduling module considers the parcel demand and consolidates them to create delivery tours. The parcel scheduling module uses as input the parcels predicted by the parcel synthesizer. It then simulates the formation of delivery tours for the distribution of these parcels. (See Figure 1) .

First, a list of parcels that are to be delivered in a specific zone is created and each list is assigned to a specific depot based on the proximity to the destination. Then for each depot, a pool of parcels is created.

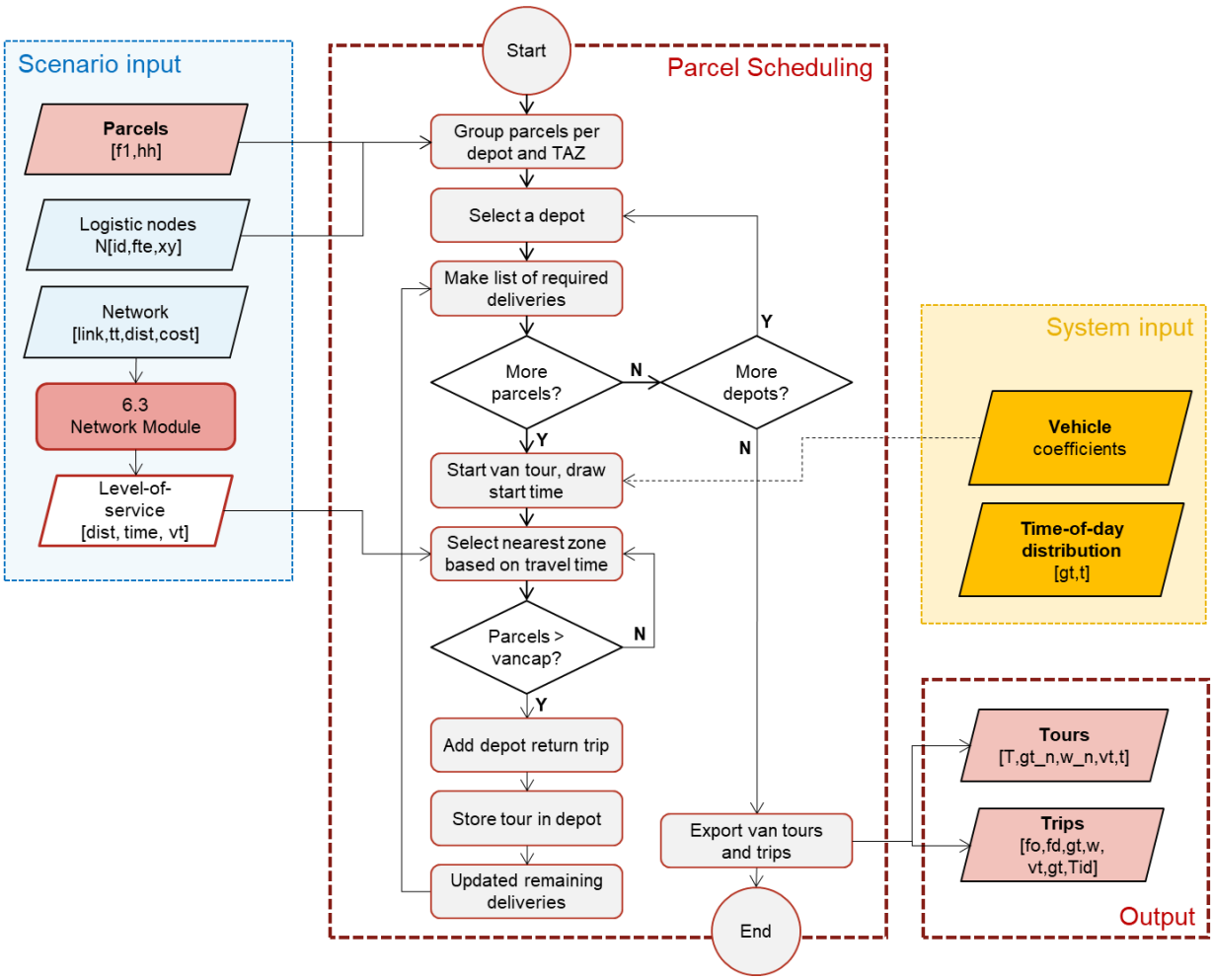


Figure 1 The parcel scheduling module

Based on the delivery location more parcels are added to the tour. The maximum number of parcels in a tour is assumed to be equal to the maximum capacity of the vehicle. For this we assumed 180 parcels. Once all parcels from the pool are allocated to delivery tours, the start time of the tour is

drawn from a time-of-day distribution. The selection of the nearest zone is based on travel time. In case the delivery of the parcel was unsuccessful, then the van returns and stores the parcels in the depot. The list of remaining parcels is updated and they are again put in the loop to be added to new parcels lists. Once all parcels are scheduled then a matrix of tours and trips is created. Each tour is described by its trips (*T*), the vehicle type (vt), the start time (t) and the parcels it consists of (type of parcel: gt_n and weight of parcel: *w_n*). Each trip is described by its origin and destination (*fo, fd*) by the goods type (*gt*) the weight (*w*), the vehicle type (*vt*), and by the tour ID (*Tid*).

2 Requirements.

2.1 Software requirements

The simulators have been built using Python version 3.8.8.
The following Python libraries need to be installed:

- python-dotenv==0.19.2
- pandas==1.3.4
- numpy==1.22
- pyshp==2.1.3

2.2 Input/Outputs

2.2.1 Inputs

The inputs of the parcel generation module are described in Table 1.

Table 1 Parcel Schedule– Inputs

Inputs	Description
Parameters.txt	Text file containing the parameters of the model
PARCELS Demand home delivery	Parcels trips to and from house to CEP depot
Parcels Demand Hub2Hub	Parcel trips between depots (consolidated trips)
TimeSkimMatrix.mtx	Matrix that contains the time (in seconds) between each pair OD
DistanceSkimMatrix.mtx	Matrix that contains the distance (in kilometres) between each pair OD

StudyAreaShapefile	Shapefile of the city. The areas delimited should be linked with the areas in the Socioeconomic Data file.
SocioeconomicData	CSV with socioeconomic data per area within the city. The mandatory fields are “zone”; "1: woningen"; "9: arbeidspl_totaal"
ParcelNodes	Shape with the location of the distribution nodes of the couriers
Departure Time Parcels CDF	Csv with the cumulative distribution function for the delivery time departure
External Zones	Csv with the external zones of the study area

2.2.2 Outputs

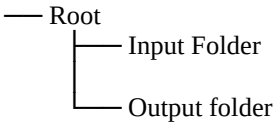
The outputs of the parcel generation module are described in Table 2.

Table 2 Parcel Schedule – Outputs

Outputs	Description
ParcelSchedule	Delivery vehicle schedules for final customer
KPIs.json	File containing key KPIs estimated in the model
Log.txt	Logs of the run
ParcelSchedule_Hub2Hub	Delivery vehicle schedules between depots

2.3 Paths structure

The directory where the model is located has the following structure:



3 Model Description

This section describes the different files and scripts present in the model

File name	Location	Description
Parcel_Sched.py	Root	Main script
__functions__.py	Root	External functions
requirements.txt	Root	Python packages required

4 Instructions to run the model

4.1 Command line execution of the model

4.1.1 Instructions and commands

The instruction to install the packages needed:

- pip install -r requirements.txt

The instruction to run the model

python3 Parcel_Sched.py Test Input Output Params_ParcelSched.txt ParcelDemand_Test.csv ParcelDemand_Hub2Hub_Test.csv skimTijd_new_REF.mtx skimAfstand_new_REF.mtx Zones_v4.shp SEGS2020.csv parcelNodes_v2.shp

4.1.2 Arguments

The arguments in the instructions to run the model are:

Table 3 Parcel Schedule – Input arguments

Arg[0]	Script name
Arg[1]	Lable (name of scenario)
Arg[2]	Input folder name
Arg[3]	Output folder name
Arg[4]	Parameter text file
Arg[5]	Parcel home delivery trips
Arg[6]	Parcel consolidated tri[s
Arg[7]	Time skim matrx
Arg[8]	Distance skim matrix

Arg[9]	Zone shapefile
Arg[10]	Socioec data
Arg[11]	Parcel Nodes shapefile

4.2 Requirements

4.2.1 Testing requirements

pip install -r requirements.txt

4.2.2 Input folder (Arg[2])

Folder 1(e.g. Input)

Which files are in the folder, what do they do, what type of file (csv, etc) what requirements do they have (table with headers, what each row represent, etC)

Table 4 Parcel Schedule – Inputs

Inputs	Type	Requirements
Parameters.txt	.txt	
ParcelDemand	.csv	Required cols: “Parcel_ID“ “O_zone“ “D_zone“ “CEP“ “DepotNumber“ “Task“
ParcelDemand_Hub2Hub	.csv	Required cols: “Parcel_ID“ “O_zone“ “D_zone“ “DepotNumber“ “CEP“ “Network“ “Type“
TimeSkimMatrix.mtx	.mtm	Id of areas ordered increasingly

DistanceSkimMatrix.mtx	.mtm	Id of areas ordered increasingly
StudyAreaShapefile	.shp	.dbf with the same name
SocioeconomicData	.csv	Required cols: “zone”; "1: woningen"; "9: arbeidspl_totaal"
ParcelNodes	.shp	.dbf with the same name

Shipment Module V1.0

Documentation

Author(s): Rodrigo Tapia, Ali Nadi

Author’(s’) affiliation (Partner short name): TU DELFT

Revision History

Version No.	Date	Details
1.0	26/08/22	

Contents

List of tables

1 Introduction

1.1 Scope and objectives

The simulation of freight transport activities requires generation of shipments for each logistic firms. The firms can be generated by a firm synthesis procedure. The shipment simulator simulates different logistic choices of these firms. First, it synthesize shipments which generates a set of shipments that are transported in the study area. Then, these shipments are distributed and scheduled based on a sequence of logistic processes: producer selection; distribution channel choices; shipment size & vehicle type choice; and desired delivery time. Finally, the distributed shipments are assigned to vehicles' tour through a bahavioural tour formation procedure. The tour formation procedure chooses the time for each tour based on the desired delivery times, and optimizes the sequence of trips in each tour.

2 Requirements.

2.1 Software requirements

The simulators have been built using Python version 3.8.8.

The following Python libraries need to be installed:

1. pandas==1.3.4
2. pyshp==2.1.3
3. tk==0.1.0
4. numpy==1.19.1
5. scipy==1.5.0
6. shapely==1.7.0
7. numba==0.53.0

2.2 Input/Outputs

2.2.1 Inputs

The inputs of the Shipment simulator are described in Table 1.

Table 1 Shipment Module– Inputs

Inputs	Description
skimTijd_new_REF.mtx	Travel time skim matrix
skimAfstand_new_REF.mtx	Distance skim matrix
nodes_v5.shp	logistics nodes
Zones_v6.shp	study area
SEGS2020.csv	Socioeconomic Data
parcelNodes_v2.shp	Parcel depot nodes
distributieCentra.csv	Distribution centers
nstrToLogisticSegment.csv	Converion NSTR to Logistic segments
MakeDistribution.csv	Distribution of making shipments per logistic sectors
UseDistribution.csv	Distribution of using shipments per logistic sectors
SupCoordinatesID.csv	External Zones
CorrectionsTonnes2016.csv	Corrections of Tonnes for the base year
CEPshares.csv	Market share of CEPs
Cost_VehType_2016.csv	Cost of transport per vehicle type
Cost_Sourcing_2016.csv	Cost of out sourcing
NUTS32013toMRDH.csv	NUTS32013 to MRDH conversion
CarryingCapacity.csv	Capacity of vehicles
LogFlowtype_Shares.csv	Market share of logistics flow type
Params_TOD.csv	Parameters of time of day choice model
Params_ShipSize_VehType.csv	Parameters of shipment size and vehicle type choice model
Params_EndTourFirst.csv	Parameters of the end of tour choice model for the later visited locations
Params_EndTourLater.csv	Paraeters of the end of tour chice model fo the later visited locations
ConsolidationPotential.csv	Consolidation potentials for different logistics sectors synthesized firms specifications

ZEZscenario.csv	Specifications for the zero emmission zones in the study area
Firms.csv	Synthesised firms location and specifications

2.2.2 Outputs

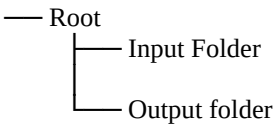
The outputs of the shipment module module are described in Table 2.

Table 2 Shipment Module– Outputs

Outputs	Description
Shipments_REF.csv	Shipments
Shipments_REF.shp	Shapefile of shipments for visualization
zonal_attractions_REF.csv	Zonal freigfht trip attraction
zonal_productions_REF.csv	Zonal freight trip production
Logfile_ShipmentSynthesizer.log	Logs of the run

2.3 Paths structure

The directory where the model is located has the following structure:



3 Model Description

This section describes the different files and scripts present in the model

File name	Location	Description
__module_SHIP__.py	Root	Main script
__functions__.py	Root	External functions
requirements.txt	Root	Python packages required

Instruction.txt	Root	Instruction to run code from console

4 Instructions to run the model

4.1 Command line execution of the model

4.1.1 Instructions and commands

The instruction to install the packages needed:

- pip install -r requirements.txt

The instruction to run the model

python3 __module_SHIP__.py REF Input Output skimTijd_new_REF.mtx skimAfstand_new_REF.mtx nodes_v5.shp Zones_v6.shp SEGS2020.csv parcelNodes_v2.shp distributieCentra.csv nstrToLogisticSegment.csv MakeDistribution.csv UseDistribution.csv SupCoordinatesID.csv CorrectionsTonnes2016.csv CEPshares.csv Cost_VehType_2016.csv Cost_Sourcing_2016.csv NUTS32013toMRDH.csv CarryingCapacity.csv LogFlowtype_Shares.csv Params_TOD.csv Params_ShipSize_VehType.csv Params_EndTourFirst.csv Params_EndTourLater.csv ConsolidationPotential.csv ZEZscenario.csv Firms.csv

4.1.2 Arguments

The arguments in the instructions to run the model are:

Table 3 Shipment Module– Inputs arguments for command line

Arg[0]	Script name
Arg[1]	Lable(Name of the Scenario)
Arg[2]	Input folder name
Arg[3]	Output folder name
Arg[4]	Time skim matrx
Arg[5]	Distance skim matrix
Arg[6]	logistics nodes
Arg[7]	study area
Arg[8]	Socioeconomic Data

Arg[9]	Parcel depot nodes
Arg[10]	Distribution centers
Arg[11]	converion NSTR to Logistic segments
Arg[12]	Distribution of making shipments per logistic sectors
Arg[13]	Distribution of using shipments per logistic sectors
Arg[14]	External Zones
Arg[15]	correction of tonnes
Arg[16]	Courier Market Shares
Arg[17]	cost per vehicle types
Arg[18]	cost of out sourcing
Arg[19]	NUTS32013 to MRDH conversion
Arg[20]	carrying capacity
Arg[21]	market share of logistic flow types
Arg[22]	parameters of time of day choice model
Arg[23]	paraeters of shipment size and vehicle type choice model
Arg[24]	Parameters of end of tour choice model for the first visited location
Arg[25]	Parameters of the end of tour choice model for the later visited locations
Arg[26]	consolidation potentials for different logistics sectors
Arg[27]	specifications for the zero emmission zones in the study area
Arg[28]	synthesized firms specifications

4.2 Requirements

4.2.1 Testing requirements

pip install -r requirements.txt

4.2.2 Input folder (Arg[2])

Folder 1(e.g. Input)

Table 4 Shipment Module– Inputs and descriptions

Inputs	Type	Description
skimTijd_new_REF.mtx	mtx	Id of areas ordered increasingly
skimAfstand_new_REF.mtx	mtx	Id of areas ordered increasingly

nodes_v5.shp	shp	Logistics node shapefile
Zones_v6.shp	shp	Zones of the study area
SEGS2020.csv	csv	Required cols: “zone”; "1: woningen"; "9: arbeidspl_totaal"
parcelNodes_v2.shp	shp	Parcel depot nodes shapefile
distributieCentra.csv	csv	Distribution centers "colnames": [{ "oppervlak":"float", "WP": "int", "Xcoor": "float", "Ycoor": "float", "AREANR": "int" }]
nstrToLogisticSegment.csv	csv	converion NSTR to Logistic segments
MakeDistribution.csv	csv	Distribution of making shipments per logistic sectors
UseDistribution.csv	csv	Distribution of using shipments per logistic sectors
SupCoordinatesID.csv	csv	Required cols: “COROP”; " Xcoor "; “Ycoor”; " AREANR ";

CorrectionsTonnes2016.csv	csv	correction of tonnes
CEPshares.csv	csv	Required cols: “CEP”; "ShareTotal
Cost_VehType_2016.csv	csv	cost per vehicle types "columns":[{ "CostPerKm":"float", "CostPerH": "float" }]
Cost_Sourcing_2016.csv	csv	cost of out sourcing "columns":[{ "CostPerKm":"float", "CostPerH": "float" }]
NUTS32013toMRDH.csv	csv	NUTS32013 to MRDH conversion
CarryingCapacity.csv	csv	carrying capacity "columns":[{ "Vehicle Type":"str", "Tonnes": "float" }]
LogFlowtype_Shares.csv	csv	market share of logistic flow types
Params_TOD.csv	csv	parameters of time of day choice model

Params_ShipSize_VehType.csv	csv	paraeters of shipment size and vehicle type choice model
Params_EndTourFirst.csv	csv	Parameters of end of tour choice model for the first visited location
Params_EndTourLater.csv	csv	Parameters of the end of tour choice model for the later visited locations
ConsolidationPotential.csv	csv	consolidation potentials for different logistics sectors
ZEZscenario.csv	csv	specifications for the zero emmission zones in the study area
Firms.csv	csv	synthesized firms specifications

Shipment Tour formation V1.0

Documentation

Author(s): Rodrigo Tapia, Ali Nadi

Author’(s’) affiliation (Partner short name): TU DELFT

Revision History

Version No.	Date	Details
1.0	26/08/22	

Contents

List of tables

1 Introduction

1.1 Scope and objectives

After generation of the shipments from to the logistic firms, the shipment simulator use tour formation module to schedule the tours from the shipments. This module use three set of choice models e.g. time of day choice model, end of tour first, and end of tour later choice models to assign shipmnets to the tours and use 2-opt algorithm to schedule tours.

2 Requirements.

2.1 Software requirements

The simulators have been built using Python version 3.8.8.
The following Python libraries need to be installed:

1. pandas==1.3.4
2. pyshp==2.1.3
3. tk==0.1.0
4. numpy==1.19.1
5. scipy==1.5.0
6. shapely==1.7.0
7. numba==0.53.0

2.2 Input/Outputs

2.2.1 Inputs

The inputs of the Tour formation simulator are described in Table 1.

Table 1 Tour formation– Inputs and description

Inputs	Description
skimTijd_new_REF.mtx	Travel time skim matrix
skimAfstand_new_REF.mtx	Distance skim matrix
Zones_v6.shp	study area

SEGS2020.csv	Socioeconomic Data
distributieCentra.csv	Distribution centers
SupCoordinatesID.csv	External Zones
CarryingCapacity.csv	Capacity of vehicles
Params_TOD.csv	Parameters of time of day choice model
Params_EndTourFirst.csv	Parameters of the end of tour choice model for the later visited locations
Params_EndTourLater.csv	Paraeters of the end of tour chice model fo the later visited locations
Shipments_REF.csv	Paraeters of the end of tour chice model fo the later visited locations
logistic_segment.txt	Logistics segments
nstr.txt	Commodity types
vehicle_type.txt	Vehicle types
combustion_type.txt	Engine types

2.2.2 Outputs

The outputs of the Tour formation module are described in Table 2.

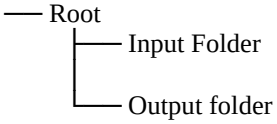
Table 2 Tour formation module– Outputs

Outputs	Description
Shipments_AfterScheduling_REF.csv	Shipment after scheduling
Tours_REF.csv	Generated tours
Tours_REF.shp	Shapefile of the shipments
tripmatrix_REF.txt	Trip Matrix for the reference day
tripmatrix_REF_TOD0.txt	Trip Matrix for the hour 0:00
tripmatrix_REF_TOD1.txt	Trip Matrix for the hour 1:00
tripmatrix_REF_TOD2.txt	Trip Matrix for the hour 2:00
tripmatrix_REF_TOD3.txt	Trip Matrix for the hour 3:00

tripmatrix_REF_TOD4.txt	Trip Matrix for the hour 4:00
tripmatrix_REF_TOD5.txt	Trip Matrix for the hour 5:00
tripmatrix_REF_TOD6.txt	Trip Matrix for the hour 6:00
tripmatrix_REF_TOD7.txt	Trip Matrix for the hour 7:00
tripmatrix_REF_TOD8.txt	Trip Matrix for the hour 8:00
tripmatrix_REF_TOD9.txt	Trip Matrix for the hour 9:00
tripmatrix_REF_TOD10.txt	Trip Matrix for the hour 10:00
tripmatrix_REF_TOD11.txt	Trip Matrix for the hour 11:00
tripmatrix_REF_TOD12.txt	Trip Matrix for the hour 12:00
tripmatrix_REF_TOD13.txt	Trip Matrix for the hour 13:00
tripmatrix_REF_TOD14.txt	Trip Matrix for the hour 14:00
tripmatrix_REF_TOD15.txt	Trip Matrix for the hour 15:00
tripmatrix_REF_TOD16.txt	Trip Matrix for the hour 16:00
tripmatrix_REF_TOD17.txt	Trip Matrix for the hour 17:00
tripmatrix_REF_TOD18.txt	Trip Matrix for the hour 18:00
tripmatrix_REF_TOD19.txt	Trip Matrix for the hour 19:00
tripmatrix_REF_TOD20.txt	Trip Matrix for the hour 20:00
tripmatrix_REF_TOD21.txt	Trip Matrix for the hour 21:00
tripmatrix_REF_TOD22.txt	Trip Matrix for the hour 22:00
tripmatrix_REF_TOD23.txt	Trip Matrix for the hour 23:00
Logfile_TourFormation.log	Log file for the tour formation

2.3 Paths structure

The directory where the model is located has the following structure:



3 Model Description

This section describes the different files and scripts present in the model

File name	Location	Description
__module_TOUR__.py	Root	Main script
__functions__.py	Root	External functions
requirements.txt	Root	Python packages required

4 Instructions to run the model

4.1 Command line execution of the model

4.1.1 Instructions and commands

The instruction to install the packages needed:

- pip install -r requirements.txt

The instruction to run the model

python3 __module_TOUR__.py REF Input Output skimTijd_new_REF.mtx skimAfstand_new_REF.mtx Zones_v6.shp SEGS2020.csv distributieCentra.csv SupCoordinatesID.csv CarryingCapacity.csv Params_TOD.csv Params_EndTourFirst.csv Params_EndTourLater.csv

4.1.2 Arguments

The arguments in the instructions to run the model are:

Table 3 Tour model – Input arguments for command line

Arg[0]	Script name
Arg[1]	Lable (name of scenario)
Arg[2]	Input folder name
Arg[3]	Output folder name

Arg[4]	Time skim matrx
Arg[5]	Distance skim matrix
Arg[6]	Zone shapefile
Arg[7]	Socioec data
Arg[8]	Distribution centers
Arg[9]	External zones
Arg[10]	Carriying capacity
Arg[11]	Params_TOD
Arg[12]	Params_EndTourFirst
Arg[13]	Params_EndTourLater

4.2 Requirements

4.2.1 Testing requirements

pip install -r requirements.txt

4.2.2 Input folder (Arg[2])

Folder 1(e.g. Input)

Which files are in the folder, what do they do, what type of file (csv, etc) what requirements do they have (table with headers, what each row represent, etC)

Table 4 Tour module– Inputs and requirments

Inputs	Type	Requirements
skimTijd_new_REF	.mtx	Id of areas ordered increasingly
skimAfstand_new_REF	.mtx	Id of areas ordered increasingly
Zones_v6		.dbf with the same name
SEGS2020	.csv	Required cols: “zone”; "1: woningen"; "9: arbeidspl_totaal"

distributieCentra.csv	.csv	Distribution centers "colnames": [{ "oppervlak":"float", "WP": "int", "Xcoor": "float", "Ycoor": "float", "AREANR": "int" }]
SupCoordinatesID	.csv	Required cols: “COROP”; " Xcoor "; “Ycoor”; " AREANR ";
CarryingCapacity	.csv	carrying capacity "columns":[{ "Vehicle Type":"str", "Tonnes": "float" }]

Params_TOD	.csv	parameters of time of day choice model
Params_EndTourFirst	.csv	Parameters of end of tour choice model for the first visited location
Params_EndTourLater	.csv	Parameters of the end of tour choice model for the later visited locations
logistic_segment	.txt	Logistic segment "columns":[{ "ID":"Int", "comment": "str" }]
nstr	.txt	Commodity types "columns":[{ "ID":"Int", "comment": "str" }]
vehicle_type	.txt	Vehicle type "columns":[{ "ID":"Int", "IsRefTypeFreight": "boolean", "IsAvailableInParcelModule":" boolean", " Comment ":"str", }]
combustion_type	.txt	Combustion type "columns":[{

		"ID":"Int", "comment": "str" }]

Network Assignment Module V1.0

Documentation

Author(s): Rodrigo Tapia, Ali Nadi

Author’(s’) affiliation (Partner short name): TU DELFT

Revision History

Version No.	Date	Details
1.0	26/08/22	

Contents

List of tables

1 Introduction

1.1 Scope and objectives

The Network assignemnt module is a static traffic assignment developed to assign trip matrices generated from parcel and shipment scheduling modules to the road networks.the result of this model is the intensities on road networks where the number of truck passing each link can be validated with actual truck counts.

2 Requirements.

2.1 Software requirements

The simulators have been built using Python version 3.8.8.
The following Python libraries need to be installed:

1. pandas==1.3.4
2. pyshp==2.1.3
3. tk==0.1.0
4. numpy==1.19.1
5. scipy==1.5.0
6. shapely==1.7.0
7. numba==0.53.0

2.2 Input/Outputs

2.2.1 Inputs

The inputs of the Shipment simulator are described in Table 1.

Table 1 Network assignment module– Inputs files

Inputs	Description
skimTijd_new_REF.mtx	Travel time matrix
skimAfstand_new_REF.mtx	Distance matrix

links_v5.shp	Road networks in the study area
nodes_v5.shp	Shapefile of the Logistics nodes
Zones_v6.shp	Shapefile of the traffic analysis zones
SEGS2020.csv	Sociodemographics of each zone
SupCoordinatesID.csv	External zones
Cost_VehType_2016.csv	Cost of transport per vehicle types
Cost_Sourcing_2016.csv	Cost of outsourcing
CarryingCapacity.csv	Capacity of the vehicles
EmissieFactoren_BUITENWEG_LEEG.csv	Emission factors for roads outside the cities when they are empty
EmissieFactoren_BUITENWEG_VOL.csv	Emission factors for roads outside the cities when they are full
EmissieFactoren_SNELWEG_LEEG.csv	Emission factors for motorways when they are empty
EmissieFactoren_SNELWEG_VOL.csv	Emission factors for motorways when they are full
EmissieFactoren_STAD_LEEG.csv	Emission factors for roads inside cities when they are empty
EmissieFactoren_STAD_VOL.csv	Emission factors for roads inside cities when they are full
emission_type.txt	Types of emission
logistic_segment.txt	Logistics segments
ParcelSchedule_REF.csv	Parcel schedule generated by parcel scheduler
Shipments_AfterScheduling_REF.csv	Shipment schedules generated by shipment scheduler
Tours_REF.csv	Tours generated bu tour module
tripmatrix_parcels_REF.txt	Trip matrix for parcels
tripmatrix_parcels_REF_TOD0.txt	Trip matrix for parcels at 0:00
tripmatrix_parcels_REF_TOD1.txt	Trip matrix for parcels at 1:00
tripmatrix_parcels_REF_TOD2.txt	Trip matrix for parcels at 2:00

tripmatrix_parcels_REF_TOD3.txt	Trip matrix for parcels at 3:00
tripmatrix_parcels_REF_TOD4.txt	Trip matrix for parcels at 4:00
tripmatrix_parcels_REF_TOD5.txt	Trip matrix for parcels at 5:00
tripmatrix_parcels_REF_TOD6.txt	Trip matrix for parcels at 6:00
tripmatrix_parcels_REF_TOD7.txt	Trip matrix for parcels at 7:00
tripmatrix_parcels_REF_TOD8.txt	Trip matrix for parcels at 8:00
tripmatrix_parcels_REF_TOD9.txt	Trip matrix for parcels at 9:00
tripmatrix_parcels_REF_TOD10.txt	Trip matrix for parcels at 10:00
tripmatrix_parcels_REF_TOD11.txt	Trip matrix for parcels at 11:00
tripmatrix_parcels_REF_TOD12.txt	Trip matrix for parcels at 12:00
tripmatrix_parcels_REF_TOD13.txt	Trip matrix for parcels at 13:00
tripmatrix_parcels_REF_TOD14.txt	Trip matrix for parcels at 14:00
tripmatrix_parcels_REF_TOD15.txt	Trip matrix for parcels at 15:00
tripmatrix_parcels_REF_TOD16.txt	Trip matrix for parcels at 16:00
tripmatrix_parcels_REF_TOD17.txt	Trip matrix for parcels at 17:00
tripmatrix_parcels_REF_TOD18.txt	Trip matrix for parcels at 18:00
tripmatrix_parcels_REF_TOD19.txt	Trip matrix for parcels at 19:00
tripmatrix_parcels_REF_TOD20.txt	Trip matrix for parcels at 20:00
tripmatrix_parcels_REF_TOD21.txt	Trip matrix for parcels at 21:00
tripmatrix_parcels_REF_TOD22.txt	Trip matrix for parcels at 22:00
tripmatrix_parcels_REF_TOD23.txt	Trip matrix for parcels at 23:00
tripmatrix_REF.txt	Trip matrix for Shipments
tripmatrix_REF_TOD0.txt	Trip matrix for Shipments at 0:00
tripmatrix_REF_TOD1.txt	Trip matrix for Shipments at 1:00
tripmatrix_REF_TOD2.txt	Trip matrix for Shipments at 2:00
tripmatrix_REF_TOD3.txt	Trip matrix for Shipments at 3:00
tripmatrix_REF_TOD4.txt	Trip matrix for Shipments at 4:00
tripmatrix_REF_TOD5.txt	Trip matrix for Shipments at 5:00

tripmatrix_REF_TOD6.txt	Trip matrix for Shipments at 6:00
tripmatrix_REF_TOD7.txt	Trip matrix for Shipments at 7:00
tripmatrix_REF_TOD8.txt	Trip matrix for Shipments at 8:00
tripmatrix_REF_TOD9.txt	Trip matrix for Shipments at 9:00
tripmatrix_REF_TOD10.txt	Trip matrix for Shipments at 10:00
tripmatrix_REF_TOD11.txt	Trip matrix for Shipments at 11:00
tripmatrix_REF_TOD12.txt	Trip matrix for Shipments at 12:00
tripmatrix_REF_TOD13.txt	Trip matrix for Shipments at 13:00
tripmatrix_REF_TOD14.txt	Trip matrix for Shipments at 14:00
tripmatrix_REF_TOD15.txt	Trip matrix for Shipments at 15:00
tripmatrix_REF_TOD16.txt	Trip matrix for Shipments at 16:00
tripmatrix_REF_TOD17.txt	Trip matrix for Shipments at 17:00
tripmatrix_REF_TOD18.txt	Trip matrix for Shipments at 18:00
tripmatrix_REF_TOD19.txt	Trip matrix for Shipments at 19:00
tripmatrix_REF_TOD20.txt	Trip matrix for Shipments at 20:00
tripmatrix_REF_TOD21.txt	Trip matrix for Shipments at 21:00
tripmatrix_REF_TOD22.txt	Trip matrix for Shipments at 22:00
tripmatrix_REF_TOD23.txt	Trip matrix for Shipments at 23:00
TripsVanConstruction.mtx	Trips of vans for construction industries
TripsVanService.mtx	Service trips made by Vans
vehicle_type.txt	Vehicle types

2.2.2 Outputs

The outputs of the Traffic module are described in Table 2.

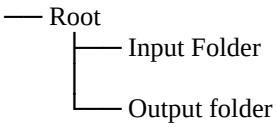
Table 2 Network assignment module– Outputs description

Outputs	Description
links_loaded_REF_intensities.csv	Links loaded with truck intensities

ParcelSchedule_REF_Emission.csv	Parcel schedules with emissions
Shipments_AfterScheduling_REF_Emission.csv	Shipment schedules with emissions
Tours_REF_Emission.csv	Tours with emission
links_loaded_REF.shp	Shape file of the loaded link with truck intensities
Logfile_TrafficAssignment.log	Log file of the simulation for network assignment

2.3 Paths structure

The directory where the model is located has the following structure:



3 Model Description

This section describes the different files and scripts present in the model

File name	Location	Description
__module_TRAF__.py	Root	Main script
__functions__.py	Root	External functions
requirements.txt	Root	Python packages required
Instruction.txt	Root	Instruction to run code from console

4 Instructions to run the model

4.1 Command line execution of the model

4.1.1 Instructions and commands

The instruction to install the packages needed:

- pip install -r requirements.txt

The instruction to run the model

python3 __module_TRAF__.py Label Input Output skimTijd_new_REF.mtx skimAfstand_new_REF.mtx nodes_v5.shp Zones_v6.shp SocioeconomicData.csv links_v5.shp SupCoordinatesID.csv Cost_VehType_2016.csv Cost_Sourcing_2016.csv CarryingCapacity.csv EmissieFactoren_BUITENWEG_LEEG.csv EmissieFactoren_BUITENWEG_VOL.csv EmissieFactoren_SNELWEG_LEEG.csv EmissieFactoren_SNELWEG_VOL.csv EmissieFactoren_STAD_LEEG.csv missieFactoren_STAD_VOL.csv

4.1.2 Arguments

The arguments in the instructions to run the model are as follows. Please note that the inputs files that are needed in the arguments are those which are not generated by any other models. The network assignment requires a lot more inputs as expressed in the Table 1 which are generated by shipment, tour and parcel modules.

Table 3 Network assignment– Inputs for command line

Arg[0]	Script name
Arg[1]	Lable(Name of the Scenario)
Arg[2]	Input folder name
Arg[3]	Output folder name
Arg[4]	Time skim matrix
Arg[5]	Distance skim matrix
Arg[6]	logistics nodes
Arg[7]	study area
Arg[8]	Socioeconomic Data
Arg[9]	Network links
Arg[10]	External Zones
Arg[11]	Cost per vehicle type
Arg[12]	Cost of our-sourcing
Arg[13]	Carrying capacity
Arg[14]	Emission factors for roads outside the cities when they are empty
Arg[15]	Emission factors for roads outside the cities when they are full
Arg[16]	Emission factors for motorways when they are empty
Arg[17]	Emission factors for motorways when they are full
Arg[18]	Emission factors for roads inside cities when they are empty
Arg[19]	Emission factors for roads inside cities when they are full

4.2 Requirements

4.2.1 Testing requirements

pip install -r requirements.txt

4.2.2 Input folder (Arg[2])

Folder 1(e.g. Input)

Table 4 Network assignment– Inputs description

Inputs	Type	Description
skimTijd_new_REF.mtx	mtx	Id of areas ordered increasingly
skimAfstand_new_REF.mtx	mtx	Id of areas ordered increasingly
links_v5.shp	shp	
nodes_v5.shp	shp	Logistics node shapefile
Zones_v6.shp	shp	Zones of the study area
SEGS2020.csv	csv	Required cols: “zone”; "1: woningen"; "9: arbeidspl_totaal"
SupCoordinatesID.csv	csv	Required cols: “COROP”; " Xcoor "; “Ycoor”; " AREANR ";
Cost_VehType_2016.csv	csv	cost per vehicle types "columns":[{ "CostPerKm":"float", "CostPerH": "float" }]
Cost_Sourcing_2016.csv	csv	cost of out sourcing "columns":[{

		"CostPerKm":"float", "CostPerH": "float" }]
CarryingCapacity.csv	csv	carrying capacity "columns":[{ "Vehicle Type":"str", "Tonnes": "float" }]
EmissieFactoren_BUITENWEG_LEEG.csv	csv	Emission Factors "columns":[{ " Voertuigtype":"str", " CO2 (gram/km)": "float" " SO2 (mg/km)":"float", "PMv (mg/km)": "float" "NOx (g/km) ":"float", "PMslijtage (mg/km) ": "float" }]
EmissieFactoren_BUITENWEG_VOL.csv	csv	Emission Factors "columns":[{ " Voertuigtype":"str", " CO2 (gram/km)": "float"

		" SO2 (mg/km)":"float", "PMv (mg/km)": "float" "NOx (g/km) ":"float", "PMslijtage (mg/km) ": "float" }]
EmissieFactoren_SNELWEG_LEEG.csv	csv	Emission Factors "columns":[{ " Voertuigtype":"str", " CO2 (gram/km)": "float" " SO2 (mg/km)":"float", "PMv (mg/km)": "float" "NOx (g/km) ":"float", "PMslijtage (mg/km) ": "float" }]
EmissieFactoren_SNELWEG_VOL.csv	csv	Emission Factors "columns":[{ " Voertuigtype":"str", " CO2 (gram/km)": "float" " SO2 (mg/km)":"float",

		"PMv (mg/km)": "float" "NOx (g/km) ":"float", "PMslijtage (mg/km) ": "float" }]
EmissieFactoren_STAD_LEEG.csv	csv	Emission Factors "columns":[{ " Voertuigtype":"str", " CO2 (gram/km)": "float" " SO2 (mg/km)":"float", "PMv (mg/km)": "float" "NOx (g/km) ":"float", "PMslijtage (mg/km) ": "float" }]
EmissieFactoren_STAD_VOL.csv	csv	Emission Factors "columns":[{ " Voertuigtype":"str", " CO2 (gram/km)": "float" " SO2 (mg/km)":"float", "PMv (mg/km)": "float"

		"NOx (g/km) ":"float", "PMslijtage (mg/km) ": "float" }]
emission_type.txt	txt	Emission type "columns":[{ "ID":"Int", "comment": "str" }]
logistic_segment.txt	txt	Logistic segment "columns":[{ "ID":"Int", "comment": "str" }]
TripsVanConstruction.mtx	mtx	Service trips matrix for construction
TripsVanService.mtx	mtx	Service Trip matrix
vehicle_type.txt	txt	Vehicle type "columns":[{ "ID":"Int", "IsRefTypeFreight": "boolean", "IsAvailableInParcelModule":" boolean", " Comment ":"str", }]

COMPUTER PROGRAM TO CALCULATE EMISSIONS FROM ROAD TRAFFIC – **COPERT**

Documentation

Author(s): Jose Manuel Vassallo, Natalia Sobrino, Juan Nicolas Gonzalez

Author'(s') affiliation (Partner short name): UPM

Revision History

Version No.	Date	Details
0	10/02/2022	Initial version

Contents

List of tables

1 Introduction

1.1 Scope and objectives

This document serves as technical documentation for the **COmputer Program to calculate Emissions from Road Traffic – COPERT**. It explains its scope, the requirements to run the program, the inputs and outputs, and how to access it.

COPERT is a Windows program used to calculate Emissions from Road Traffic and was designed in principle for National Experts to estimate emissions from road transport for inclusion in official annual national inventories. The Joint Research Centre of the European Commission has been coordinating the model's scientific development since 2007. The COPERT methodology is also part of the EMEP/CORINAIR Emission Inventory Guidebook. The COPERT methodology is entirely consistent with the Road Transport chapter of the Guidebook.

In addition, the use of a software application to compute road transport emissions enables a transparent and uniform data collection and reporting system, which is consistent and comparable, in conformity with international conventions and protocols and EU regulations.

In this project, we focus on the Tier 3 Method, where emissions are computed using a combination of firm technical data (e.g., emission factors) and activity data (e.g., total vehicle km). The methodology also includes expressions for cold start and evaporative emissions and accounts as far as possible for national variations in such parameters as vehicle parking duration, vehicle age, driving patterns, fuel composition, and climate.

Figure 1 shows the Tier 3 Method algorithm used to estimate emissions.

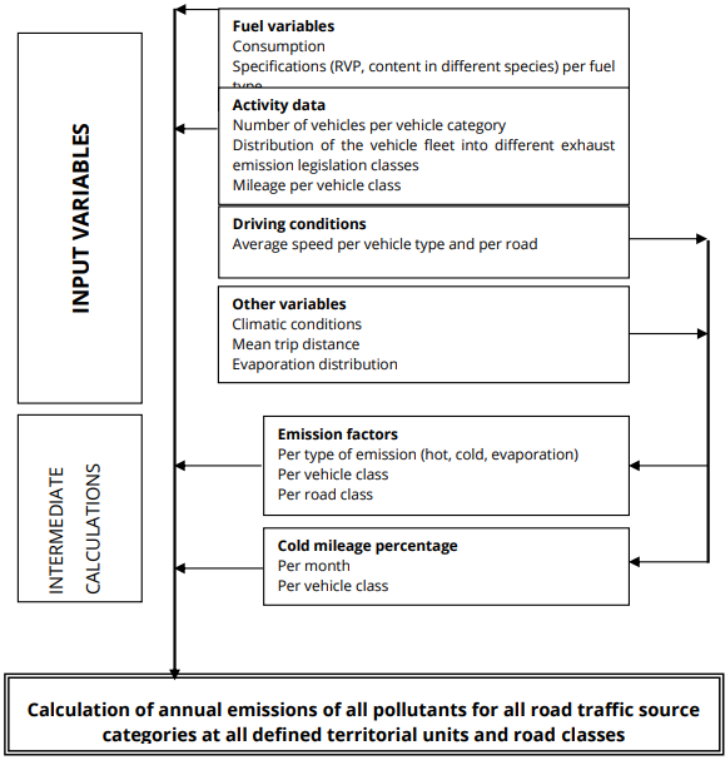


Figure 1 Tier 3 Method Algorithm. Source: EEA (European Environment Agency), 2020[1]

Moreover, in this project, we use the COPERT Command Line Interface feature that allows multiple automatic executions of COPERT by externally providing input data and subsequently receiving results in an external file.

2 Requirements.

2.1 Software requirements

COPERT 5.5 is a 32-bit application and requires a 32-bit/64-bit Microsoft Windows operating system.

The minimum software requirements are:

- Microsoft® Windows XP SP3 or newer Windows version
- Microsoft .NET Framework v4.0 Clinet Profile or later [Official Version Download]

The minimum hardware requirements are:

- 1 GHz, 32-bit(x86) processor

[1] Methodology for the calculation of exhaust emissions – SNAPs 070100-070500, NFRs 1A3bi-iv. October 2020.

- 1 GB RAM
- 30 MB for installation, 100 MB for a full run
- Screen Resolution: 1024x768 pixels

Note: COPERT runs in a LEAD Windows Server in Amazon Web Services cloud (AWS)

2.2 Input/Outputs

2.2.1 Inputs

COPERT needs an input Excel file containing the following inputs in different spreadsheets (see Table 1).

Table 1 COPERT – Inputs

Inputs	Description
MIN_TEMPERATURE	Environmental information – Minimum temperature per month (ºC)
MAX_TEMPERATURE	Environmental information – Maximum temperature per month (ºC)
HUMIDITY	Environmental information – Humidity temperature per month (%)
URBAN_PEAK_SHARE	Activity data - Percentage of traffic on urban peak hour (%)
URBAN_OFF_PEAK_SHARE	Activity data - Percentage of traffic on the urban off-peak hour (%)
URBAN_PEAK_SPEED	Activity data – Average speed of traffic on urban peak hour (km/h)
URBAN_OFF_PEAK_SPEED	Activity data – Average speed of traffic on the urban off-peak hour (km/h)
MEAN_ACTIVITY	Activity data – Average kilometers travel per vehicle type (km)
STOCK	Activity data – Number of vehicles per vehicle type (km)
VEHICLE_CATEGORY	Vehicles types available in the COPERT Database

2.2.2 Outputs

COPERT generates an Excel file with more than 113 spreadsheets. These spreadsheets contain the emission estimated for each pollutant and estimation steps (COLD, HOT, TYRE, etc.). In order to estimate Project KPI, we focus only on TOTAL emissions for each pollutant. Table 2 shows the central sheets that pose the necessary outputs generated in the Excel file.

Table 2 COPERT – Outputs

Outputs	Description
EC_TOTAL	Energy consumption (TJ)
CO2_TOTAL	Carbon Dioxide (t)
PM_2.5_TOTAL	Fine Particulate Matter (t)
NO2_TOTAL	Nitrogen Dioxide (t)
VOC_TOTAL	Volatile Organic Compounds (t)

2.3 Paths structure

- Root

3 Model Description

This section describes the different files and scripts present in the model

Table 3 Script and files

File name	Location	Description
inputCOPERT.xlsx	Root	File containing the excel input file

4 Instructions to run the model

4.1 Command line execution of the model

curl --location --request POST "3.120.144.230:8000" --form 'File=<input_directory_path>"' --form 'keep ="true"' --form 'noevap ="false"' --form 'alt_country =" alt_country "' --form 'EF="true"' --form 'EB="true"' --form 'outputFile="xlsx"' --output '<name>.zip'

Table 4 COPERT execution values

Attributes	Possible Values	Description
POST	3.120.144.230:8000	AWS URL and port
File	<input_directory_path>	Add the input excel file path
keep	true/false	to keep the file for further processing the “keep” switch at the end of the basic command prevents COPERT from deleting the file.
noevap	true/false	Save time up to 30% (true) and for N0x
alt_country	Six LEAD countries abbreviation	Country abbreviations based on the Alpha-2 code of the ISO 3166 country code.
outputfile	xlsx or json	Choose output format.
output	<name>.zip	The output is a zip file including 2 xlsx/json files.

4.2 Requirements

4.2.1 Testing requirements

Details of the software versions required

- NAME="Ubuntu"
- VERSION="20.04.2 LTS (Focal Fossa)"
- ID=ubuntu
- ID_LIKE=debian
- PRETTY_NAME="Ubuntu 20.04.2 LTS"
  VERSION_ID="20.04"
- HOME_URL="https://www.ubuntu.com/"
- SUPPORT_URL="https://help.ubuntu.com/"
- BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
- PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
- VERSION_CODENAME=focal
- UBUNTU_CODENAME=focal
- Kernel: Linux 5.4.0-66-generic
- Architecture: x86-64

4.2.2 Root

InputCOPERT.xlsx: contain the needed data to compute emission. Each spreadsheet has the headers predefined.

COMPUTER FUNCTION TO CALCULATE NOISE IMMISSIONS FROM ROAD TRAFFIC

COMFUNOISE

Documentation

Author(s): César Asensio, Ignacio Pavón

Author'(s') affiliation (Partner short name): UPM

Revision History

Version No.	Date	Details
0	3/06/2022	Initial version
1	13/10/2022	KPI defined. Model ready.
2	10/03/2023	Category Q4 added for electric van

Contents

List of tables

1 Introduction

1.1 Scope and objectives

COMPRONOISE is an algorithm for the calculation of noise emissions produced by road traffic in a city. It is a simplified version of the CNOSSOS-EU method that was developed by the JRC with the support of national experts from member states, and has become the standard for strategic noise mapping in Europe.
From the vehicle flow, speed and typology, the software will calculate the noise levels on the façades of the buildings, and from these data, determine the number of people exposed to noise, an indicator that is considered the ultimate objective of the software.
The model will use static data, at the living lab level, to describe the number of residents in the study area, the location of buildings in the study area and the characteristics of the streets. On the other hand, it will have dynamic inputs to update the noise description on the basis of a working day. The software will allow the calculation of the percentage of people exposed to noise above a reference threshold.

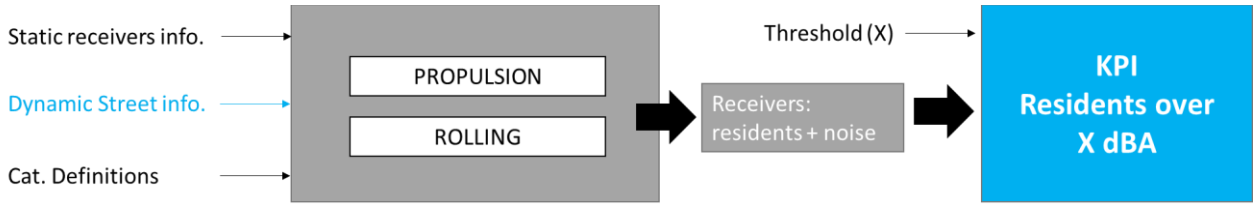


Figure 1 COMPRONOISE scheme

2 Requirements.

2.1 Software requirements

COMPRONOISE has been developed for use on the MatLab platform, and has also been tested on GNU/OCTAVE. Although the developments have only been tested in a Windows environment, given that both MatLAB and GNU/OCTAVE have versions for other work environments (iOS, Linux), it is considered that the code can be easily adapted to both of them.

COMPRONOISE requirements are derived from those provided by the platform where it is executed, depending on the version used and the operating system selected.

2.2 Input/Outputs

2.2.1 Inputs

COMPRONOISE requires, for each Living Lab, information about the buildings and the residents that live in them, as well as information about the routes travelled on the streets. In addition, it requires information on the typologies of vehicles in the city. Table 1 will describe all the parameters required as input.

Table 1 COMPRONOISE – Inputs

	Inputs	Description
MODEL	MODEL_CONST	The noise emission model is based on the CNOSSOS-EU method used for strategic noise mapping according to the environmental noise directive. The equations used in this method to calculate the sound power level of rolling and propulsion systems have constants for each vehicle category. In the absence of more precise estimates for each of the vehicle models used in the different Living Labs, COMPRONOISE will redefine 4 categories: - Category 1 (Q1): corresponding to the definition of light-duty vehicle in the CNOSSOS-EU method (CE categories M1). Constants, as defined in CNOSSOS-EU (Cat 1) - Category 2 (Q2): These are electric light vehicles. Constants for rolling noise as defined in CNOSSOS-EU (Cat 1). Constants for propulsion noise, 0. - Category 3 (Q3): corresponding to the definition of medium heavy-duty vehicle in the CNOSSOS-EU method (CE categories M2 and N2). Constants, as defined in CNOSSOS-EU (Cat 2) - Categories 4 (Q4). Electric version of Q3
MODEL	MODEL_ADJ	The software provides for a correction factor that can be added to the calculated power levels for each vehicle category, in case there are measurements available that can be used to adjust the model.
ROAD	ROAD_ID	Id of the street segment

ROAD	INTENSITY_CATx	For every road segment, number of vehicles of category x during the working day
ROAD	SPEED (KMH)	For every road segment, vehicles mean speed (alternatively, speed limit)
RECEIVER	RECEIVER_ID	For each façade receiver, ID of the point along every residential building. This parameter has been previously elaborated in a GIS environment.
RECEIVER	CLOSEST_ROAD	For every façade receiver, ID of the closest street segment. This parameter has been previously elaborated in a GIS environment.
RECEIVER	CLOSEST_DIST	For every façade receiver, distance to the closest street segment. This parameter has been previously elaborated in a GIS environment.
RECEIVER	RESIDENTS	For each façade receiver, number of residents in the building over the number of façade receivers for that building. This parameter has been previously elaborated in a GIS environment.
KPI	THRESHOLD	The KPI will describe the % of inhabitants exposed to noise levels over this reference value. In order to assess the impact of the project, this reference value has not been selected based on health impacts. For each Living Lab, the threshold is the median of the sound level calculated for all the façade receivers in the baseline scenario. BASELINE: 50% of the façade receivers will be below the THRESHOLD, 50% will be above this value.

2.2.2 Elaboration of receivers input file

Noise exposure calculations are performed on a series of receivers located on the exterior facades of all buildings. To obtain a file containing all these receivers, the following steps can be carried out.

1. From a GIS layer with the buildings, the necessary operations will be carried out to obtain the exterior contour of the blocks.
2. Assigned to each polygon, an estimate of the number of residents in the block will be available.

3. Along this outline, using GIS tools, a file of points will be generated with a separation of 10m.

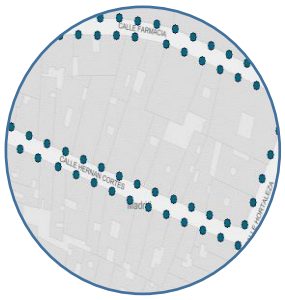


Figure 2 Receivers on block's facade

4. Using GIS, locate the nearest street segment and calculate the 2D distance to it.

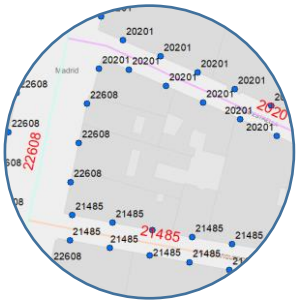


Figure 3 Receivers linked to street segments

5. Generate Fac_Point.csv, with the list of receivers, linked to the ROAD_ID and the distance to that road.

2.2.3 Outputs

COMPRONOISE will produce a spreadsheet file with the contents described in Table 2.

Table 2 COMPRONOISE – Outputs

Outputs	Description
RES_EXP	Residents exposed (%). Percentage of residents in the study area who are exposed to noise levels above the defined threshold (THRESHOLD).

2.3 Paths structure

- Root. City lab level, it includes the constants for vehicle emissions descriptpion and the file with façade receivers location.
- Root\Input. Where the model will search for travel files.

3 Model Description

This section describes the different files and scripts present in the model

File name	Location	Description
	Root	Script calculating KPI from Input folder, and writing resultin output folder
Travel.csv	Root\input_dynamic	Set of files describing City Lab's working day travels. All the files in the folder will be computed to calculate the KPI
Fac_Point.csv	Root\input_static	Static file describing City Lab's façade points for every building within the study area
Output.csv	Root\output	KPI. Dinamic file describing the % of people exposed to noise level over the threshold

4 Instructions to run the model

4.1 Command line execution of the model

Command line (GNU/OCTAVE or MatLab): Compronoise

It does not have any arguments. It takes the input files, and produces an output file.

4.2 Requirements

4.2.1 Root

This folder contains the function and the static files that describe the façade receivers and the constants describing the vehicle categories for each Living Lab.

4.2.2 Root\input

This folder contains a dynamic file with the list of road segments having the mean speed and the number of vehicles circulating on it during a working day.

4.2.3 Root\output

This folder contains a dynamic file with the relative histogram of the noise levels at the receivers, and the percentage people exposed to noise over the threshold.

Electric vehicle GHG emissions estimation

Documentation

Author(s): Jose Manuel Vassallo, Natalia Sobrino, Juan Nicolas Gonzalez

Author’(s’) affiliation (Partner short name): UPM

Revision History

Version No.	Date	Details
0	10/02/2022	Initial version

Contents

List of tables

1 Introduction

1.1 Scope and objectives

This document serves as technical documentation for the Electric vehicle GHG emissions estimation. It explains its scope, the requirements to run the program, the inputs and outputs, and how to access it.

This methodology to estimate the greenhouse gas emissions (GHG) from electric vehicles is associated with the electrical mix used to produce electricity. The method estimates the electric vehicles' CO_{2eq} emissions based on the sum of the mean electricity consumed by each vehicle (kWh) and the carbon intensity -the CO_{2eq} emissions per energy unit- of the electricity generation system of each region or study area (gCO_{2eq}/kWh). For instance, the greenhouse gas emissions intensity of power generation in the EU has been continuously decreasing over the last three decades, in 2020, the EU-27 GHG emissions intensity was 230.7 gCO2eq/kWh[1] (EEA, 2021).

To estimate Electric Vehicles (EV) CO_{2eq} emissions, the following methodology calculates the emissions based on mean kWh consumed by each vehicle and the carbon intensity for the electricity generation (gCO_{2eq}/kWh).

$$Em_{EV} = EC_{EV} * \sum_{k} P(k,t) * CI(k)$$

Em_{EV} = Total CO_{2eq} emissions of electric vehicles (gCO_{2eq})

EC_{EV} = Total electricity consumption of electric vehicles (kWh)

$P(k,t)$ = Participation of each power source in the electricity generation (%)

$CI(k)$ = Carbon intensity of each power source (gCO2e/kWh)

Figure 1 shows the flow followed by the methodology to estimate emissions.

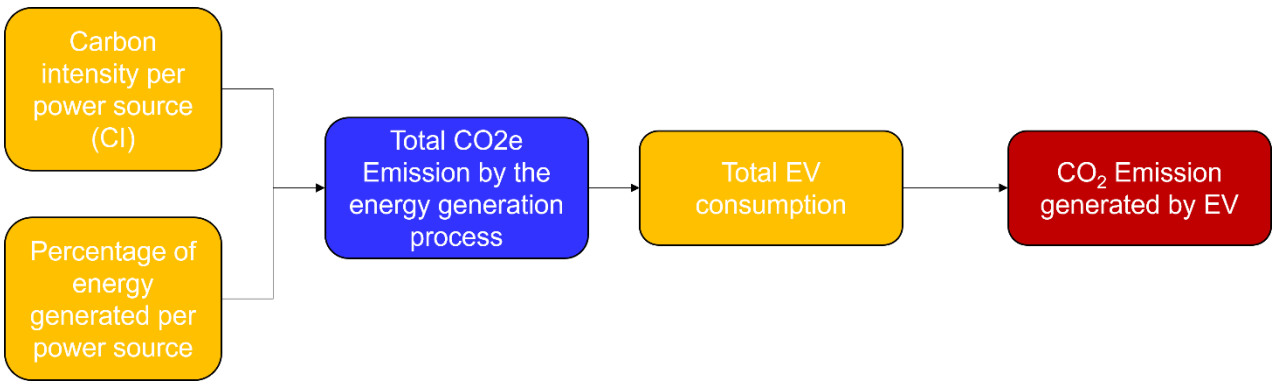


Figure 1 Methodology flow

[1] https://www.eea.europa.eu/data-and-maps/daviz/co2-emission-intensity-9/#tab-googlechartid_googlechartid_googlechartid_googlechartid_chart_11111

Carbon intensity for each country or region is generally updated annually. In addition, they are updated whenever circumstances so require. These emission factors take into account only direct emissions associated with electricity production and do not include indirect emissions associated with the construction of generation plants, transport of fuels, maintenance, etc.

For this model, we considered the following electricity production technologies:

- Coal-fired power plant
- Combined Cycle Thermal Power Plant (Natural Gas)
- Fuel-Gas Thermal Power Plant
- Cogeneration
- Waste

2 Requirements.

2.1 Software requirements

The simulators have been built using R version 4.0.2.

The following R libraries need to be installed:

1. “tidyr” – Tidy Messy Data – 1.1.4
2. “tidyverse” - Easily Install and Load the ‘Tidyverse’ – 1.3.1
3. "data.table" – Extension of data.frame – 1.14.2
4. “readxl” – Read Excel files – 1.3.1

2.2 Input/Outputs

2.2.1 Inputs

The model needs an input excel file containing the following information

Table 1 Electric vehicle GHG emissions estimation – Inputs

Inputs	Description
Emission_Factor	Carbon intensity of each power source (gCO2e/kWh)
Generation_percentage	The percentage of energy generated by every power source (%)
Stock	Number of electric vehicles used
Consumption	The total energy consumption required by the vehicles (kWh) is usually reported in kWh.

2.2.2 Outputs

Table 2 Electric vehicle GHG emissions estimation – Outputs

Outputs	Description
date	Date when the estimation was made
emissions_gCO2	Equivalent Carbon Dioxide (g)

2.3 Paths structure

- Root

3 Model Description

This section describes the different files and scripts present in the model

File name	Location	Description
EVCO2.R	Root	File containing the code
EV_FACTORS.xlsx	Root	Emission factor and generation percentage
EV_CONS.XLSX	Root	Stock and consumption

4 Instructions to run the model

4.1 Command line execution of the model

4.1.1 Instructions and commands

1. Navigate to the \bin subdirectory on your R version directory (C:\Program Files\R\your_R_version_directory \bin)
2. Search and add the path where the code Is located
3. Define args[1] (working directory)

"C:\Program Files\R\R-4.0.2\bin\Rscript.exe" ” <script path>" <args[1]>

4.1.2 Arguments

args[1] = working directory where the code and input and outpu files are located.

4.2 Requirements

4.2.1 Testing requirements

Details of the software versions required

- $platform
  - "x86_64-w64-mingw32"
- $arch
  - "x86_64"
- $os
  - "mingw32"
- $system
  - "x86_64, mingw32"
- $status
  - ""
- $major
  - “4”
- $minor
  - “0.2”
- $year
  - “2020”
- $month
  - “06”
- $day
  - “22”
- $`svn rev`
  - “78730”
- $language
  - “R”
- $version.string
  - "R version 4.0.2 (2020-06-22)"
- $nickname
  - “Taking Off Again”

4.2.2 Root

Script.R: File containing the code

EV_FACTORS.xlsx: Emission factor and generation percentage

EV_CONS.xlsx: Stock and consumption

2-echelon

Documentation

Author(s): Beatriz Royo

Author’(s’) affiliation (Partner short name): ZLC

Revision History

Version No.	Date	Details
1.0	25.11.2021	First version of the document.

Contents

List of tables

1 Introduction

1.1 Scope and objectives

The -ecommerce growth has increased the transport flows in the cities while policymakers are introducing delivery constraints with the creation of low emissions zones or low traffic zones that banned the access to fossil fuel or high size vehicles. This panorama is forcing last mile sector to shift from the business as usual (BAU) scenario or 1-echelon (Figure 1 – left) to 2-echelon networks (Figure 1 – right).

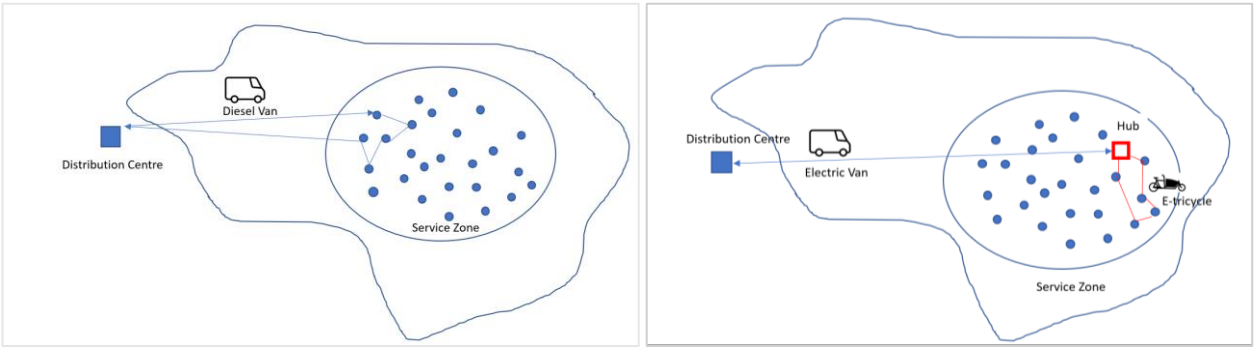


Figure 1. 1-Echelon vs 2- Echelon last-mile distribution networks.

The objective of this model is to support the last-mile sector to compare the impacts of different configuration networks with minimums levels of data granularity. Based on the results (number of vehicles, distances and time) the user can calculate the environmental, operational and financial impact and easily respond to a wide list of "what if" scenarios such as:

- "*what if I move from 1-echelon network using Diesel vans to a 1-echelon using electric vans?"*
- "*what if I move from 1-echelon network using Diesel vans to a 2-echelon using electric vans and e-trikes?"*
- "*what if I install the hub within the city in another location?"*
- *"what if the demand changes"?*
- *….*

Regardless of the scenario, we assume a single distribution Centre (origin) where products are demanded at many destinations within a specified service zone, using a unique type of vehicle with known capacity and within a specific time. We assume all the nodes within the service zone are delivery points.

For estimating the resources needed, the distances and the delivery time per type of vehicle, the following assumptions are required according to the scenario.

1-echelon:

For this scenario (Figure 1 – left) there is only a distribution Centre to serve to the specified service zone. It uses only one type of vehicle “i” with a known capacity. Handling time in the distribution Centre is constant.

Single Distribution Centre

- Surrounding the city boundaries
- Products are demanded at many destinations
- All destinations within specified service zone
- Hanldling time is considered

Assumptions

- 1 echelon network
- Vehicles are homogenous
- Same Capacity
- Workshift constant
- Handling time is constant

2-echelon:

For this scenario (Figure 1 – right), there is a distribution Centre and a hub to serve to the specified service zone. The hub is cross-dock facility where the goods are moved from one vehicle of type “i” supplying the hub from the distribution Centre with direct shipments to another vehicle of type “j” distributing from the hub to the service zone with multiple delivery points. Vehicle type “i” is different to vehicle type “j”. Vehicles type “i" and “j” are known and constant. Handling time in the distribution Centre is constant and handling time in the hub is zero. In the hub we assume the shift time is included in the stop time of vehicle of type “i”. The workshift of the driver of vehicle of type “i” is independent of vehicle type “j” driver’s workshift.

Single Distribution Centre

- Surrounding the city boundaries
- Products are for the hub
- All destinations to the hub
- Handling time is considered

Single crossdock

- Products from one origin
- Products are demanded at many destinations
- All destinations within specified service zone
- Handling time not considered

Assumptions

- 2 echelon network
- Vehicles for 1st leg are homogenous
- Vehicles for 2nd leg are homogenous
- 1st leg and 2ng leg vehicles capacities and characteristics are known
- 1st leg and 2ng leg vehicles are different
- 1st leg and 2ng leg constant and independent
- Handling time for the DC is constant
- Handling time for the hub is zero, shift time include in the stop time of the 1st leg

2 Requirements.

2.1 Software requirements

The language and the software version for running the model is shown in Figure 2.:

```
language       R
version.string R version 4.0.5 (2021-03-31)
nickname       Shake and Throw
```

Figure 2. Model version of the language and software.

The following libraries are needed for obtaining the area and centroid of the area of the shapefile with the boundaries of the service area (optional).

- geojosonio package in R (Cran, github)
- sf package in R (Cran github),

2.2 Input/Outputs

2.2.1 Inputs

The 2-echelon model has the following inputs and its outputs.

Table 1 2-echelon model – Inputs

Inputs	Description
facilitesASIS.csv	Table with the characteristics of the distribution centre outside the city boundaries. For executing the 1-echelon network.
facilitesTOBE.csv	Table with the characteristics of the distribution centre outside the city boundaries and the hub within the service zone. For executing the 2-echelon network.
vehiclesASIS.csv	Table with the characteristics of the vehicle that supplies final consumers within the city boundaries from the distribution centre. For executing the 1-echelon network.
vehiclesTOBE.csv	Table with the characteristics of the vehicle that supplies the hub from the distribution centre to the hub within the service zone and the characteristics of the vehicle that delivers the final consumers within

	the city boundaries from the hub. For executing the 2-echelon network.
services.csv	Table with the with the list of orders to deliver within the delivery or service zone. The same file for 1-echelon and 2-echelon networks.
Config.csv	Configuration file with some parameters and default values. The same file for 1-echelon and 2-echelon networks
serviceZone.shp	Shapefiles with the service zone polygon or boundaries. The same file for 1-echelon and 2-echelon networks.

2.2.2 Outputs

Table 2. 2-echelon model – Outputs

Outputs	Description
testOutputASIS.txt	File with the number of vehicles, distance and time for the executing the 1-echelon network.
testOutputTOBE.txt	File with the number of vehicles, distance and time for the executing the 2-echelon network.

2.3 Paths structure

The structure of the folder for running the 2-echelon model for the 1-echelon and 2-echelon networks is presented in with an example of the city of Madrid in Spain.

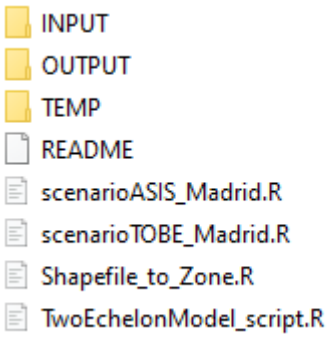


Figure 3. 2-Echelon directory structure.

The INPUT folder contains the files for running the AS IS and to TO BE scenarios. More specifically, it contains the following files:

- facilitiesASIS.csv, facilitiesTOBE.csv
- vehiclesASIS.csv, vehiclesTOBE.csv
- services.csv
- config.csv

The TEMP folder contains the input with the geographic data of the service zone.

- Madrid_Central.shp, Madrid_Central.dbf, Madrid_Central.prj, Madrid_Central.qpj, Madrid_Central.shx

The OUTPUT folder contains the results of running the AS IS and TO BE scenarios. More specifically, it contains the following files:

- testOutputASIS.txt, testOutputTOBE.txt

The scripts for running the scenarios of Madrid are explained in the following section.

3 Model Description

This section describes the different files and scripts present in the model

File name	Location	Description
Shapefile_to_Zone.r	Root folder	Functions for reading geographic data
TwoEchelonModel_script.r	Root folder	Functions for calculating the number of vehicles, distance and times for delivering for one leg (ASIS) and two legs (TOBE) scenarios.
scenarioASIS_Madrid.r	Root folder	Script for executing the scenario ASIS and writing the results in a specific document. The information required is in the INPUT folder and

		the output will be saved in the OUTPUT folder.
scenarioTOBE_Madrid.r	Root folder	Script for executing the scenario TOBE and writing the results in a specific document. The information required is in the INPUT folder and the output will be saved in the OUTPUT folder. (NOT WORKING)

4 Instructions to run the model

4.1 Command line execution of the model

4.1.1 Instructions and commands

- Commands for preparing the libraries and check versions

```
# pre-requirements for geojsonio package in R

sudo apt-get install libprotobuf-dev protobuf-compiler libv8-dev libjq-dev


# pre-requirements for sf package in R

sudo apt install -y libudunits2-0 libudunits2-dev
sudo apt install libgdal-dev
```

Figure 4. Pre-requirements.

- Commands to actually run the model

```
Rscript ~/Documents/zlc_models/2ECHELON/scenarioASIS_Madrid.R
~/Documents/zlc_models/2ECHELON/INPUT/config.csv
~/Documents/zlc_models/2ECHELON/INPUT/services.csv
~/Documents/zlc_models/2ECHELON/INPUT/facilitiesASIS.csv
~/Documents/zlc_models/2ECHELON/INPUT/vehiclesASIS.csv
~/Documents/zlc_models/2ECHELON
```

4.1.2 Arguments

Identification of the arguments in the command line are described below:

args[1] = csv with the config parameters "U:/2ECHELON/INPUT/config.csv"; the file in the INPUT FOLDER

args[2] = csv with the daily services from the operator "U:/2ECHELON/INPUT/services.csv"; see file in the INPUT FOLDER

args[3] = csv with the daily facilities "U:/2ECHELON/INPUT/facilitiesASIS.csv"; see file in the INPUT FOLDER

args[4] = csv with the daily vehicles "U:/2ECHELON/INPUT/vehiclesASIS.csv"; see file in the INPUT FOLDER

args[5] = working directory "U:/2ECHELON"; to create the TEMP folder and the OUTPUT of the model concrete information about the inputs and outputs

4.2 Requirements

4.2.1 Testing requirements

```
# System requirements for rjava R library

sudo apt-get install -y default-jre
sudo apt-get install -y default-jdk
sudo R CMD javareconf

# pre-requirements for geojsonio package in R

sudo apt-get install libprotobuf-dev protobuf-compiler libv8-dev libjq-dev


# pre-requirements for sf package in R

sudo apt install -y libudunits2-0 libudunits2-dev
sudo apt install libgdal-dev
```

4.2.2 Folder INPUT

The inputs of the model see Table 1.

4.2.3 Folder OUTPUT

The outputs of the model see Table 2.

4.2.4 Folder TEMP

Optional folder to unzip the geographic data of the model if available in an open repository as in the case of the example. The content below is after downloading and unzipping the information of the Low Emissions Zone (LEZ) in Madrid from the open repository.

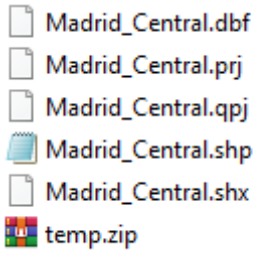


Figure 5. Content of Optional Folder (TEMP) from the open repository of the LEZ of Madrid