Design and use of Quetzal, an open-source Python library for transport modelling.

CHASSERIEAU Quentin
GOIX Robin

SYSTRA

qchasserieau@systra.com
rgoix@systra.com

ABSTRACT

Since its first release in 1991, Python has shifted from a scripting language to versatile tool. In the mid 2000's it gained momentum and became ubiquitous amongst data-oriented applications following the first release of Pandas and Scikit-Learn. The concomitant development of Networkx and Shapely enabled efficient network analysis and geomatics. Consequently, python has been used in parallel of conventional transport modelling software for many years. The aim of the library *Quetzal* is to provide a reliable standalone solution that addresses some limitations of the closed-source modelling software. Quetzal offers tools to help us carry out modelling tasks in the fields of data pre-processing and integrity checks, data-management and transport-modelling.

Data pre-processing is a prerequisite to every modelling study and is often time consuming. The use of an adapted toolbox may considerably reduce this preparation time. Quetzal was designed around the use of open data and provides implementation of state-of-the-art algorithms to process poorly georeferenced GTFS and Open Street Map data. It takes advantage of graph analysis to build a consistent network-conscious timetable from a stop sequence-based, network-naïve timetable and an extraneous road network. To save computer time, it is often worth reducing the size of the input data. Quetzal makes extensive use of geomatics algorithms to build aggregated zonings and station clusters. Consequently, it is possible to target a run time for the model from the early stages of the study and automatically size the input data accordingly. Inconsistencies in the input data can have insidious effects in the results or make the models crash in obscure ways. Quetzal embeds automated consistency checks on the timetables and the road network that spot the most frequent error sources such as non-convex road networks, incomplete stop sequences and namespace collisions (names shared between independent objects).

In addition to its pre-processing toolbox, Quetzal provides a complete algorithm suite to help design a standard four-stage model or more customized ones. There is no real need for a specialized library to achieve the generation step and the Python language offers many options to do so. Once the productions and attractions are estimated, Quetzal offers a doubly constrained distribution based on an impedance matrix that can be provided as an input or generated from a gravitational model. The growth of the OD matrix can later be performed with a Fratar. The levels of service (LOS) of the public transports (time, transfers, fares, …etc.) are estimated with graph algorithms. All of them are variations of the Dijkstra Algorithm. The fastest option is to compute only the best path. The most accurate one is to alternatively break sections of the Public Transport graph and compute the best paths in the broken graphs. The dominated paths obtained in the broken graphs can later be gathered and fed to a nested Logit Model along with the best one to estimate the aggregated levels of service of every origin-destination pair. The car travel time can be estimated with a simple use of the Dijkstra Algorithm if the congestion is of no concern. However, Quetzal also provides an implementation of the Frank Wolfe Algorithm which leads to a car assignment in Wardrop equilibrium. The goal of the public transport and car LOS estimation is to gather for all origin-destination pairs a collection of paths and compute a voyager utility for each of them. The downstream articulation between modal split and assignment is flexible since both use a utility-based nested Logit. If there is no need for loops between the modal split and the LOS estimation, the modal split and the assignment (volume split between the many PT options) can be performed with a single nested logit model. This approach helps enhance the consistency between the modal split and the assignment steps.

In terms of data storage, the models designed with Quetzal relies on light data formats. They can be stored as zipped HDF5 which is hyper fast or as JSONs, which are human readable and easy to track for changes. They can be stored, shared and versioned with Github which makes collaborative projects easier.

As a conclusion, Quetzal can be used as toolbox or as a self-sufficient modelling system. It is a cool open-source tool, particularly relevant in an open-data context.

INTRODUCTION

Since its first release in 1991, Python has shifted from a scripting language to a versatile tool. In the mid 2000's it gained momentum and became ubiquitous amongst data-oriented applications following the first release of Pandas and *Scikit Learn*. The development of *Networkx* and Shapely enabled efficient network analysis and geomatics; consequently, Python has been used in parallel with conventional transport modelling software for many years.

With the increasing ability in Python programming within the new Python programmers in the new generation of transport modelers, the steady development of efficient libraries and the growing interest in open solutions; the opportunity and means for writing an Open Source Python Library for Transport Modelling was presented.

MOTIVATIONS

Closed source software suffer from three major intrinsic downsides: *affordability*, *integrability* and *transparency*. Many counterparts in developing countries cannot *afford* to spend several thousand euros a year to maintain their transport model software. The *integration* of traffic forecasts in a data workflow is often hampered by the Application Programming Interface of the modelling software. The use of a library permits the full power and flexibility of Python to be utilized for common tasks, such as interfacing to a variety of traffic forecasts sources, while specialized transport modelling tasks can be completed within the library. Moreover, despite considerable efforts to write and maintain documentation of closed-source software, the actual algorithms and heuristics sometimes remain blurry, producing a lack of *transparency*, which can result in frustratingly deficient errors without a full stack trace.

The aim of the library *Quetzal* is to provide a reliable standalone solution that addresses these limitations of closed-source modelling software. Relying on a defined object model, *Quetzal* offers tools to help us carry out modelling tasks in the fields of data pre-processing and integrity checks, data-management and transport-modelling.

PRIOR ART

Many theoretical object models are documented in the literature. Closed source software such as VISUM and CUBE were influential for this work, largely due to their adoption across much of the transport sector. However, it was determined that the data specification of the conventional software was not as versatile as the Global Transport Feed Specification (GTFS) which therefore also influenced this work.

Procedures and algorithms for efficient multiple pathfinding are extensively described in the literature (Pajor, 2013), (Bast, 2014), (Bast, 2010): The descriptions provided in (Ortuzar, 2011) and (Bonnel, 2004) were used for the implantation of the Frank-Wolfe algorithm (Frank, 1956). The approaches described by (Dibbelt, 2013) and (Delling, 2014), were also tested during the design phase. However, the single origin-destination solutions did not scale well to the pairwise problem. Otherwise, it was determined that the simplicity of a weighted graph was more convenient in a changing multimodal context. Indeed, when the transit options are represented in more sophisticated structure, it is more challenging to combine them with another mode in an efficient pathfinding strategy. Therefore, the search was narrowed down to the pathfinding algorithms in a naïve weighted graph (where the attributes of the vertices are not used, and the edges carry no information but their weight). In dense networks, the number of paths returned by these multiple pathfinders increase exponentially with the tolerance criteria. In order to enumerate a broad diversity of paths while maintaining low computational time and memory requirements, another multiple pathfinding algorithm was designed.

Finally, an interesting starting point was found in (Poletti, 2016) and (Vuurstaek, 2018) for mapping GTFS data on road networks, the other preparation tasks were performed by combining the algorithms of the four key libraries: *Pandas, Networkx, Scikit-Learn and Shapely.*

TABLE OF CONTENTS

1 OBJECT MODEL

In an Object-Oriented language, an «object» is an entity that can contain data also called «attributes» and code or «methods». The input of the methods can be internal to the object's «attributes» or external, in which case they are called «parameters». The outputs can either be stored as attributes or returned. Quetzal is based on a main object called *StepModel.* It stores the data as specified by the standard and makes the modelling procedures available as methods.

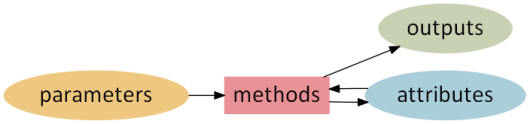


Figure 1 Interactions between methods, parameters, outputs and attributes

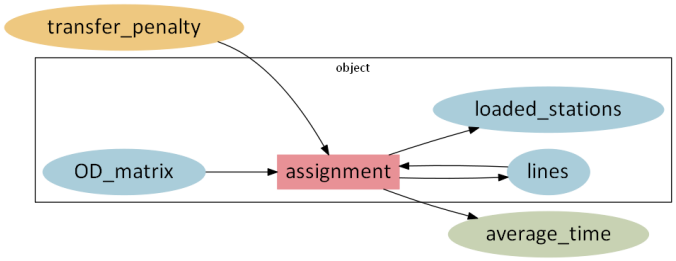


Figure 2 Example of interactions

In the simplified example displayed above, the focus is on the assignment method. This method is based on:

- the Origin Destination volume matrix (attribute)
- the transport lines (attributes)
- a transfer penalty (parameter)

The method creates new data:

- boarding and alighting at the stations (loaded_stations)
- load in the transit lines
- An average trip time

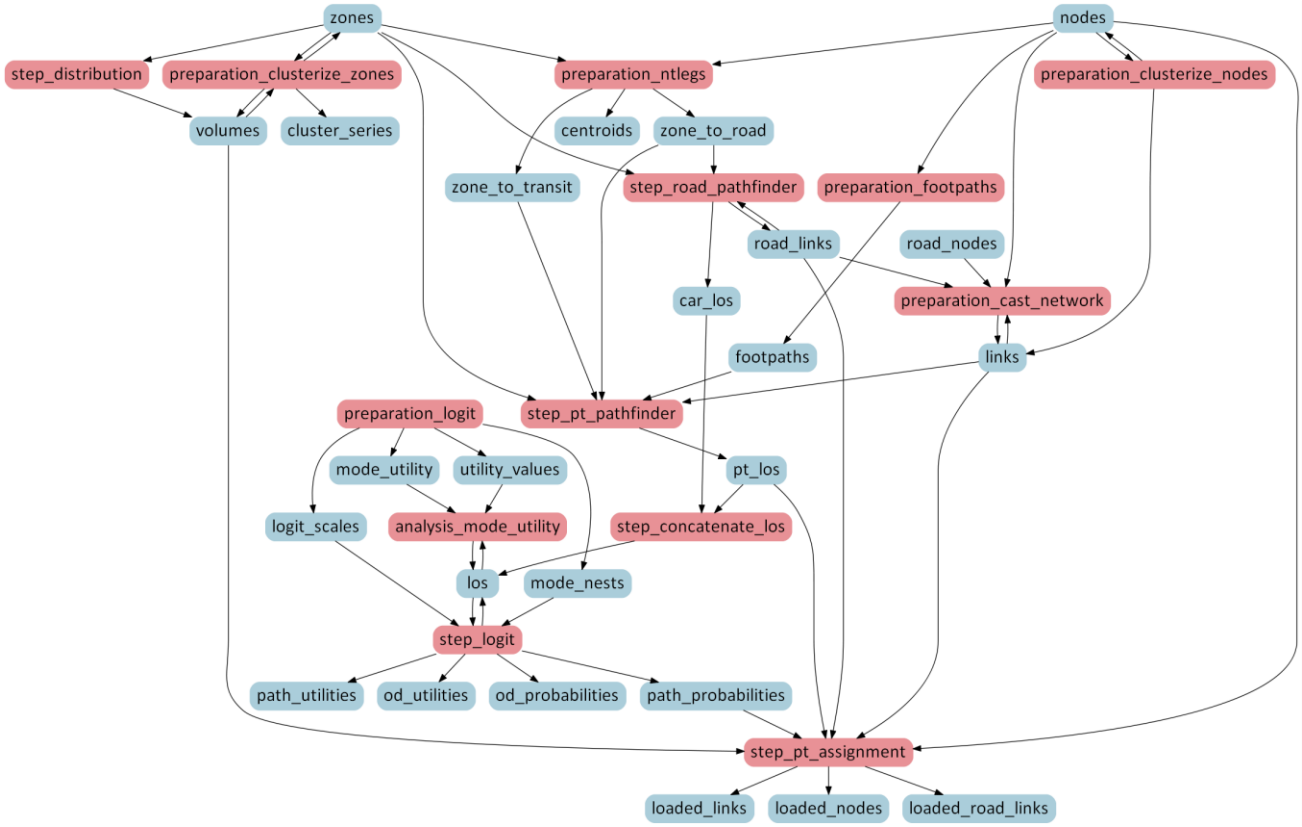


Figure 3 Quetzal Object model

For every method of Quetzal, the required attributes and the expected products are listed. These interactions can be mapped in the form of an oriented graph. When designing a project, one must progress through this network. From a set of available data, attributes are gradually added to the object until the goal – "loaded_links" for example – is reached.

The two main groups of methods are the preparation or pre-processing methods and the step methods. The latter contain the algorithms for the implementation of an actual transport model such as the distribution or assignment steps

2 DATA PREPARATION

Data preparation is a prerequisite to every modelling study. As it is often a time-consuming task, the use of an adapted toolbox may considerably reduce this preparation time. In addition to data cleaning, which is usually project-specific and thus done mostly manually, data preparation often involves data aggregation and network information merging.

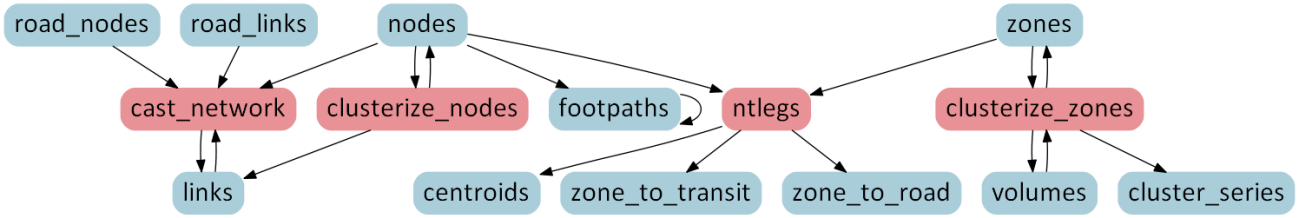


Figure 4 Object Model of the Pre-Processing Toolbox

2.1 COMPLEXITY AND AGGREGATION

To reduce computation time, it is often worth reducing the size of the input data. Making extensive use of existing geomatics algorithms, Quetzal helps building aggregated zones and station clusters. It is therefore possible to target a run time for the model from the early stages of the study and automatically size the input data accordingly.

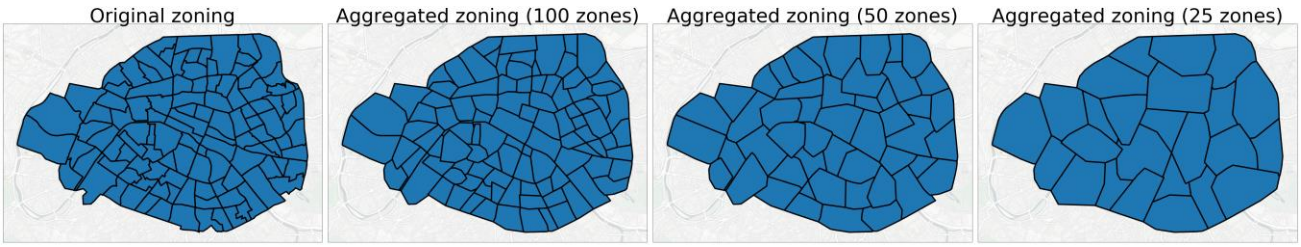
Considering a Quetzal model, with L links, N nodes, and Z zones, its main algorithms have the following complexity:

- Pathfinders are mostly based on The Dijkstra Algorithm. For a single source, the algorithm has a complexity of $O((L+N)*\log N)$. For a the pairwise OD pathfinder the complexity is $O(Z*(L+N)*\log N)$. (Cormen, 2009)
- Many procedures, such as modal split, are applied pairwise on the OD matrices and have an $O(Z^2)$ complexity.

This highlights the advantages of reducing the number of zones and nodes for improving computational complexity.

2.1.1 Zonal aggregation

The initial zoning is a collection of polygons, the aggregation procedure focuses the coordinates of their centroids. They are clustered with a K-means method where K is the number of desired aggregated zones. After computation of the centroid clusters, polygons are merged to form the aggregated zoning.



2.1.2 Stations' clustering

The stations' clustering relies on the same K-means algorithm as the zonal aggregation:

- The original stops are fed to a K-means algorithm
- Each stop is assigned to one of the K clusters
- In the transit links, the original stop is replaced by its cluster

2.2 NETWORK MERGING | CAST_NETWORK

Public transport timetables and road networks are key inputs to many studies, but often come from independent sources; it is common to use the Open Street Map road network, and PT data from a GTFS database. Hence, they are seldom connected, but it is necessary to link them in order to model their physical interactions such as the impact of private car traffic on the commercial speed of road-based public modes.

Each stop can be linked to many road nodes and it is uncommon for the nearest one to be most optimal one. Several solutions have been developed to overcome this issue (Poletti, 2016), (Vuurstaek, 2018) and a similar

one was implemented in Quetzal. In this way, the network and PT data can be maintained independently and efficiently merged when building the model.

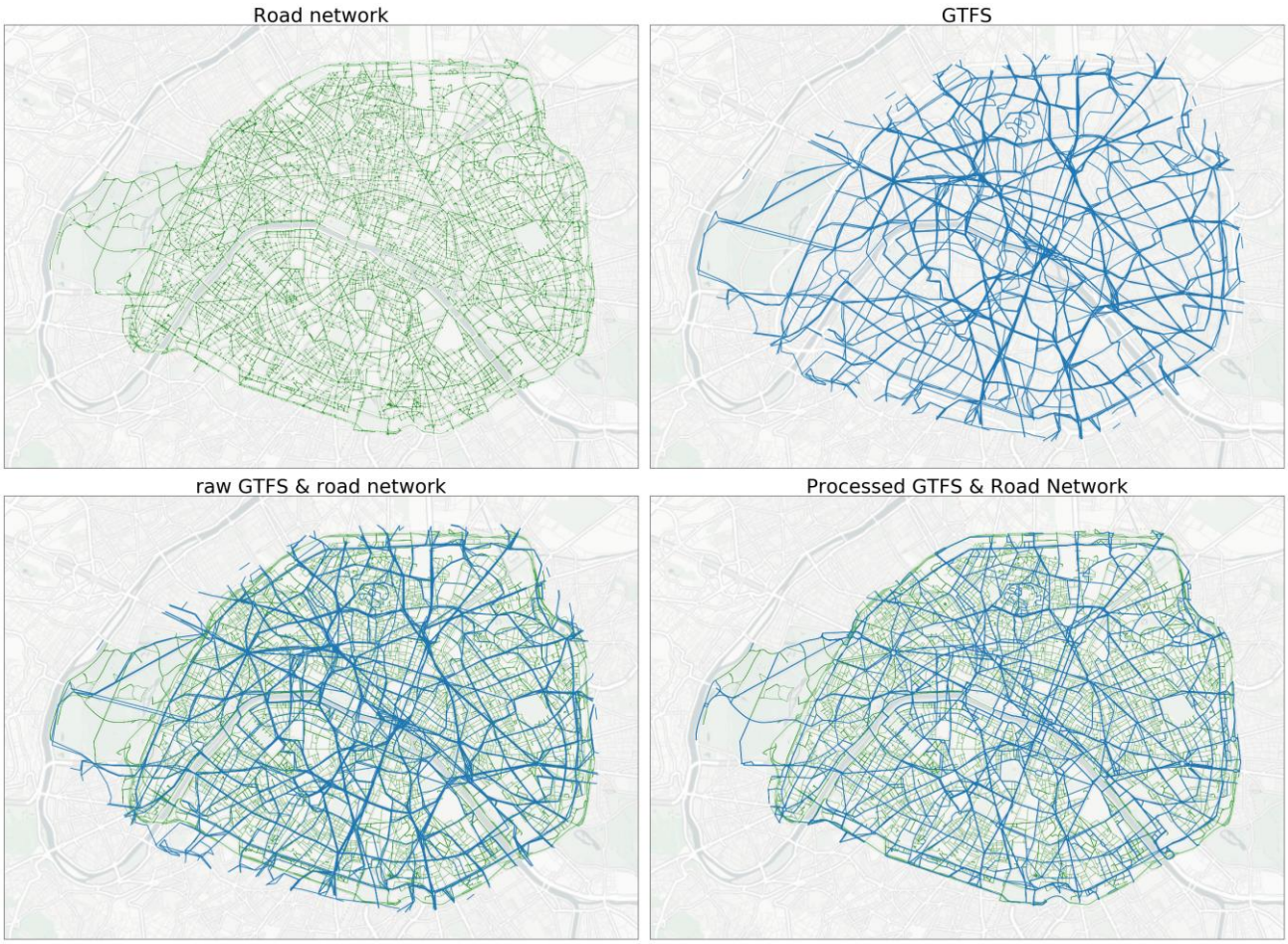


Figure 5 procedure of public transit mapping on road network (GTFS on OpenStreetMap)

2.3 INTEGRITY AND CONSISTENCY

Integrity tests are often left aside in data preparation and consistency problems are usually discovered only when raising a clear error. Such problems are however very common in the complex and highly connected data handled in transport modelling. *Quetzal* embeds tools for automated testing of the most common errors encountered with road networks and timetable data, such as non-convex road networks, incomplete stop sequences and namespace collisions (names shared between independent objects).

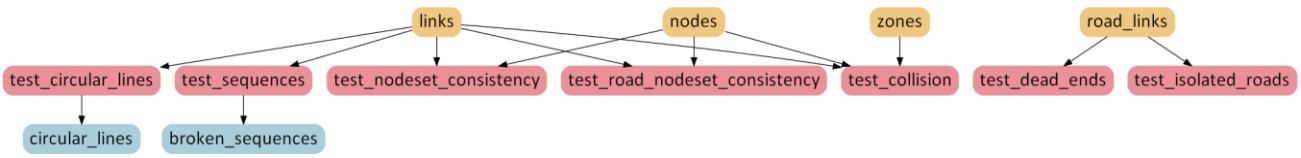


Figure 6 Quetzal *integrity methods*

Quetzal's preparation toolbox provides handy tools to quickly and efficiently map any PT data on any road network, whether it is captured on Open Street Map or collected from another model. The number of zones and stations can be reduced afterward so as to be consistent with the actual geographical accuracy of the input data and the targeted run time of the model. The processed standard data is then used as input of the core procedures of the traffic model.

3 TRANSPORT MODELLING: QUETZAL CORE ALGORITHMS

A traffic model is a sequence of procedures, also called «steps». The widely known «four-step model» is based on the following methods: generation, distribution, modal split and assignment. Although Quetzal is a library that can be used in different kinds of architectures, the four-step model can be used to introduce the library's main methods.

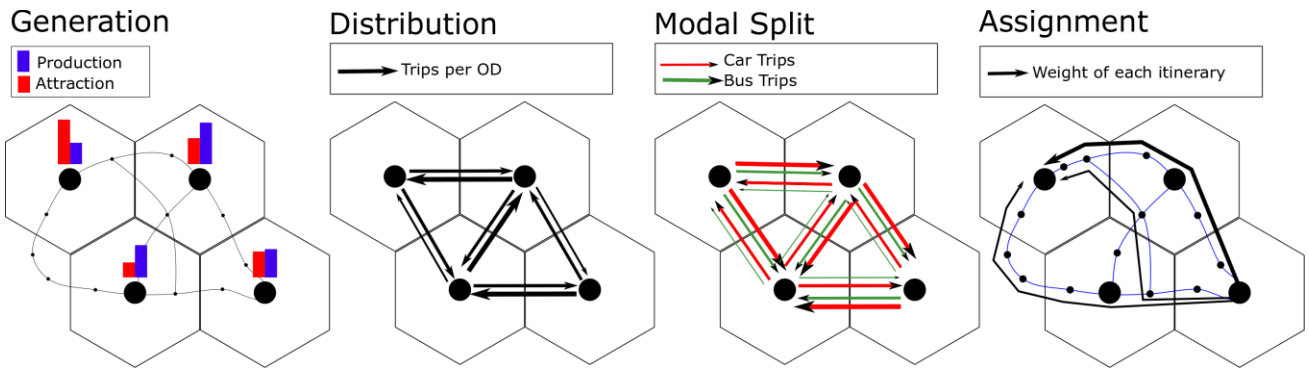


Figure 7 four-step model

- The **generation** step does not require specific methods and is usually performed in raw Python code
- The **distribution** is performed in a dedicated module of Quetzal [distribution]
- The **modal split** is achieved with the logit module [logit]
- The **assignment** is the product of
    - the enumeration of the paths performed by pathfinder algorithms [road/pt_pathfinder]
    - the weighting of paths which is done with the [logit] and [pt_assignment] module;
    - the merge between the weight of the paths and the load on the corresponding OD.

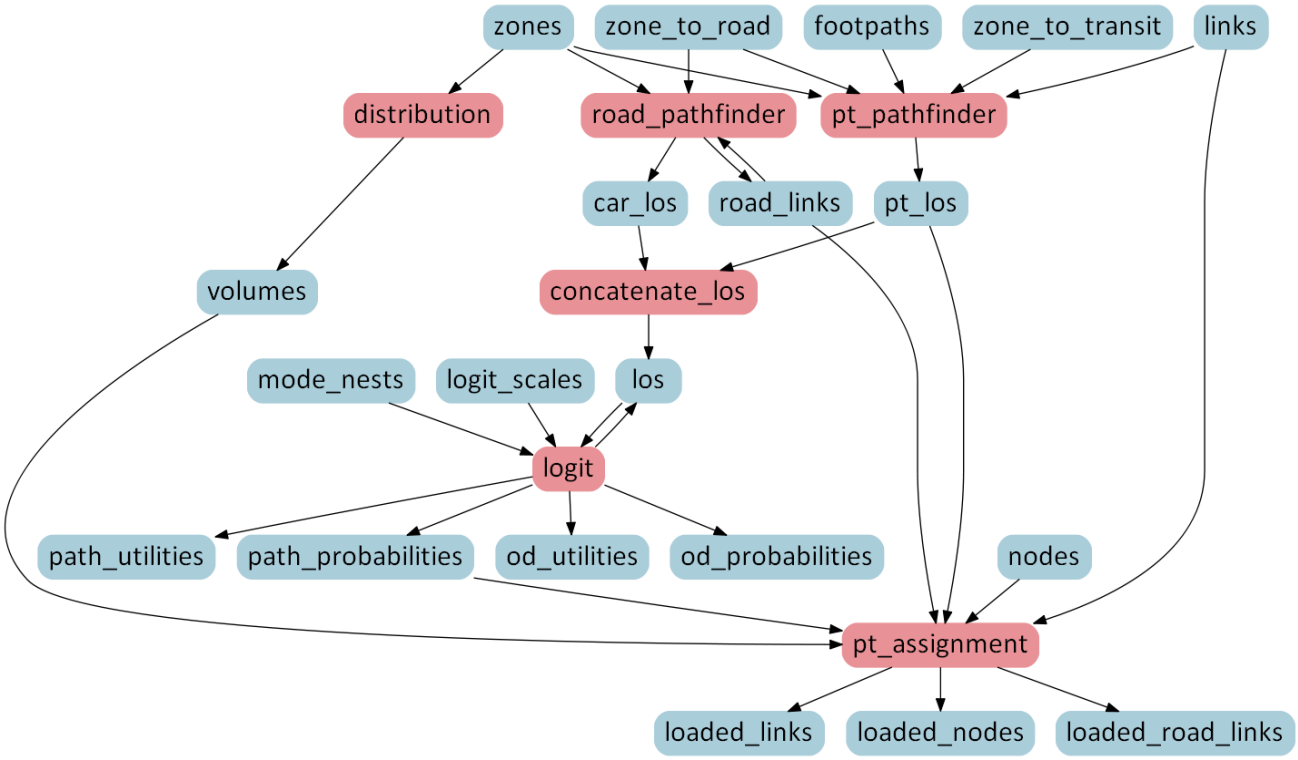


Figure 8 Quetzal Object Model | methods used to build a four-step model

The four-step model is a good way to understand a voyager's sequence of choices, however, the model's implementation is often more complex: loops can be built with the methods, two steps can be integrated in a unique procedure, and the steps themselves are the product of several sub-steps, such as the enumeration and the evaluation in the case of the assignment step. In order to implement a model, we must find a logical sequence of procedures in the object model presented above. When there is no need to loop between the methods, it can be presented as a tree (acyclic directed graph).

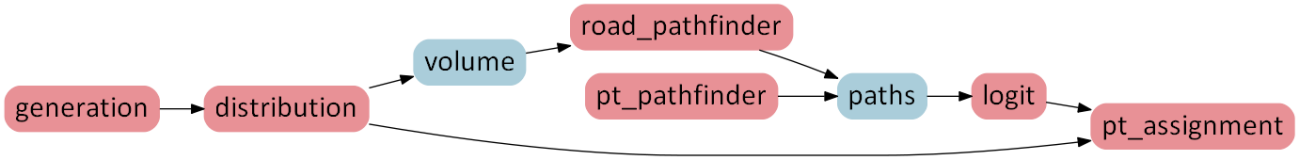


Figure 9 simplified example of an acyclic sequence of procedures

1. the demand matrix is generated by the generation and distribution
2. It is then used as an input in the road pathfinder (and assignment)
3. The public transport pathfinder is simultaneously performed
4. The paths enumerated by the two pathfinders are weighted with a logit
5. The volumes are ultimately assigned to the links by the public transport assignment.

3.1 GENERATION AND DISTRIBUTION STEPS

The generation and distribution are the two first steps of a model. In Quetzal, the results of the generation can be used as an input of the distribution. The production and attractions are used alongside an impedance matrix in a doubly constrained iterative algorithm. The default impedance matrix leads to a gravitational distribution relying on zone-to-zone distance.

3.2 ROAD PATHFINDER AND ASSIGNMENT

The car trip durations can be computed from an exogenous road speed or with the Franck Wolfe algorithm (Frank, 1956) in order to reach Wardrop Equilibrium. The algorithm was chosen for its simplicity, unfortunately its convergence slows down when getting closer to equilibrium on large networks (Ortuzar, 2011, p401). Despite this, it remains a significant improvement compared to an all or nothing assignment.

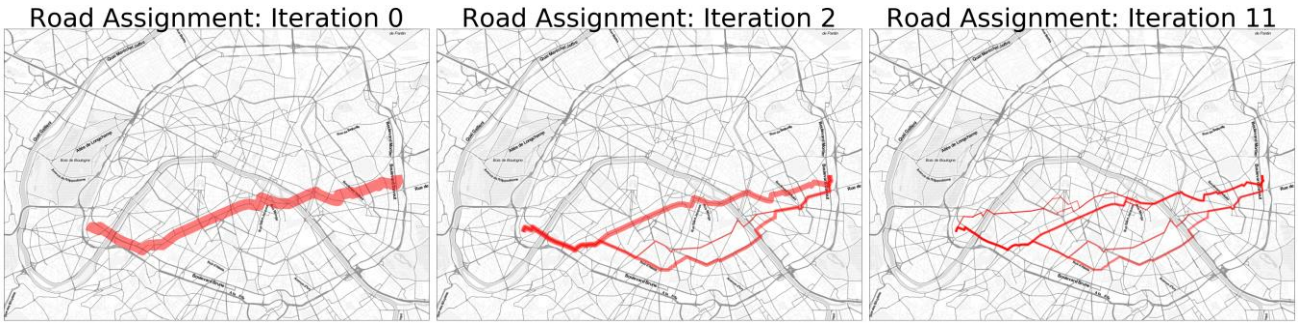


Figure 10 Three iterations of Frank-Wolfe assignment

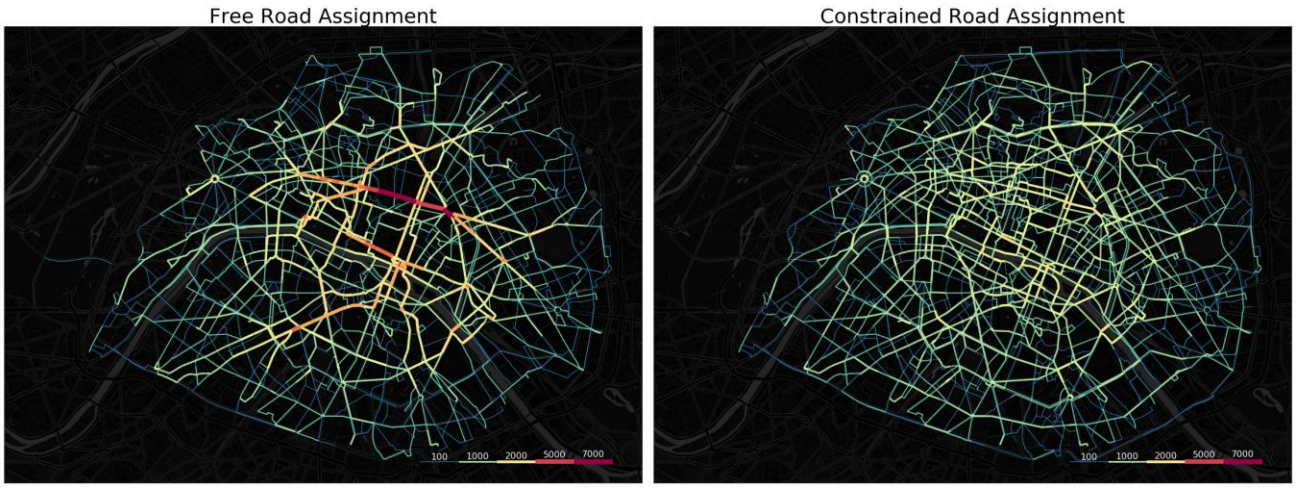


Figure 11 Left: All or Nothing car assignment, Right: Frank-Wolfe car assignment

Given that the public transport is mapped to the road network, the results of the car assignment can be used to update the public transport travel time if the timetables are not reliable. The public transport pathfinder can also be run independently of the car assignment.

3.3 PUBLIC TRANSPORT PATHFINDER

3.3.1 Construction of the public transport graph

A public transport itinerary from one zone to another always includes a walking trip. Therefore, the PT paths are found in a graph that is a superposition of two components: a pedestrian and a transit graph.

3.3.1.1 Pedestrian graph

Pedestrian components connect zone centroids to tangible, physical objects; the road nodes (intersections) and the transit stops (stations). Two approaches were developed to link these vertices: the connectors can either be generated from the road network (walk on road approach) or much simpler, as the crow flies.

3.3.1.1.1 Walk on road (WOR)

It is common to assume that there are sidewalks on most streets, the road network can thus be used as the main component of the pedestrian graph. It is nevertheless necessary to add connectors between the zone centroids and the road nodes (*zone_to_road*) and between the road nodes and the transit stops (*road_to_transit*).

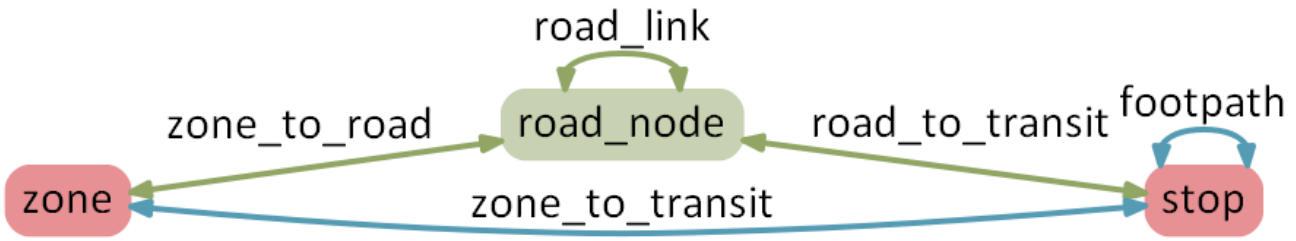


Figure 12 Nodes and edges of the pedestrian graph. Green: Walk on Road, Blue: as the Crow Flies

3.3.1.1.2 As the Crow Flies (ACF)

When the road network is not reliable or when there is no need for such level of detail, the connectors can be generated as the crow flies between the zone centroids and the transit stops (*zone_to_transit*) and between the different stops (*footpaths*). The lesser the number of footpaths, the faster the pathfinder: an efficient

method was implemented to connect the whole set of stops with a reasonable number of connectors using the Voronoi diagram of the stops that defines the connected neighbours.

Figure 13 Map of the pedestrian graph. Green: Walk on Road, Blue: as the Crow Flies

As a conclusion, both ACF and WOR method generate a graph of bidirectional edges between stops and centroids and establish a basis on which the transit components can be added.

3.3.1.2 Transit graph

The transit components are more abstract than the pedestrian ones. Two kind of vertices are involved:

- Physical transit stops (that are shared with the pedestrian graph)
- Transit links: a transit *link* is a vehicle of a given directed route, about to arrive at a given stop.

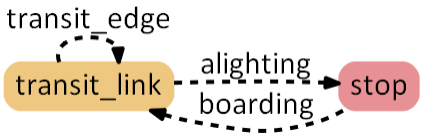


Figure 14 Transit Graph

Considering an ABC directed route: A–>B–>C.
The vertices of the transit component would be:

- Two transit links [AB] and [BC]
- Three stops A, B and C

They would be connected by the following edges:

- A *transit edge* between [AB] and [BC] which represents the action of someone that remains on the vehicle while passing by the station B
- Two boarding links A->[AB] and B->[BC]
- Two alighting links [AB]->B and [BC]->C

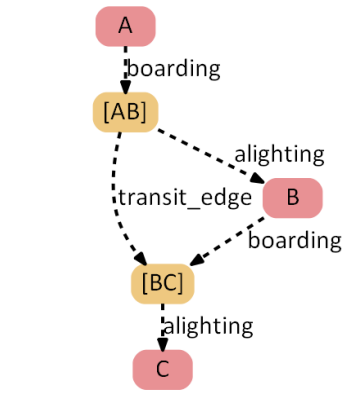


Figure 15 transit graph of the A->B->C route

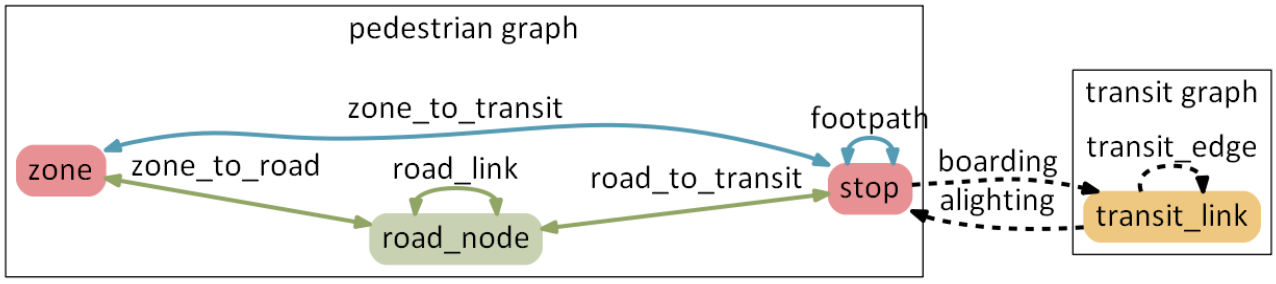


Figure 16 Complete graph for public transport pathfinder

The PT graph is the superposition of the pedestrian graph and the transit graph, connected through the transit stops.

This graph superposition method also allows the modelling of different types of intermodality problems. For instance, the road graph can be added to the PT graph, with connections through park and ride facilities as specific vertices.

3.3.2 Pathfinder algorithms

3.3.2.1 Shortest path

A single source Dijkstra algorithm is wrapped in Quetzal, into a method allowing to compute the shortest path OD matrix by looping on the zones of origin. Between each pair of zones, a path is a represented as a succession of vertices and links, as displayed below. This succession of graph components can lead to the physical itinerary (lines and stations) taken by the traveler.

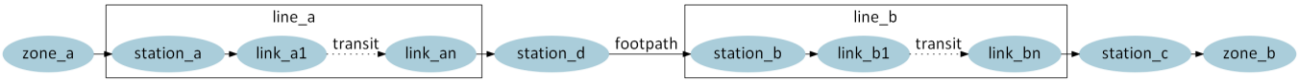


Figure 17 Example public transport path through the transit and pedestrian graph

3.3.2.2 Secondary PT paths

Many economic factors are relevant to the users' choice but cannot be easily formalized in a graph, the zonal fares or the overall perception of a mode for example. Therefore, the path that appears to be the best in the previous algorithm may be worse than another itinerary with –for example– longer transit edges but much cheaper fares.

This highlights the need for wider enumeration, that list not only the shortest path but all the *relevant* ones. The definition of a relevant path will vary from a model to another, depending on the downstream evaluation and weighting process. For example, it is sometimes considered that all paths that have their generalized time less than 10 minutes (or 10%) longer than the best are relevant. However, when these parameters are high (+20 minutes or +20% for example) the number of relevant paths tend to increase exponentially: the enumeration becomes slow. On the other hand, when these parameters are low, dominated paths that could be relevant after the evaluation step tend to be ignored. For example, an itinerary that is 12 minutes longer but twice as cheap could be relevant, yet it would be discarded with a strict 10-minute spread limit.

The origin-destination transit calculators such as CityMapper or Google Maps are used by Million, and the comparison of their results with the previous approach is interesting since these are the options faced by the user in the end. Most of the time, the classic software approach does not reproduce all the diversity of the origin-destination calculators. In CityMapper, the user will find a bus-only path although it is twice slower and more expensive than the subway. Conversely, he will always find an itinerary combining rail and subway even when buses are more efficient. Among the bus only itineraries, you will find not only the best path but also the less optimized one. To achieve this goal for a given origin-destination, a transit calculator such as CityMapper can use an iterative pruning algorithm:

- *Search the best path*
    - *remove a route from the original transit data so that the best path is no longer possible*
- *Search for the best path*
    - *remove another route so that neither the best nor the second best are possible*
- *Search for the best path*
    - *remove all the buses*
- *Search for the best path*
    - *remove the subway and the trains*
- *…*

At each iteration, it is only the best path of a transit dataset that is returned, but many transit datasets are used. In fact, there is no need for a graph when looking for the itineraries of a unique origin-destination with timetables. The Connection Scan Algorithm which is based on an array can do it quickly without having to build all these broken graphs (Dibbelt, 2013).

Unfortunately, traffic forecasts do not only require the dominated itineraries for a unique OD but for all the OD pairs and looping over the origins and destinations with the previous algorithm would be too time consuming. However, it is possible to use an iterative pruning algorithm with a single source pathfinder although the pruning combinations must be determined beforehand. A pruning combination is a list of edges to remove from the graph. It is typically a list of lines and modes such as: [bus_line_3], [tram, subway] or [bus, tram, rail, metro_line_3]. The pseudocode of the algorithm is reproduced below.

1. For each origin
    a. Find the best path to all the destinations
2. For all the pruning combinations
    a. For each *relevant* origin
        i. For all the destinations, find the best path and add it to the list

The concept of relevant origin is a speed up for route combinations: Considering a pruned graph which is lacking one unique. If none of the paths originating from a given zone uses removed route, it is useless to search the shortest path for it will be the same as the one found in the original graph.

Since the shortest path search (bullet 2a) is fast, many combinations can be built to get a diverse set of paths. The choice of the combinations is up to the modeler, depending on the downstream evaluation of the paths and the aim of the model. If the focus is on the subway for example, a wide enumeration of subway paths is required, a lighter method can nevertheless be chosen for the other modes, namely, one bus only itinerary may be enough for the forthcoming modal assignment.

Figure 18 Pruning combinations and resulting itinerary in the PT pathfinder

In this example, the best path uses the subway lines 9 and 11. The algorithm first remove all the combinations of modes in subway, bus, rail and tram. Then focuses on the subway –breaking all the other modes– and removes alternatively the subway lines 9 and 11. Each pruned graph returns a different path. The story is told through the paradigm of an OD, but the pathfinder searches for each relevant origin in every pruned graph.

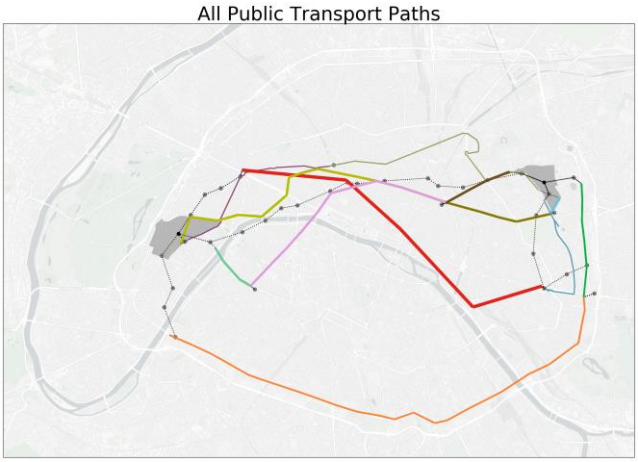


Figure 19 Complete set of PT itineraries for one OD

3.4 Discrete Choice Models

Discrete Choice Models are commonly used in traffic forecasts. Their theoretical framework and practical implementation are extensively described in the literature. The Multinomial Logit Model (MNL) is the simplest and most popular [Domencich and McFadden (1975)]. The Nested Logit Model (NL) can be described as a generalization of the MNL given the NL can collapse to the MNL (Modelling Transport p 238). Both are widely used in the modal split step.

The practical implementation relies on an ascending utility aggregation followed by a descending probability estimation.

3.4.1 Ascending utility aggregation for one Origin-Destination

The ascending aggregation is performed with an expected maximum utility (Ortuzar, 2011, p240)

Considering expected maximum utility (EMU) of a nest j

$$EMU_j = \log \sum_k \exp(\frac{U_k}{\Phi j})$$

The composite utility of the nest j is $Uj = \Phi j * EMU_j$

In this example:

$$U_{rail} = \Phi_{rail} * \log(\exp\left(\frac{U_{tram}}{\Phi_{rail}}\right) + \exp\left(\frac{U_{subway}}{\Phi_{rail}}\right))$$

$$U_{pt} = \Phi_{pt} * \log(\exp\left(\frac{U_{rail}}{\Phi_{pt}}\right) + \exp\left(\frac{U_{bus}}{\Phi_{pt}}\right))$$

The composite aggregation of the upper nest is the utility of the origin-destination

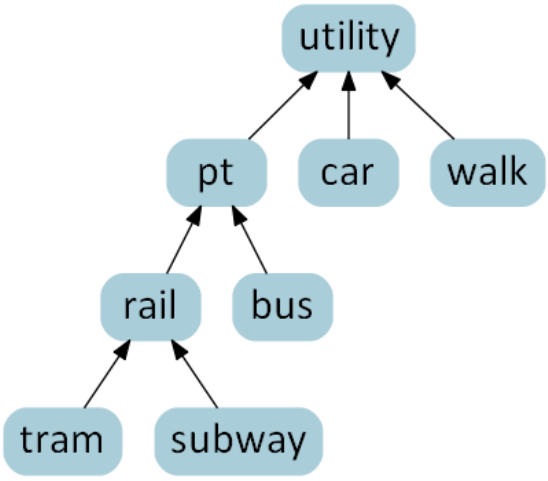


Figure 20 Example of nested structure

3.4.2 Descending probability estimation for one Origin-Destination

The descending probability estimation is a sequence of Multinomial Logit Models (MNL). Considering the example described above, its formulation is the following:

$$P_{tram} = \frac{\exp(U_{tram})}{\exp(U_{tram})+\exp(U_{subway})} * P_{rail} \text{ and } P_{rail} = \frac{\exp(U_{rail})}{\exp(U_{bus})+\exp(U_{rail})} * P_{pt} \ldots$$

Where P_{tram} is the probability of the tram U_{tram} is its utility.

The sum of all the probabilities of the upper nest is equal to 1.

3.4.3 Overall practical implementation in the modal split step

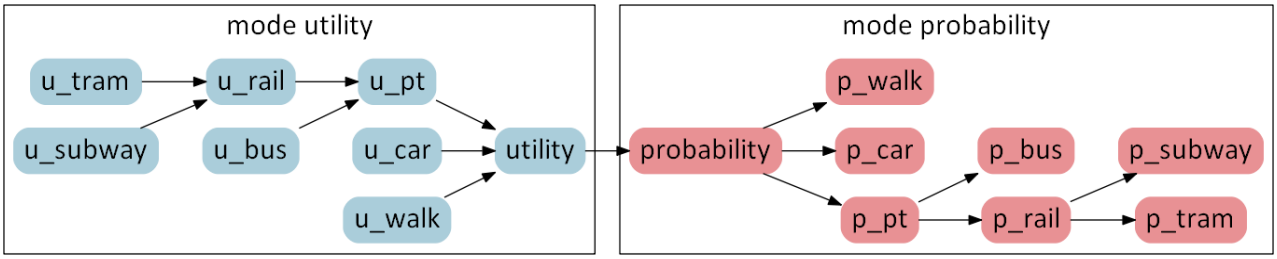


Figure 21 Nested Logit practical implementation

The utility aggregation and probability estimation are performed for each OD. The utility of the lower modes (tram, subway, bus, car and walk in this example) are calculated from the utility of the paths.

3.4.4 Path utility

After the PT and car pathfinder, a set of paths is available for each OD. An evaluation procedure is run to assess their economic *utility.* In addition to the price and the time, all the attributes of the path, such as the mode and the number of transfers can contributes to the utility.

For example, the linear price-time model uses the following formulation for a path *p*:

$$U = -Time(p) * ValueOfTime - Cost(p)$$

3.4.5 Extension of the nested logit to the paths

In the classical four step model, the assignment and modal split are independent. The estimation of a mode composite utility from the different paths can be done by several means. Composite utilities such as weighted average or logsums are commonly used.

When the aggregation of the paths' utilities is not consistent with the modal split discrete choice model, paradoxical behaviors can occur. (Ortuzar, 2011, p238)

A level can be added to the modes' nest structure in order to achieve the paths' utilities aggregation. The probability of the PT paths is estimated afterward with the descending method. The utility of the car paths is not updated; at the end of the car assignment, all the paths of an OD approximately the same duration but they do not have the same probability. Updating the probabilities would change the whole car assignment.

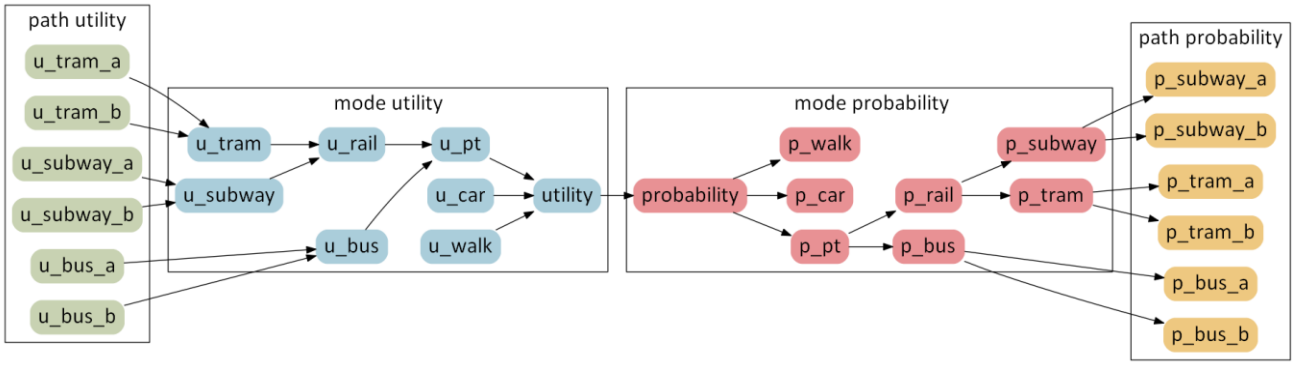


Figure 22 Extended nested logit; modal split and weighting of the PT paths

4 USE OF QUETZAL

Quetzal is a Python library and must therefore be used through programming scripts. A *Quetzal* model is usually an organized set of scripts that read and export different data files. Each script imports parts of the *Quetzal* library to use its different resources. Although any text editor or Integrated Development Environment can be used to develop these scripts, a convenient option is Jupyter Notebook (Kluyver, 2016) as it can flexibly combine live code, narrative text and data visualization while allowing to run separated code cells.

Quetzal contains two main built-in methods to export a model content:

- A slow export to human-readable files: all model components are stored as text files in JSON or GEOJSON formats, that can be opened in a text editor or a GIS software for geographic data;
- A fast export to an HDF5 binary file automatically zipped to obtain light models

Scripts reading and exporting data files can conveniently be parallelized to optimize the use of computer resources: if computer RAM is large enough several scenarios or independent model steps can be launched in parallel to shorten the model run duration.

Another interesting benefit coming from the use of scripts as main components of a model is the availability of powerful control version systems to save and track changes in a model and to assist in merging of contributions from different sources, allowing simultaneous work of several engineers on the same model. Solutions as GitHub or GitLab that enclose the control version system Git (Blischak, 2016) on an online website allow to easily share models and access them from anywhere.

Quetzal source code and documentation is available online:

- Documentation: https://systragroup.github.io/quetzal
- Sources: https://github.com/systragroup/quetzal

5 PAST AND ONGOING PROJECTS WITH QUETZAL

Figure 23 Map of Quetzal models around the word

Quetzal was used to implement a model in the following cities and countries.

- Paris (training model)
- Limoges (France)
- Clermont Ferrand (France)
- Villanueva (Guatemala)
- Santo Domingo (Dominican Republic
- Douala (Cameroun)
- Kinshasa (Congo)
- Cairo (Egypt)
- Mansoura (Egypt)
- Viet Nam (national)
- Manila (Philippines)

CONCLUSION

Quetzal introduces an object model and data specification that encompass a broad range of applications. It thereby facilitates the data management on a project scale, from the preparation to the core algorithms of a traffic forecast model.

It makes available algorithms for each step of the modelling. Proven methods such as the doubly constrained distribution and the Frank-Wolfe algorithm are used alongside new algorithms, designed to model intermodality and widen the set of possible model structures. The pairwise iterative pruning algorithm and the extended nested logit combine well in a redefined modal-split-assignment sequence.

Its' BSD compatible license is a distinctive feature which makes modelling more affordable and transparent. Hopefully, this asset will encourage an increasing number of transport modelers to join the community and contribute to the project.

REFERENCES

Bast, H., & Storandt, S. (2014, November). Frequency-based search for public transit. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (pp. 13-22). ACM.

Bast, H., Carlsson, E., Eigenwillig, A., Geisberger, R., Harrelson, C., Raychev, V., & Viger, F. (2010, September). Fast routing in very large public transportation networks using transfer patterns. In *European Symposium on Algorithms* (pp. 290-301). Springer, Berlin, Heidelberg.

Blischak, J. D., Davenport, E. R., & Wilson, G. (2016). A quick introduction to version control with Git and GitHub. *PLoS computational biology*, 12(1), e1004668.

Bonnel, P. (2004). Prévoir la demande de transport (No. halshs-00077292).

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009*). Introduction to algorithms*. MIT press.

de Dios Ortuzar, J., & Willumsen, L. G. (2011). *Modelling transport*. John wiley & sons.

Delling, D., Pajor, T., & Werneck, R. F. (2014). Round-based public transit routing. *Transportation Science*, 49(3), 591-604.

Dibbelt, J., Pajor, T., Strasser, B., & Wagner, D. (2013, June). Intriguingly simple and fast transit routing. In *International Symposium on Experimental Algorithms* (pp. 43-54). Springer, Berlin, Heidelberg.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1), 269-271.

Domencich, T. A., & McFadden, D. (1975). *Urban travel demand-a behavioral analysis*

Frank, M., & Wolfe, P. (1956). An algorithm for quadratic programming. *Naval research logistics quarterly,* 3(1-2), 95-110.

Pajor, T. (2013*). Algorithm Engineering for Realistic Journey Planning in Transportation Networks* (Doctoral dissertation, Karlsruhe Institute of Technology).

Poletti, F. L. A. V. I. O. (2016*). Public Transit Mapping on Multi-Modal Networks in MATSim*. Master, IVT, ETH Zurich.

Vuurstaek, J., Cich, G., Knapen, L., Ectors, W., Bellemans, T., & Janssens, D. (2018). GTFS bus stop mapping to the OSM network*. Future Generation Computer Systems*.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., ... & Ivanov, P. (2016, May). Jupyter Notebooks-a publishing format for reproducible computational workflows. In ELPUB (pp. 87-90).