

MATRIX ESTIMATION FROM TRAFFIC COUNTS: INTEGRATING THE PROPORTIONAL PATH AVERAGES ALGORITHM INTO TRAFFIC SIMULATION SOFTWARE

J deV OBERHOLZER

Hatch Africa (Pty) Ltd, 25 Richefond Circle, Ridgeside Office Park, Umhlanga 4319
Tel: 031 936-9400; Email: dev.oberholzer@hatch.com

ABSTRACT

Traffic simulation software packages that attempt to match assigned volumes with traffic counts rely on tried and tested demand matrix adjustment techniques. Several different methods are commonly used, most of which successively adjust the input demand matrices within iterative traffic assignment procedures. This paper follows on from a previous 2021 SATC paper wherein the author described an alternative approach based on proportional path averages, illustrated by two practical case studies. Since the algorithm could be applied independently of the assignment technique, it was implemented previously using an Excel spreadsheet containing a set of VBA macros. The algorithm requires as input only three data sets: a demand prior matrix, a table of link and/or turn traffic counts, and the assignment volumes along all Origin-Destination paths. This paper describes the direct integration of the algorithm into Emme/4 using the built-in Python scripting tools that access the Emme/4 Application Programming Interface. Three case studies illustrate the performance of the algorithm versus the standard Emme/4 demand adjustment module. The paper concludes with summary results illustrating proof of concept. Three recommendations identifying further research are also included.

1. INTRODUCTION

1.1 Background

Adjustment of Origin Destination (OD) demand matrices using traffic counts is a complex topic that has enjoyed the attention of many analysts during the last five decades, as summarised concisely by others (Abrahamsson, 1998, Noriega and Florian, 2009 and Lindström and Persson, 2018).

The strategic transport software package Emme/4 (INRO, 2020) incorporates its own matrix adjustment variant based on the Gradient Method. Originally developed in Emme/2 for the uncongested, single traffic class assignment (Spiess, 1987) and subsequently adapted for equilibrium traffic assignments (Spiess, 1990), the method has been in general use for more than three decades. It was extended in Emme/3 to handle multi-class equilibrium traffic assignments (Noriega and Florian, 2007) and later enhanced via a weight factor to increase the importance of the original demand relative to the flow comparison (Noriega and Florian, 2009). The mathematics, however, are quite complicated and somewhat daunting for non-mathematicians to decipher.

As explained in the previous paper on this topic (Oberholzer, 2021), matrix adjustment based on proportional path averages (PPA) is an easily understood concept that incorporates two simple steps applied iteratively to all modelled path volumes passing

through a given set of traffic counts. The PPA technique differs from the Emme/4 approach in that the matrix adjustment is applied iteratively on a given set of assignment paths within an inside loop, rather than singly within the outer assignment loop. Fewer outer assignment iterations are required, as the bulk of the demand adjustment is performed iteratively within the inner loop using only the input demand matrix, the link and turn counts, and the equilibrium assignment paths.

1.2 Objectives of This Paper

This paper describes the development of a custom written Python script to implement the PPA algorithm as an Emme/4 Modeller Tool, complete with a simple user interface that is commensurate with the standard Emme/4 Modeller Toolbox (INRO, 2020). Given that the Emme Standard Toolbox already has a tried and tested module that achieves the same aim, the paper also compares the performance of the two methods, using the same input data.

The overall objectives are thus to:

- Confirm the PPA proof of concept, when applied within the context of commercial transportation software as applied to project case studies.
- Compare the performance of the PPA algorithm with its Emme/4 counterpart, in terms of accuracy and execution speed.
- Conclude on the viability of incorporating the PPA algorithm into mainstream commercial transportation software.
- Identify the way forward for further research.

As the PPA algorithm is still in its infancy, the comparative analysis is limited to the assignment and adjustment of a single traffic demand matrix.

2. PROCESS OUTLINE

2.1 The Proportional Path Averages Algorithm

As described in the previous paper on this topic (Oberholzer, 2021), the PPA algorithm is formulated simply as follows:

- Step 1: For each turn or link volume where a count is available, adjust all modelled origin to destination (OD) path volumes passing through the count pro rata, to match the total modelled volume with the count, as illustrated in Figure 1.



Figure 1: Adjusting OD Path Movements through Traffic Count Stations

- Step 2: For each unique OD movement path that passes through one or more count locations, adjust the OD path volume to match the average of all partial modelled volumes along that path, as illustrated in Figure 2.

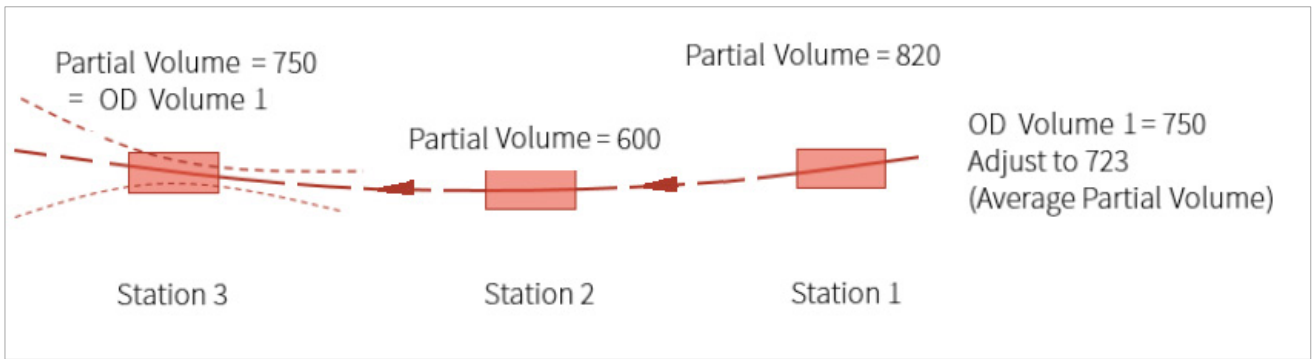


Figure 2: Adjusting OD Trips using Averaged Partial Volumes along OD Paths

Since the path volume should remain constant, the average path volume smooths out any differences in partial path count volumes, in a “best-fit” manner. Each step will change the partial OD volumes, so steps 1 and 2 should be repeated until no further discernible changes occur. Note that the iterative adjustments are applied separately from the assignment.

The algorithm is analogous to two-dimensional matrix balancing, where OD path volumes represent individual matrix cells, with the link or turn count number as the row index and the OD path number as the column index. Table 1 illustrates a fragment of one iteration of the analogous balancing operations, using the example values from Figure 1 and Figure 2.

Table 1: Two-Dimensional Matrix Balancing Analogy

Count Number	Path Number				Total	Count
	1	2	3	4		
1	... 820 823
2	... 600 823
3	500 750 823	200 300	300 450	...	1000 1500	1500
Path Average	823		

Colour coding, as per successive adjustments in yellow shaded cell, row 3, column 1:
 500 = Initial value
 750 = Step 1: Adjust path volumes pro-rata to match count, by row
 823 = Step 2: Adjust path volumes to match the path average, by column

The iterations should continue until the difference between successive root mean square errors, calculated by comparing the counts with the modelled values, is negligible, or the number of iterations exceed a suitable cut-off threshold, typically 200.

Although the two-dimensional matrix balancing analogy helps to explain the concept, processing of such a count/path table using matrix balancing techniques would be possible, but not practical. It is more efficient to handle the iterative calculations using Python lists, in conjunction with Python add-on data processing libraries such as NumPy (NumPy, 2022) and Pandas (Pandas, 2022).

In addition, to simplify data handling, housekeeping tasks must ensure that:

- The prior trip matrix is imported from the Emme databank.
- Link and turn traffic counts are imported from Emme attributes into a single list wherein each link or turn count is inserted as a new row with a count ID, the node numbers identifying the link or turn in the sequence: i-node, j-node, k-node, and the count volume. For link counts, the k-node will be zero.
- Path volumes are imported from Emme output paths, directly after assignment of the prior trip matrix onto the road network.
- The root mean square error (RMSE) is calculated and compared with the RMSE of the previous iteration, directly after each iterative adjustment. The RMSE can be considered as the simplified objective function which needs to be minimised.
- On convergence, the adjusted partial OD path volumes are consolidated into the adjusted trip matrix with the necessary changes to all affected OD cells (and therefore also to the overall OD matrix totals). The adjusted matrix must then be exported back to the Emme databank, after termination of the outer assignment iterations. This step completes the adjustment of the OD matrix, fulfilling the purpose of the algorithm.

2.2 The Comparative Emme/4 Algorithm

In essence, the Emme/4 algorithm, as encapsulated within the Modeller Module “Multiclass Traffic Demand Adjustment” (INRO, 2020), comprises seven steps, summarised from the relevant research (Noriega and Florian, 2009) as follows:

1. Multi-class equilibrium assignment of the demand matrices.
2. Calculation of the link derivatives and the objective function.
3. Multi-class equilibrium assignment, with path analysis, to calculate the gradient matrices, followed by the addition of the demand term into the objective function.
4. Multi-class equilibrium assignment to obtain the descent direction.
5. Update the demand matrices.
6. Update the iteration counter. If less than the maximum number of iterations, return to step 1.
7. Otherwise terminate.

As can be seen, each iteration requires 3 equilibrium assignments. For large, congested networks, module run-times could become excessive. Normally, one would use the fastest multi-class traffic assignment tool available within Emme, i.e., either the Standard Equilibrium or Second Order Linear Approximation (SOLA) algorithms.

3. IMPLEMENTATION OF THE PPA ALGORITHM AS AN EMME/4 MODELLER TOOL

3.1 Flow Chart and User Interface

The PPA algorithm is dependent on all OD paths and volumes being readily available directly after the traffic assignment. Emme’s path-based assignment is the only feasible equilibrium assignment technique where all assignment paths are stored in separate path files per assignment class, for later path-based analysis. Although Emme/4’s SOLA traffic assignment tool also allows path analyses, it does not explicitly generate external path files that can be interrogated separately. Consequently, the PPA technique is currently limited to using the Emme path-based assignment technique.

Figure 3 illustrates the overall data flow within the PPA algorithm, when converted into an Emme/4 Modeller Tool, inclusive of required housekeeping tasks. The colour coding indicates the three separate stages.

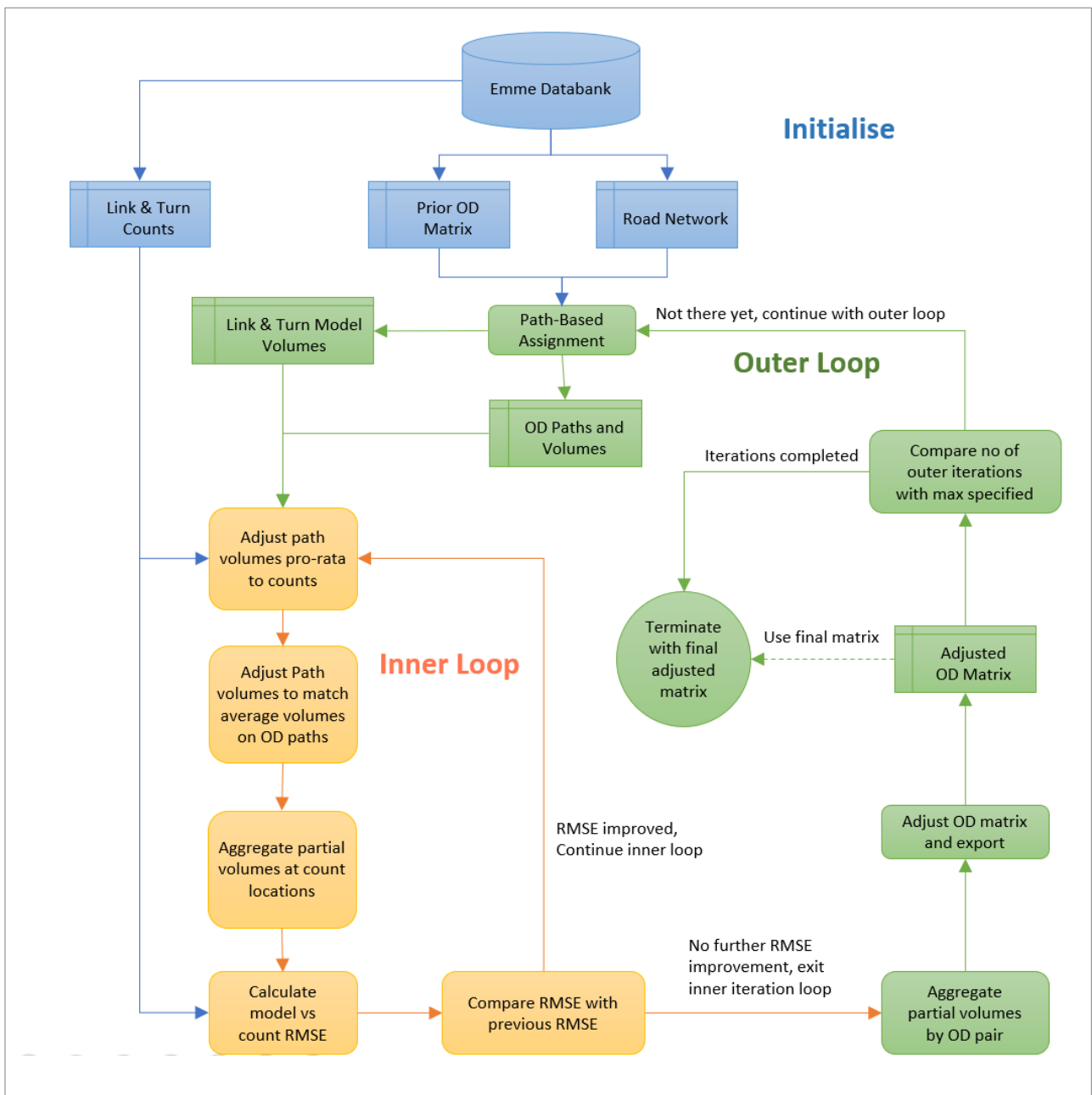


Figure 3: PPA Flow Chart Implemented as Emme Modeller Tool

In addition, the standard Emme demand adjustment module does not allow path-based assignment to be used as the assignment specification, which complicates, but does not negate, one-on-one comparison between the two techniques.

To assist with extracting data from the path files, the Emme Application Programming Interface (API) reference documentation provides example python coding.

The PPA user interface is commensurate with the comparative Emme/4 Multiclass Demand Adjustment Tool in appearance, with the complexity of the algorithm hidden by the user interface, as illustrated in Figure 4.

PPA

MxAdjPPA. (rev 1.02 2022-01-14)

Matrix Adjustment - Proportional Path Averages MxAdjPPA

Run in Scenario
 ↕
 Matrix adjustment will be done within the selected scenario

Prior Matrix
 ↕
 Prior matrix must exist

Adjusted Matrix
 ↕
 Adjusted matrix must exist, but not same as prior

Link count attribute
 ↕
 Previously imported

Turn count attribute
 ↕
 Previously imported

No of Outer Loop iterations

 Assignment outer loop iterations. Note that the inner PPA loops terminate automatically on RMSE

▶ Run

Figure 4: PPA Emme Modeller Tool User Interface

As is the case for the standard Emme Demand Adjustment, the number of assignment iterations is specified externally. The value selected will be based on a judgement decision relating to the overall accuracy of the demand adjustment, after assessment of convergence charts and model versus count scatterplots.

3.2 Emme/4 Modeller and Python

Python, the primary Emme/4 Modeller development tool, is a cross-platform, open-source programming language that handles both object-oriented and function-based procedural coding in a hybrid interpretive/compiled manner. It was originally devised in 1991 by Guido van Rossum (Van Rossum, 1993), a Dutch programmer. Ongoing Python development is managed by the Python Software Foundation, a non-profit organisation. Python distributions, source code, documentation, help-files, and examples are freely available as downloads from the official Python website (Python, 2022). Third-party modules, numbering in their thousands, are also available from a separate website (Python Package Index, 2022).

To standardise development of custom-built Modeller Tools, Emme/4 relies on Python 2.7, installed by default into the folder EMMEPATH/Python27. The usual approach for developing Emme/4 Modeller Tools consists of creating a new module in the Project Toolbox by specifying the Python text file containing the Python source code. From the

Emme/4 desktop, one can also access the Modeller Shell, an integrated Python environment for quick ad hoc scripting that allows direct access to the Emme Modeller API. The Modeller Shell operates within either standard Python or the interactive iPython programming environment (iPython, 2022).

The Emme APIs are grouped into five main categories: Database API, Network API, Matrix API, Desktop API and Data Table API which collectively provide access to all the Emme Standard Toolbox modules, as well as all built-in and customised Emme Desktop worksheets, tables and views.

3.3 Emme Optimisation

In computer science, time complexity (Sipser, M, 2006, p. 247) is defined as a measure of the computer time taken to run a process or algorithm. OD demand datasets are stored in 2-dimensional square matrices with dimensions $n \times n$, where n = number of zones. Repetitive processes that involve OD matrices exhibit *quadratic time complexity*, i.e., computer runtimes are approximately proportional to n^2 .

Network size also influences program execution times, but the dominant contributor is the number of zones. In congested networks, multiple paths between OD pairs are possible, increasing the runtimes for matrix-based operations. Table 2 compares Emme/4 traffic assignment runtimes for the three case studies described in Chapter 4.

Table 2: Emme Traffic Assignments – Computer Runtime Comparison

Description	Computer Runtimes (seconds)		
	Umhlathuze	Winnipeg	eThekwini
Number of zones	80	154	646
Path-based traffic assignment:			
Cold start	<1	2	46
Warm start, using existing paths	<1	<1	10 to 20
SOLA traffic assignment	<1	<1	10
Standard equilibrium traffic assignment	<1	<1	10

Clearly, most of the excessive computer runtime for the Emme/4 path-based assignment can be ascribed to the generation of valid OD paths stored in the path file during a cold start. Since this standard Emme/4 Modeller Tool is an integral component of the PPA algorithm, specifying a warm start for the second and subsequent path-based assignments contained in the outer loop will reduce the overall runtime somewhat, a feature that will be evident only when dealing with large transport networks.

3.4 Python Code Optimisation

For PPA adjustment in its simplest form, the creation of a master database table as quickly as possible is essential. The master table cross-references the OD adjusted fractional path volumes to both path IDs and count IDs, emulating the count/path matrix analogy described earlier. To facilitate the associated housekeeping tasks, additional columns containing the turn definition (nodes I-J-K), the total count volume, and the fractional path volume from the previous iteration, also need to be included in the master table.

Table 3 illustrates a data fragment of the PPA master table, as generated during the final PPA adjustment iteration in the eThekwini case study described in Chapter 4.

Table 3: PPA Master Table – Data Fragment

Orig Zone	Dest Zone	Path ID	Count ID	Nodes I-J-K	Total Count	Previous Volume	Adjusted Volume
...
...
303	49	100558	131	2085-5361-0	2070	0.90	0.90
303	49	100558	170	5398-5396-0	2106	0.90	0.90
303	49	100558	199	7107-7103-0	1659	0.90	0.90
303	50	100559	131	2085-5361-0	2070	1.35	1.35
303	50	100559	170	5398-5396-0	2106	1.35	1.35
303	50	100559	199	7107-7103-0	1659	1.35	1.35
...
...

Since only link counts were considered for this case study, all K-nodes are zero. The full table contains 982 938 rows, each representing a unique combination of a valid OD path with a link count, i.e., all non-zero OD path volumes passing through the specified link counts. This type of structure allows adjustment of fractional volumes in aggregated groups by either the count ID or the path ID.

The multiple occurrences of individual OD paths, as well as the multiple paths going through individual count IDs in the data fragment shown in Table 3, reflect only a very small portion of the related path/count data. In the full table, 12 734 OD paths pass through Count ID 131, each contributing a fraction to the total modelled volume at the count location. The previous volume and the adjusted volume total 2 054 and 2 062 respectively, when aggregated by Count ID 131.

Inclusion of the previously adjusted fractional volume allows quick calculation of the adjustment factors to be applied to all fractional paths passing through each count, as well as the averaging of fractional OD path volumes passing through multiple counts.

In addition to minimising the effects of quadratic time complexity, optimal Python coding is essential for reducing computer runtimes as far as possible. To this end, Python is very efficient when dealing with loops in iterable lists.

For example, the Emme API documentation recommends using the following code snippet to import path data from the binary path file generated by a path-based assignment:

```
file_name = os.path.join(emmebank_dir, "PATHS_s3001_c1")
path_data = _pathfile.PathFile(file_name, scenario, network)
for orig, dest, volumes, paths in path_data:
    for vol, path in _izip(volumes, paths):
        for link in path:
```

Since the code snippet processes iterable Python nested lists, it executes very quickly, when running in the Modeller Shell or the custom PPA Modeller Tool. For the eThekwini case study, adding a timer and a counter to the code snippet resulted in a runtime of 3.6 seconds to iterate through 17.93 million OD/path/link combinations, illustrating Python efficiency when processing nested lists using loops.

However, an immediate timing problem emerges when going one step further to check whether a link is present in the counts table. The typical Python list statement: “if link in countlist” creates a significant delay when the count list is long. In the eThekwini case study, building the new list containing the 982 938 valid path ID / count ID combinations increased the runtime from 3,6 to 90 seconds.

This is quite understandable, considering that Python is being asked to execute 17,93 million tests to ascertain whether the link is also a count by scanning through 260 count IDs. *Optimisation of Python coding in this critical section of the PPA algorithm is earmarked for future research.* However, since the primary objective of this paper is proof of concept, the results are acceptable for now, even with the identified timing issue.

Add-on libraries such as Numpy and Pandas are best for managing operations that involve arrays or relational databases. The Pandas library in particular offers capabilities that are highly optimised for processing databases by column. When using a Pandas DataFrame, one should also, as far as possible, avoid code that executes database procedures in a “Row by Agonising Row” (RBAR) manner, as first described by Jeff Moden (Microsoft, 2021) in the SQLServerCentral online forums (Moden, 2014). The efficiency of the Pandas library is optimised when processing by column, as is the case for any relational database management system (RDBMS) that uses Structured Query Language (SQL) syntax.

The code snippet below illustrates the adjustments done in only a few lines of code within the PPA inner loop, using both Pandas DataFrame and Numpy methods:

```
# Aggregate Count, PriorVol and AdjVol by CountID for all turns
query=df_pathlist.groupby('CountID',as_index=False).agg({'Count':'mean','PriorVol':'sum','AdjVol':'sum'})
df_countfactors = pd.DataFrame(query)

# Add a column for the adjustment factor
df_countfactors['Factor'] = np.round(
    np.where(abs(df_countfactors['AdjVol'])>=0.0001,
    (df_countfactors['Count'] / df_countfactors['AdjVol']),0),6)

# Apply the adjustment factor to all fractional counts, using a left join
df_pathlist = pd.merge(df_pathlist,df_countfactors[['CountID','Factor']], on="CountID", how="left")
df_pathlist['AdjVol']=df_pathlist['AdjVol']*df_pathlist['Factor']
del df_pathlist['Factor'] #Drop Column Factor, no longer needed

# Compute the average AdjVol for each unique PathID
query=df_pathlist.groupby('PathID',as_index=False).agg({'AdjVol':'mean'})
df_pathmeans = pd.DataFrame(query)
df_pathmeans.rename(columns = {'AdjVol':'AdjAve'}, inplace = True)
df_pathmeans['AdjAve'] = np.round(df_pathmeans['AdjAve'],6)

# Replace AdjVols with the path averages, using a left join
df_pathlist = pd.merge(df_pathlist,df_pathmeans[['PathID','AdjAve']], on="PathID", how="left")
df_pathlist['AdjVol']=df_pathlist['AdjAve'] # Replace the AdjCount with path average
del df_pathlist['AdjAve'] #Drop path average, no longer needed
```

The code snippet above illustrates:

- Execution of SQL-type queries operating on the Pandas DataFrame by column, inclusive of group summation, path averaging and left joins.
- Numpy numerical processing, applied to Pandas DataFrame columns to calculate the adjustment factors.

These column-based operations avoid RBAR and run extremely quickly, with very little impact on the overall computer run-time.

4. EMME/4 MODEL RESULTS AND ANALYSIS

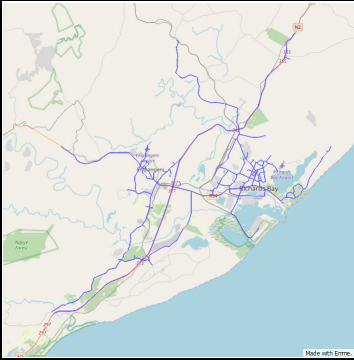
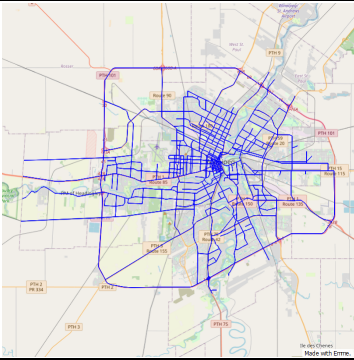
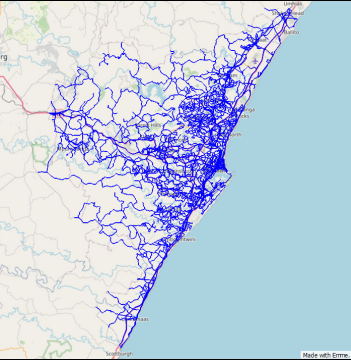
4.1 Case Studies

Three case studies with network sizes ranging from small to large, were used to test the conversion of the PPA algorithm into a custom Emme/4 Modeller Tool:

- The Umhlathuze model used for the preparation of a Traffic Impact Assessment in support of a proposed wastewater reuse facility. This model, used previously (Oberholzer, 2021), was also included in this paper to illustrate application of the PPA algorithm in a small network.
- The Winnipeg model supplied with Emme/4 and used throughout the help files and API documentation to explain Emme/4's capabilities using appropriate examples drawn from this model.
- The eThekweni Macro Model 2008 Base Year, as expanded and calibrated by Goba, on behalf of the eThekweni Transport Authority (eThekweni Municipality, 2013).

Table 4 illustrates the main Emme/4 model features, for each of the selected case studies.

Table 4: Case Studies – Model Features

Description	Umhlathuze	Winnipeg	eThekweni
Network Map			
Size	Small	Medium	Large
Zones	80	154	646
Nodes	182	893	4 617
Links	596	3 027	13 054
Link Counts	15	70	260
Turn Counts	130	-	-
Prior Demand	20 398	56 219	114 373

4.2 Model Runs and Results

The Emme/4 models were already calibrated and contained suitable prior demand matrices and link and/or turn counts stored in attributes. Model runs were thus straightforward for all three case studies, comprising:

- Matrix Adjustment, using the custom-developed PPA Modeller Tool.
- Standard Multi-class traffic demand adjustment, using SOLA.
- Extraction of module run-times timings from the Emme/4 logbook.
- Display of Emme/4 standard link and turn scatterplot worksheets.

Table 5 illustrates the case study modelling results.

Table 5: Case Studies - Results and Analysis

Description	Umhlatuze	Winnipeg	eThekwini
PPA runtime	04 iterations: 09 s	20 iterations: 1 m 07 s	10 iterations: 21 m 42 s
Std Runtime	30 iterations: 64 s	30 iterations: 2 m 41 s	24 iterations: 11 m 56 s
PPA Scatterplots			
Standard Emme matrix adjustment Scatterplots			

5. CONCLUSION

The matrix estimation algorithm based on proportional path averages, implemented as a custom-built Emme/4 Modeller Tool, is a viable alternative to Emme/4's Multiclass traffic demand adjustment. For the small- and medium-sized case studies, the PPA tool performed faster than its Emme/4 counterpart, with equally accurate results, as indicated by the scatterplots.

For the large case study, the PPA tool was also as accurate, but the computer runtime was almost double that of the built-in Emme/4 module. The inner loop that checks whether any links in an OD path are in the list of counts, was identified as the primary cause of the lengthy runtime, adding 86 seconds to each outer loop that re-creates the master fractional path table, after a path-based traffic assignment.

This paper confirms proof of concept. Further research will be required to:

- Expand the algorithm to include multi-class assignments.
- Optimise the Python coding to reduce the timing delays during master table creation.
- Delve deeper into the underlying path-building within the Emme/4 core, i.e., methods and procedures currently not exposed by the Emme/4 APIs, to explore further optimisation.

6. REFERENCES

Abrahamsson, T, 1998. Estimation of Origin-Destination Matrices Using Traffic Counts – A Literature Survey. IIASA Interim Report. Laxenburg, Austria.

eThekwini Municipality, 2013. eThekwini Macro Transport Model Development - Model Calibration and Validation. Goba Reference: EMTM-MCV-20130130-REV0.

INRO, 2020. Emme/4 documentation Release 4.4 (Last Modified February 25, 2020).

iPython, 2022. BSD licenced computing tool used with Python. Available at: <http://ipython.org/>. Accessed 17 January 2022.

Lindström, A & Persson, F, 2018. Estimation of Hourly Origin Destination Trip Matrices for a Model of Norrköping. Working Paper. Department of Science and Technology, Linköping University, Norrköping, Sweden.

Microsoft, 2021. Most Valuable Professional – Jeff Moden. Available at: <https://mvp.microsoft.com/en-us/PublicProfile/4020758>. Accessed 17 January 2022.

Moden, J, 2014. The "Numbers" or "Tally" Table: What it is and how it replaces a loop (first published 2008). Available at: <https://www.sqlservercentral.com/articles/the-numbers-or-tally-table-what-it-is-and-how-it-replaces-a-loop-1>. Accessed 17 January 2022.

Noriega, Y & Florian, M, 2007. Multi-Class Demand Matrix Adjustment. CIRRELT, Montreal.

Noriega, Y & Florian, M, 2009. Some Enhancements of the Gradient Method for O-D Matrix Adjustment. CIRRELT, Montreal.

NumPy, 2022. IP[y] IPython Interactive Computing. Open-source numerical computing tool used with Python. Available at: <https://ipython.org/>. Accessed 17 January 2022.

Oberholzer, J de V, 2021, Matrix Estimation from Traffic Counts: An Alternative Approach Based on Proportional Path Averages. 39th Southern African Transport Conference.

Pandas, 2022. Open-source data analysis and manipulation tool, built on top of the Python programming language. Available at: <https://pandas.pydata.org/>. Accessed 17 January 2022.

Python, 2022. Python Software Foundation. Available at: <https://www.python.org/>. Accessed 17 January 2022.

Python Package Index, 2022. Software Repository for the Python programming language. Available at: <https://pypi.org/>. Accessed 17 January 2022.

Sipser, M, 2006. Introduction to the theory of computation, 2nd Edition. Course Technology, a division of Thomson Learning, Inc. ISBN 0-534-95097-3.

Spieß, H, 1987. A maximum-likelihood model for estimating origin-destination matrices. Transportation Research 21B.

Spieß, H, 1990. A Gradient approach for the O-D matrix adjustment problem. CRT Pub, Centre de recherche sur les transports (CRT), University of Montréal, Québec, Canada.

van Rossum, G, 1993. An Introduction to Python for UNIX/C Programmers. Proceedings of the NLUUG Najaarsconferentie (Dutch UNIX Users Group).