Traffic Assignment with Junction Modeling in TAPAS

Author:
Oedsen van der Kooi

Supervisors:
ir. Feike Brandt
dr. Georg Still
prof. dr. Marc Uetz

June 2015

UNIVERSITY OF TWENTE.

dat. mobility

Contents

Chapter 1

Introduction

1.1 Overview

During the twentieth century the amount of traffic increased all around the world, resulting in crowded roads and increased travel times. This sparked the need for research to predict traffic flows. One of the most widely used traffic models is the four step model. This model is used for various purposes, such as predicting the impact of road works, calculating the long term effects on the road network of building new population areas or comparing different solutions for restructuring a bottleneck in the network. The fourth step of this model (which will be treated in more detail in section 2.3) is the Traffic Assignment Problem (TAP). The TAP uses as input a study area divided up into zones, with a graph representing the road network, and an Origin-Destination (OD) matrix which contains the amount of traffic that needs to travel over the network from one zone, the origin, to another zone, the destination. There are different types of Traffic Assignment problems, based on the assumptions made on the network and the travelers, with different solution methods for each type. Throughout this study, it is assumed that there is congestion in the network, which is modeled by link cost functions $c_a(f_a)$ that increase monotonically as the flow on the link f_a increases. Also all travelers behave the same, each traveler has complete information on the state of the network and the decisions of other travelers and each traveler tries to make his travel time as small as possible.

The TAP can be applied to both static and dynamic models. Each model has its advantages: in a dynamic model the traffic propagates through the network as time passes and the temporal behavior of traffic jams and queues at traffic lights can be modeled precisely, but algorithms for dynamic models take a lot of

calculation time. Algorithms for static models on the other hand are faster and can be used for much larger study areas, but are less capable to show the effects of traffic jams and queues. This study focuses on static models.

The notion of travelers making their own choices and trying to minimize their own travel time was formalized by Wardrop[1] in 1952 when he stated the principle of User Equilibrium (UE). A mathematical programming formulation was introduced by Beckmann et al.[2] and this formulation was used by several algorithms to solve the TAP. More information on this topic can be found in section 2.3. One shortcoming of these algorithms is that the travel time of a traveler only depends on the amount of traffic on the road the traveler uses, whereas in real life situations a substantial amount of travel time can be attributed to travelers on crossing roads. An example of this is a junction where the traveler has to wait because he has to give way to crossing traffic. This behavior can be captured by adjusting certain link cost functions so that travel time on these links also depend on crossing traffic. This methodology, called Junction Modeling, adds realism to the traffic model but it also complicates the model: the User Equilibrium is no longer unique and the link cost functions that depend on multiple link flows are no longer differentiable and the Beckmann notation can no longer be used. Several heuristics are available to tackle these problems.

This master thesis is commissioned by DAT.Mobility. DAT.Mobility, based in Deventer, the Netherlands, is a software company specialized in traffic and transport. DAT.Mobility has approximately 30 employees. The company was formed in 2014 when DAT.Mobility's predecessor Omnitrans International BV merged with several departments of Goudappel Coffeng to form DAT.Mobility. While the core business of DAT.Mobility is developing and marketing software products such as the transport modelling software package OmniTRANS, a substantial amount of time is committed to other projects covering different subjects in the field of traffic modelling and mobility. Consultants and authorities use OmniTRANS for strategic long term planning. OmniTRANS has an extensive Junction Modeling tool: for each node in the network the user can specify the type of the junction, such as roundabout or traffic signal, and each type can be further specialized by entering parameters such as the number of lanes. Two algorithms for solving the TAP are implemented in OmniTRANS, called Frank Wolfe and Volume Averaging. These algorithms are known to converge badly and in order to keep up with competitors, DAT.Mobility wants to add a new algorithm to the tool set of OmniTRANS. Several new algorithms have been introduced in recent years and during an internship at DAT.Mobility, a comparison was made between the most promising of these new algorithms. One of these is Traffic Assignment by Paired Alternative

Segments (TAPAS), introduced by Bar-Gera in 2009[3]. TAPAS was found to be the best new suited algorithm for adding to OmniTRANS[4]. However, there is no Junction Modeling functionality in TAPAS as described in the original paper by Bar-Gera. So this functionality has to be added to TAPAS so that TAPAS can use the extensive Junction Modeling tool box available in OmniTRANS and be a suitable candidate for replacing the algorithms currently implemented in OmniTRANS. Also some parts of the algorithm are not described in complete detail or are left open for interpretation, so these parts need to be filled in.

1.2 Research Question

The main research question of this Master Thesis is:

Can TAPAS be fit with the Junction Modeling functionality of OmniTRANS such that it can be used in the OmniTRANS software as an improvement of the currently implemented algorithms?

In order to accurately answer this question it is split up in several sub questions:

- How does TAPAS work? Can we give a detailed and comprehensive description of all components of TAPAS?
- How is Junction Modeling implemented in OmniTRANS?
- What changes need to be made to TAPAS in order to be able to function with asymmetric cost functions?
- Can we show, using a prototype, that TAPAS with the proposed changes calculates results that are better than the results calculated by the algorithms currently implemented in OmniTRANS?

The rest of this thesis is structured as follows: chapter 2 contains the notation and formulas used throughout the thesis, along with a description of the Traffic Assignment Problem and an overview of related work on algorithms for TAP, both with regular and with asymmetric cost functions. Chapter 3 gives a detailed description of TAPAS, with solutions offered for all elements of the algorithm that are not clearly described in the original paper. In chapter 4 the Junction Modeling module of OmniTRANS is described and all changes that need to be made to TAPAS in order to handle Junction Modeling are treated. Chapter 5 gives an overview of the prototype that was implemented in OmniTRANS and shows

the results acquired with the prototype in the form of a comparison between the solutions calculated by TAPAS and by OmniTRANS. Chapter 6 points out some elements that could be added to TAPAS and to the prototype made and gives recommendations for further research. Chapter 7 contains the conclusions of this master thesis and recommendations for further research.

Chapter 2

Background

2.1 Notation

The transportation network will be represented by the graph $G = \{N, A\}$, where N is the set of nodes and A is the set of links. $N_o \subseteq N$ is the set of origins and for each $p \in N_o$ we define $N_d(p) \subseteq N$ as the set of destinations for origin p. All links $a = (n, m) \in A$ are directed and start at node n and end at node m. A route segment s is a sequence of distinct nodes $[n_1, n_2, \ldots, n_x]$ for some $x \in \mathbb{N}$ such that $a_i = (n_i, n_{i+1}) \in A$ for each $1 \leq i \leq x - 1$. If $x = 1$ the route segment consists of just one node n_1. If $n_1 \in N_o$ and $n_x \in N_d(n_1)$, we call the segment a route, denoted by r. The first node n_1 of a segment s is called the tail, denoted by s_t and the last node n_x is called the head, denoted by s_h. Similarly, the start node n of a link $a = (n, m)$ is called the tail a_t and the end node m is called the head a_h. The set of all routes connecting origin p to destination q is called R_{pq}. The set of all possible routes connecting an origin to a destination is called $\boldsymbol{R} = \bigcup_{p \in N_o} \bigcup_{q \in N_d(p)} R_{pq}$. For a node n we define the set of incoming links as $IN_n = \{a \in A | a_h = n\}$ and the set of outgoing links as $OUT_n = \{a \in A | a_t = n\}$. The demand for an OD pair pq is denoted by d_{pq}. We denote flows by three levels of aggregation. The highest level of aggregation is the total flow on link a, denoted by f_a. The vector of all link flows is denoted by $\boldsymbol{f}$. The flows can be disaggregated by routes: the flow along a route $r \in \boldsymbol{R}$ is denoted by h_r. The vector of all route flows is denoted by $\boldsymbol{h}$. Link flows can also be disaggregated by origin, resulting in an $|N_o|$ by $|A|$ vector of origin based (OB) flows $\boldsymbol{f}$ whose elements are denoted by f_{pa}. From the context it will be clear whether $\boldsymbol{f}$ denotes $(f_a, a \in A)$, or $(f_{p,a}, p \in N_o, a \in A)$. We denote by $G_p = \{N, A_p\}$ the subgraph of G with $A_p = \{a | f_{pa} > 0\}$. For origin p, the total incoming origin based flow in a node n is called the node flow,

denoted by g_{pn}. If the OB node flow g_{pn} for node n is greater than zero, we can define for a link $a \in IN_n$ the proportion of g_{pn} that enters node n through link a, called the approach proportion α_{pa}. The origin based flow on a route segment s is called the segment flow k_{ps}, which can be calculated using formula 2.2.4. We can calculate link flows, node flows, segment flows and approach proportions given the OB link flows, these formulas are defined in section 2.2. Link costs are denoted by $c_a(\boldsymbol{f})$. The cost of a route segment $s = [n_1, n_2, \ldots, n_x]$ is the sum of the link costs of all links in s: $c_s(\boldsymbol{f}) = \sum_{i=1}^{x-1} c_{(n_i,n_{i+1})}(\boldsymbol{f})$. Given the current flow pattern $\boldsymbol{f}$ and corresponding link costs $c_a(\boldsymbol{f})$ we denote the cost of the shortest path from node n to node m as $\pi_{nm}(\boldsymbol{f})$. The main building blocks of the algorithm, Paired Alternative Segments or PASs, are defined as follows: a PAS consists of two route segments s_1 and s_2, where both segments have the same head and tail node and no other common nodes. The tail node is called the diverge node, the head node is called the merge node. Throughout this thesis we assume that segment s_1 is the segment with the highest cost, unless explicitly stated otherwise. We define $\mathcal{P}$ as the set of PASs, the set of relevant origins of PAS P is called O_P. For quick reference, please consult the list of abbreviations at the end of this report.

2.2 Commonly Used Formulas

This section introduces a list of formulas that are widely used throughout the rest of the thesis. Since TAPAS uses the OB flows f_{pa} as the basic solution variable, all flow variables are expressed as functions of the OB flows. The total flow on a link is simply the sum of all OB flows on that link:

$$f_a = \sum_{p \in N_o} f_{pa} \tag{2.2.1}$$

The origin based node flow of a node n is defined as the sum of the OB flow of all incoming links:

$$g_{pn} = \sum_{a \in IN_n} f_{pa} \tag{2.2.2}$$

The origin based approach proportion of a link a for origin p is the fraction of the total node flow at node a_h that enters a_h through a:

$$\alpha_{pa} = \frac{f_{pa}}{g_{pa_h}} \tag{2.2.3}$$

The origin based segment flow for a route segment $s = [n_1, n_2, \ldots, n_x]$ is calculated as follows:

$$k_{ps} = g_{pn_x} \cdot \prod_{i=1}^{x-1} \alpha_{p(n_i, n_{i+1})} \tag{2.2.4}$$

If we have a route flow solution $\boldsymbol{h}$ that satisfies the proportionality principle (see also section 3.6), we can also calculate the segment flow for s using route flows by taking the sum of the route flows for all routes from origin p that contain s:

$$k_{ps} = \sum_{q \in N_d(p)} \sum_{r \in R_{pq} | s \in r} h_r \tag{2.2.5}$$

The link cost function used most widely is the BPR function, published by the Bureau of Public Roads:

$$c_a(\boldsymbol{f}) = \frac{L_a}{v_a}\left(1 + \alpha\left(\frac{f_a}{q_a}\right)^{\beta}\right)$$

where

$c_a(\boldsymbol{f})$	Cost of link a (depends only on f_a in this function)
L_a	Length of the link
v_a	Free flow speed on the link
f_a	Flow on the link
q_a	Maximum capacity of the link
α, β	Constants

Here α is usually set to 0.87, β is usually 4. The measure for determining the convergence of a solution $\boldsymbol{f}^i$ calculated in iteration i that is used throughout this thesis is called the relative duality gap. The relative duality gap calculates the difference between the total travel time of the solution $\boldsymbol{f}^i$ in iteration i and the travel time of a solution that only used the shortest paths in iteration i, weighted by the shortest path travel times. It is defined as follows

$$\frac{\sum_{a \in A} f_a^i c_a(\boldsymbol{f}^i) - \sum_{p \in N_o,\, q \in N_d(p)} \pi_{pq}^i d_{pq}}{\sum_{p \in N_o,\, q \in N_d(p)} \pi_{pq}^i d_{pq}} \tag{2.2.6}$$

The travel time of a solution that only uses shortest paths is a lower bound for the travel time in a UE, so the duality gap is an upper bound for the difference in cost between the current solution and the UE solution. The duality gap is weighted by the shortest path travel times so that it is possible to compare performance of algorithms on different networks.

2.3 The Traffic Assignment Problem

The traffic assignment problem is part of the widely used general traffic model called the four step model, described by Ortuzar and Willumsen [5]. This traffic model is used to describe and predict traffic flows in a study area. A network that models the relevant transport infrastructure in the study area is created and the study area is divided into different zones. For each zone socioeconomic data such as population, employment, income, car ownership, recreational and economical facilities is collected. This data is acquired from sources such as questionnaires, road sensors and local authorities. The data is then processed in four steps:

1. Trip generation
2. Trip distribution
3. Modal split
4. Assignment

In the first step, called trip generation, the socioeconomic data is used to estimate the total amount of traffic leaving from and going to each zone, using statistical methods. A unit of flow leaving a zone is called a production, a unit of flow arriving at a zone is called an attraction. The second step, called trip distribution, links each production to an attraction, resulting in an Origin-Destination (OD) matrix. This step is performed using for instance a gravity model or a discrete choice method. In the third step, the modal split, each trip in the OD matrix is linked to a mode of transport, such as car or public transport. In the last step, the traffic assignment, the trips in the OD matrix are assigned to routes in the network. The study in this report focuses completely on the last step.

In 1952 Wardrop [1] introduced the notion of the User Equilibrium as a means to describe behavior of travelers. In a User Equilibrium no traveler can reduce his travel time by deviating from his current route. In this model it is assumed that all travelers have complete information and travel times on a link increase when more travelers use the link. Wardrop formalized this notion by stating the User Equilibrium conditions: a path flow solution $\boldsymbol{h} = (h_r, r \in \boldsymbol{R})$ is a UE solution if

it satisfies the following conditions:

$$h_r(c_r(\boldsymbol{h}) - \pi_{pq}) = 0, \quad \forall r \in \boldsymbol{R}; \tag{2.3.1}$$

$$c_r(\boldsymbol{h}) \geq \pi_{pq}, \quad \forall r \in \boldsymbol{R}; \tag{2.3.2}$$

$$\sum_{r \in R_{pq}} h_r = d_{pq}, \quad \forall p \in N_o, \ q \in N_d(p); \tag{2.3.3}$$

$$h_r \geq 0, \quad c_r(\boldsymbol{h}) \geq 0, \quad \forall r \in \boldsymbol{R}, \tag{2.3.4}$$

where π_{pq} is the optimal travel time between OD pair p and q. Equation (2.3.1) states that either the flow on a path between OD pair p and q is zero or, if a route has flow then the cost of the path is equal to the optimal travel time. Equation (2.3.2) ensures that no path has lower cost than the optimal travel time. Finally equation (2.3.3) ensures that all demand is satisfied and equation (2.3.4) are the nonnegativity constraints.

In 1956 Beckmann et al.[2] introduced a mathematical programming notation that can be used to find the User Equilibrium for the TAP with link flows $\boldsymbol{f} = (f_a, a \in A)$:

$$(BE) \quad \min_{\boldsymbol{f},\boldsymbol{h}} z(f) = \sum_a \int_0^{f_a} c_a(\omega) \mathrm{d}\omega \tag{2.3.5}$$

$$\text{s.t.} \quad \sum_{r \in R_{pq}} h_r = d_{pq}, \quad \forall p \in N_o, q \in N_d(p); \tag{2.3.6}$$

$$h_r \geq 0, \quad \forall r \in \boldsymbol{R}; \tag{2.3.7}$$

$$f_a = \sum_{r \in \boldsymbol{R}} \delta_{a,r} h_r, \quad \forall a \in A \tag{2.3.8}$$

Here equation (2.3.6) ensures that the demand is satisfied, equation (2.3.7) makes sure that all path flows are non-negative and equation (2.3.8) connects the path flows to the link flows. Here $\delta_{a,r}$ is 1 if link a is part of route r, and 0 otherwise. The link between the Beckmann notation and the User Equilibrium is stated without proof in the following theorem:

Theorem 2.1. *Assume $c_a(\boldsymbol{\omega})$ is monotonically increasing and $\boldsymbol{h}$ is a feasible route solution with corresponding link solution $\boldsymbol{f}$. Then $\boldsymbol{h}$ is a User Equilibrium solution if and only if $(\boldsymbol{f}, \boldsymbol{h})$ is an optimal solution of (BE)*

Remark r1: If the functions $c_a(\omega)$ are strictly increasing the link flow part f of a UE $(\boldsymbol{f}, \boldsymbol{h})$ is uniquely determined the path flow part h is not.

2.4 Related Work

2.4.1 Algorithms for TAP

In 1975 LeBlanc et al.[6] introduced the Frank Wolfe (FW) algorithm for solving the Traffic Assignment problem by approximately solving (BE). FW was the first algorithm that could calculate solutions that are close to the UE solution. FW stores a link flow vector $\boldsymbol{f}^n$ that is improved in each iteration n: first shortest paths from each origin p to each destination q are calculated, with the link costs induced by the flow vector of the previous iteration $c(\boldsymbol{f}^{n-1})$. Then an All or Nothing assignment is performed on these shortest paths, resulting in a descent direction w^n. Finally the optimal step size $0 \leq \lambda_n \leq 1$ is calculated as minimizer λ_n of

$$\min_{0 \leq \lambda_n \leq 1} \sum_a \int_0^{\lambda_n f_a^{n-1} + (1-\lambda_n) w_a^n} c_a(\omega) \mathrm{d}\omega$$

Flows are then updated using

$$f_a^n = \lambda_n f_a^{n-1} + (1 - \lambda_n) w_a^n \forall a \in A$$

FW has the advantage that it is quick and easy to implement and has low memory usage, but converges badly in later iterations. An algorithm similar to FW is Volume Averaging (VA). The only difference between FW and VA is the calculation of the step size, which depends only on the iteration index in VA: $\lambda_n = 1 - \frac{1}{n}$. VA also shares the bad convergence of FW. Both of these algorithms are currently implemented in OmniTRANS.

For many years, FW and VA were the only algorithms used in practical settings, due to limitations in memory and processor speeds of computers. When computer performance increased rapidly during the 90s, research in new algorithms that exploited these new possibilities started, and several new algorithms were published. All these new algorithms had in common that they store flows on a more disaggregated level than FW, so that they are able to shift flows across the network much more precisely. So all algorithms trade off memory usage for shorter calculation time. In 2002 Bar-Gera[7] started the line of bush-based algorithms by introducing a new algorithm called Origin Based Algorithm (OBA). Other bush-based algorithms include Algorithm B by Dial [8], introduced in 2006, and the Linear User

Cost Equilibrium Algorithm (LUCE) by Gentile [9], introduced in 2009. This type of algorithms use origin based flows f_{pa} and introduced the notion of bushes. The algorithm stores a bush $B_p = (V', A')$ for each origin p. B_p is a subgraph of the original graph G with $V' = V$ and $A' \subset A$ such that B_p is acyclic, rooted at p and each node that is reachable from p in G is also reachable in B_p. Acyclicity is important for two reasons: first, it is possible to make a topological ordering of the nodes in an acyclic graph. In a topological ordering of an acyclic graph with root p each node i is assigned a number τ_{pi} such that if there exists a path from node i to node j then $\tau_{pi} < \tau_{pj}$. With this topological ordering, finding shortest and longest paths in an acyclic subnetwork can be done much more efficiently than in full networks. Second, bush-based algorithms such as OBA use the fact that in a UE solution the OB flows do not contain cycles. Therefore, if one is able to include in each bush the links that carry flow in the UE solution, the problem of finding a User Equilibrium for the whole network reduces to finding a User Equilibrium for the bushes B_p. All bush based algorithm share the same structure: in each iteration the bush is updated by adding links that could improve the current solution and removing unneeded links, while still keeping the acyclic structure, and then flows are shifted on the updated bush such that the new solution is closer to the User Equilibrium. Each of the bush based algorithms uses a different approach to updating the bush and shifting flows. For more information on this topic we refer to the original papers.

The Projected Gradient method was introduced by Florian et al.[10] in 2009. The main decision variables in PG are path flows h_r. For each OD pair pq it stores a set of paths R'_{pq}. Flow is only shifted between these paths. An iteration of PG consists of two parts: first for each OD pair pq the set of paths R'_{pq} is updated by adding paths that could improve the solution and removing paths that are not used anymore, then flow is shifted between the paths such that the solution is closer to the UE, using the Projected Gradient Method introduced by Rosen[11].

In 2012 Inoue et al.[12] published a study in which they compared convergence rates of nine different algorithms for the Traffic Assignment problem, including TAPAS and all algorithms described above. The results of this study can be seen in figure 2.1. In this figure OBA is called Bar-Gera's algorithm and PG is called DSD/PG. It shows that TAPAS performs best, followed by Alg B, OBA, LUCE and PG.

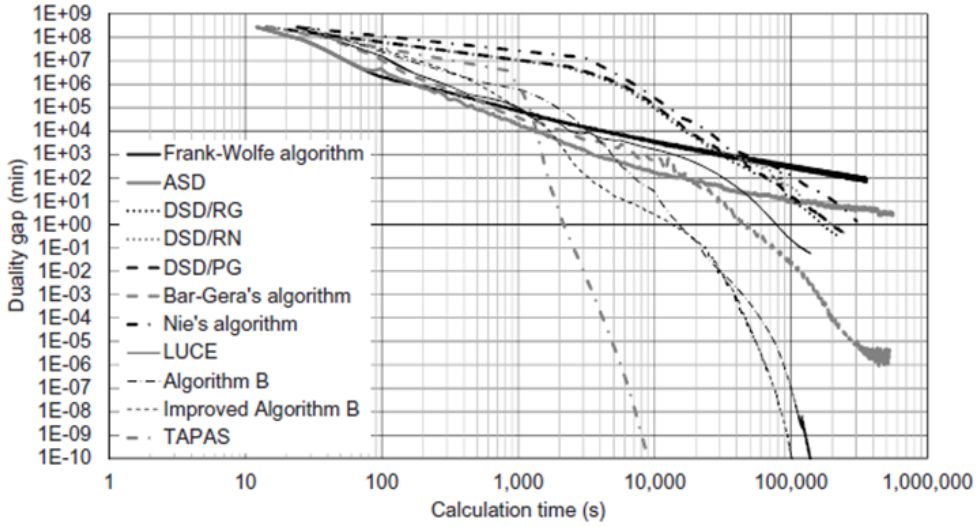

FIGURE 2.1: Speed of convergence of different algorithms

2.4.2 TAP with asymmetric link costs

In 1971 Dafermos [13] starts the research of the field of traffic assignment with asymmetric link cost functions: in this so called extended traffic assignment model the cost of traveling across a link becomes a function of the link flows on the entire network. Dafermos shows that in the extended model a unique UE is only guaranteed if

$$\sum_{a \in A} c_a(\boldsymbol{f}) \tag{2.4.1}$$

is strictly convex, or equivalently if the Jacobian

$$\left[\frac{\partial c_a(\boldsymbol{f})}{\partial (f_b)} \right] \tag{2.4.2}$$

is positive definite for all feasible $\boldsymbol{f}$. This is a very strong condition which will not hold in most practical applications. Also a conceptual algorithm for the extended model is given where flow is shifted from the longest used path between an origin and destination to the shortest path. This algorithm is not usable in practical problems, because finding longest paths is an NP-complete problem. In 1980 Dafermos [14] expanded this work by showing the connection between a UE flow solution $\overline{\boldsymbol{f}}$ and variational inequalities which will be stated without proof here: a flow solution $\overline{\boldsymbol{f}}$ is a user equilibrium if and only if

$$c(\overline{\boldsymbol{f}})(\boldsymbol{f} - \overline{\boldsymbol{f}}) \geq 0 \quad \forall \, \text{feasible} \boldsymbol{f} \tag{2.4.3}$$

Dafermos then uses the variational inequality notation to prove that a unique UE exists if the cost functions are nonconstant and affine. Also another algorithm is proposed that uses linear estimations of the cost functions to compute its descend direction, comparable to the Newton method. Smith [15] introduces a concept algorithm for the asymmetric problem that is similar to FW. A method for finding a descend direction using the variational inequality is given in a later paper (Smith [16]). This algorithm works in theory, but needs to store all AON paths calculated, so it is not usable in a practical setting. Lawphongpanich and Hearn [17] introduces a similar algorithm that uses simplicial decomposition. It approximately solves the variational inequality. The algorithm shows decent results on small test networks. Florian and Spiess [18] proposes a diagonalization method that can be used for less restrictive cost functions that have an asymmetric Jacobian matrix. In this method at the beginning of each iteration the cost functions are approximated by fixing the off-diagonal flow values for each link cost function. This ensures a Jacobian matrix that has zeroes on all off-diagonal places. Local convergence of the diagonalization method is proved under the condition that the cost functions are differentiable in the neighborhood of a local equilibrium $\overline{\boldsymbol{f}}$. More information on this method can be found in 4.3.1. Muijlwijk [19] showed in her master thesis that multiple equilibria exist when using the junction functions implemented in OmniTRANS. Some solutions for finding the same equilibrium, regardless of the starting state, are presented. More information on this subject can be found in section 4.3.3.

Chapter 3

A Detailed Description of TAPAS

The Traffic Assignment Problem can be seen as an generalized version of the minimum cost flow problem. A technique widely used in algorithms for the minimum cost flow problem is identifying negative cost cycles and reducing flow on these negative cost cycles. Examples of these algorithms are described by Klein [20] and Goldberg and Tarjan [21]. A negative cost cycle is a sequence of nodes $C = \{n_1, n_2, \ldots, n_x\}$ such that $n_1 = n_x$ and every two consecutive nodes are connected by either a link in the same direction as the cycle, called a forward link $(n_i, n_{i+1}) \in A$ or a link in the direction opposite to the direction of the cycle, called a backward link $(n_{i+1}, n_i) \in A$. The set of forward links of a cycle is called F, the set of backward links is called B. Also each backward link in a negative cost cycle should have positive flow and the sum of the link costs of the negative cycle c_C, where $c_C = \sum_{a \in F} c_a - \sum_{a \in B} c_a$, should be negative. If we define $\delta = \min_{a \in B} f_a$, we can add δ units of flow to each forward link and subtract δ units of flow from each backward link. This operation is called sending δ units of flow along the cycle. It leaves all flow constraints satisfied and reduces the overall cost of the flow, because c_C is negative. An example of a negative cost cycle is given in figure 3.1 by the clockwise path through the links in bold. Here $\delta = \min_{a \in B} f_a = f_{(10,14)} = 2$ and $c_C = -2$. Sending an amount of less than 2 units of flow along the cycle keeps all flows positive and results in a reduction of the total cost.

In TAPAS flows are saved for each origin separately. Given a flow pattern $\boldsymbol{f} = (f_{pa}, p \in N_o, a \in A)$ that is not converged for at least one origin p, TAPAS improves the solution by searching the sub graph G_p of all links that carry flow coming from origin p for two types of negative cost cycles. The first and simplest one is a cycle C that consists only of backward links. In this case there is cyclic flow present on the network for origin p. A UE solution can not contain cyclic flow, so we can take $\delta_p = \min_{a \in C} f_{pa}$ and eliminate the cycle by sending δ_p units of

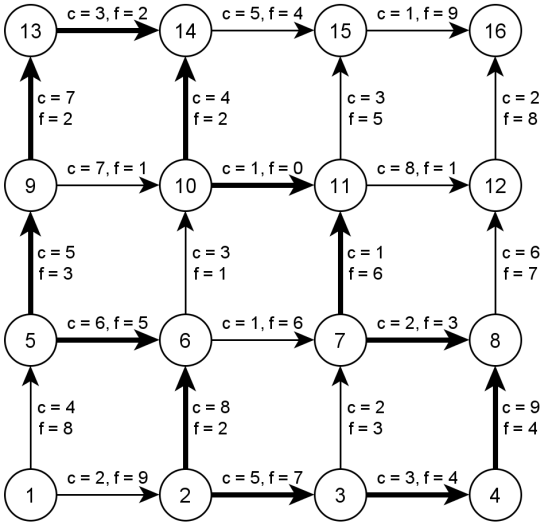

FIGURE 3.1: Example of a negative cost cycle

flow along it. Finding such a cycle can be done efficiently by algorithms such as Tarjan's algorithm described in [22] or the path-based strong component algorithm by Dijkstra [23, Ch. 25]. The second type of negative cost cycle is a Pair of Alternative Segments: two route segments that have the same diverge node and merge node, and no other nodes in common, where the segment with the highest cost has positive flow on each link of the segment. If we choose the direction of the negative cost cycle such that all links in the lower cost segment are forward links and all links in the higher cost segment are backward links, we can shift flow from the high cost segment to the lower cost alternative by sending flow along the cycle. This results in a decrease of the objective function. The basic steps of the algorithm are therefore identifying and removing cyclic OB flow, identifying PASs and shifting OB flow from the high cost segment to the low cost alternative for all PASs.

TAPAS shares similarities with the bush-based algorithm Algorithm B, introduced by Dial [8]. Both use solution variables f_{pa} that are disaggregated by origin and also flow shifts are performed by shifting flow from a high cost route segment to a low cost alternative. There are also differences: Algorithm B stores an acyclic set of links called a bush B_p for each origin p. Flow is shifted only between links that are part of the bush. So the main steps of Algorithm B are adding all links that are part of the shortest path tree of each origin p to the bushes B_p and equilibriating all bushes. High cost and low cost segments are found by backtracking over the bushes, starting at a merge node. Here the acyclic structure of the bushes helps greatly: shortest and longest paths can be found in one pass over the links. TAPAS does not use the bush structure so it has more efficient memory usage, since it

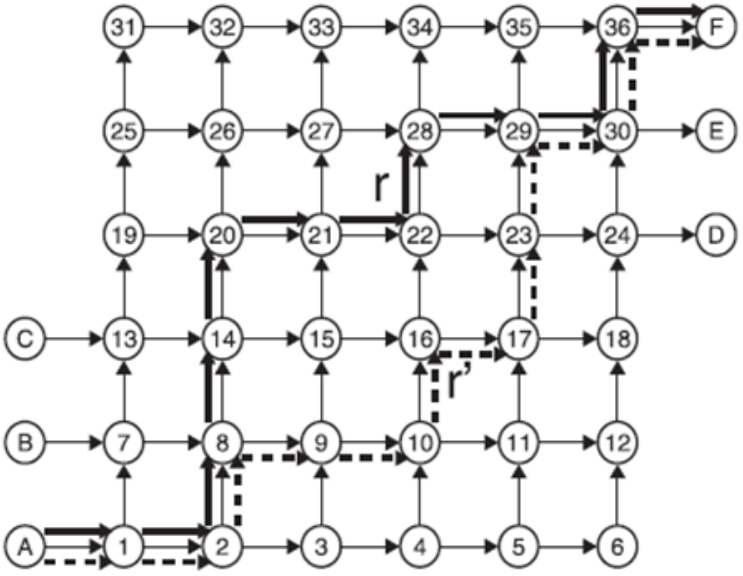

FIGURE 3.2: Example of a PAS

does not need to store all bushes. TAPAS uses a different approach and stores all PASs that are found, so that the algorithm can prevent constructing the same PAS over and over again. Moreover, TAPAS stores all relevant origins for a PAS. This way flow shifts can be performed on path segments that are used by several origins in one calculation.

We can show the strength and flexibility of the PAS structure using the example of page 1028 of Bar-Gera [3] using the network shown in figure 3.2. Here two routes, r and r', from origin A to destination F are depicted. The routes diverge at node 8 and merge at node 29. We assume that in this example route r is the route with the higher cost. A simple way to shift flow from route r to r' would be to construct a PAS with segments $[8, 14, 20, 21, 22, 28, 29]$ and $[8, 9, 10, 16, 17, 23, 29]$. There are 252 different routes from origin A to destination F, so in the simple approach up to $252 \cdot 251 = 31,626$ PASs may be needed to perform flow shifts for all pairs of routes. So while this approach would indeed ensure that flow shifts between any two routes are possible, a lot of PASs are needed and each PAS can only shift flow between two routes. Instead, TAPAS uses a different approach where one PAS can shift flow between many routes for more than one OD pair: instead of using one PAS with long segments, a set of PASs with short segments is used. In the example the smallest possible PAS is a set of two routes around a block, such as $\{[1, 7, 8], [1, 2, 8]\}$. This is called a basic PAS. There are 25 basic PASs in the network. These basic PASs can be used to perform a flow shift between any two routes from origin A to destination F. For example, a flow shift between routes r and r' in figure 3.2 can be achieved by performing six flow shifts on basic

PASs in a specific order, as shown in table 3.1. Moreover, a basic PAS such as $\{[14, 20, 21], [14, 15, 21]\}$ can also be used for routes between all other OD pairs in the example. Therefore, the set of 25 basic PASs can be used to perform flow shifts between any two routes for all nine OD pairs.

As seen in the example it is possible to perform flow shifts between any two routes in the network using only a small set of PASs. Such a set is called a covering set. When all PASs in the covering set that have flow on both segments have segments of equal cost, all pairs of routes using any combination of these PASs have the same cost and UE is reached. In theory any covering PAS set can be used to reach equilibrium, but in practice some PASs move flow much more efficient than others. Therefore in each iteration the algorithm will evaluate the effectiveness of PASs, which will be covered in detail in section 3.3. So the main iteration scheme of the algorithm can be divided into two main parts: updating the PAS list by adding new PASs that lead to an improvement on the current solution and removing outdated or inefficient PASs, and shifting flow between the segments of the PASs to reduce the objective function. An overview of the general structure of TAPAS as presented in [3] is given in algorithm 1, with references to sections containing more information on each topic in parentheses.

The rest of this chapter is built up as follows: section 3.1 treats PAS removal and PAS construction, section 3.2 describes flow shifts, in section 3.3 conditions for effective PASs are treated. Section 3.4 describes branch shifts, which are needed when no effective new PAS can be found for a certain link. Section 3.5 describes the cyclic flow removal procedure. In section 3.6 the property of proportionality is explained and a method for calculating proportionality is described. Finally section 3.7 gives a formal proof for the convergence of the algorithm.

Used PAS	Resulting route	
	$A, 1, 2, 8, 14, 20, 21, 22, 28, 29, 30, 36, F$	(route r)
$\{[14, 20, 21], [14, 15, 20]\}$	$A, 1, 2, 8, 14, 15, 21, 22, 28, 29, 30, 36, F$	
$\{[15, 21, 22], [15, 16, 22]\}$	$A, 1, 2, 8, 14, 15, 16, 22, 28, 29, 30, 36, F$	
$\{[8, 14, 15], [8, 9, 15]\}$	$A, 1, 2, 8, 9, 15, 16, 22, 28, 29, 30, 36, F$	
$\{[9, 15, 16], [9, 10, 16]\}$	$A, 1, 2, 8, 9, 10, 16, 22, 28, 29, 30, 36, F$	
$\{[22, 28, 29], [22, 23, 29]\}$	$A, 1, 2, 8, 9, 10, 16, 22, 23, 29, 30, 36, F$	
$\{[16, 22, 23], [16, 17, 23]\}$	$A, 1, 2, 8, 9, 10, 16, 17, 23, 29, 30, 36, F$	(route r')

TABLE 3.1: Flow shift from route r to r' using basic PASs

Algorithm 1: Overview of TAPAS

```
Find an initial solution using an AON assignment
Define maximum allowed duality gap ϵ
Set iteration counter i = 1
Define maximum amount of iterations i_s
while Duality gap> ϵ and i <= i_s do
    for each PAS P do
        Remove if needed(3.1)
        Check for effectiveness(3.3)
    end
    for each origin p do
        Construct the Shortest Path Tree SPT_p for origin p using current flow
        solution
        for each link a where f_pa > 0 and a ∉ SPT_p do
            Construct new PAS or add p as relevant origin to an existing PAS
            (3.1)
        end
    end
    for each effective PAS do
        Shift flow for all relevant origins (3.2)
    end
    Remove cyclic flow(3.5)
    Perform proportionality iteration (3.6)
    Calculate new duality gap
    i++
end
while not proportionalized do
    Perform proportionality iteration (3.6)
end
```

3.1 PAS Construction

Updating the PAS set consists of three operations:

- remove unused PASs
- check PASs for cost and flow effectiveness (treated in section 3.3)
- constructing new PASs where needed

PAS removal is needed because PASs can be ineffective for several iterations or PASs can overlap in parts of segments, which may result in one of the PASs not being used. Removing PASs that have not been used for several iterations keeps the PAS set small and saves calculation time during the flow shift iterations and

effectiveness checks. In the current implementation a PAS is removed if it has not been used in the last three main iterations.

For PAS construction we first need to introduce the notion of reduced cost. The reduced cost rc_{pa} is defined for each origin link combination (p, a) as:

$$rc_{pa} = \pi_{pa_t} + c_a(\boldsymbol{f}) - \pi_{pa_h}$$

Here π_{pa_h} is the cost of the shortest route from origin p to node a_h when using all links is allowed, $\pi_{pa_t} + c_a(\boldsymbol{f})$ is the cost of the shortest path from p to a_h provided that link a is the last link of the path. The reduced cost can be interpreted as the penalty in travel cost that is charged for using link a to reach node a_h. The reduced cost is always non-negative and is only equal to zero when link a is part of the shortest path from p to a_h. In each iteration the algorithm checks for each origin p if PASs are needed. First the shortest path tree (SPT) from p to each node n is calculated using Dijkstra's algorithm. In a UE solution each link a that carries flow from origin p has reduced cost $rc_{pa} = 0$, so a PAS is needed when there is a link a that carries flow originating from origin p, but where $rc_{pa} > 0$. We call such a link a a critical link. The logical candidate of the low cost segment is (a part of) the current shortest path from p to a_h. The last link of the shortest path from p to a_h is called the shortest path alternative a' of critical link a. Therefore we visit all links to check if link a ending at node n has positive OB flow from origin p and has $rc_{pa} > 0$. We can achieve an improved solution by shifting flow from the critical link a to the shortest path alternative link a' also ending at node a_h. Node a_h will be the end node of the segments for the PAS, called the merge node. We construct a new PAS by backtracking from a_t over the links that carry OB flow from origin p using a breadth first search until a node is found that is part of the shortest path from p to a_h. We use a breadth first search to ensure that the segments will be as short as possible. At some point in the breadth first search we will meet a node m that is in the shortest path from p to a_h. This will be the begin node of the segments for the PAS, called the diverge node. Segment s_1 will consist of the links traversed in the backtracking search from node a_h via link a to node m, segment s_2 will consist of all links connecting node m to node a_h in the SPT.

We will clarify this method using the example in figure 3.3. Here the bold arrows represent links with flow and the shortest path tree is represented by the dashed arrows. The algorithm will visit all links to see if there are links that carry flow but are not part of the SPT. The first critical link is link $(5, 6)$. The SPT alternative is link $(2, 6)$. Now the algorithm will backtrack starting from

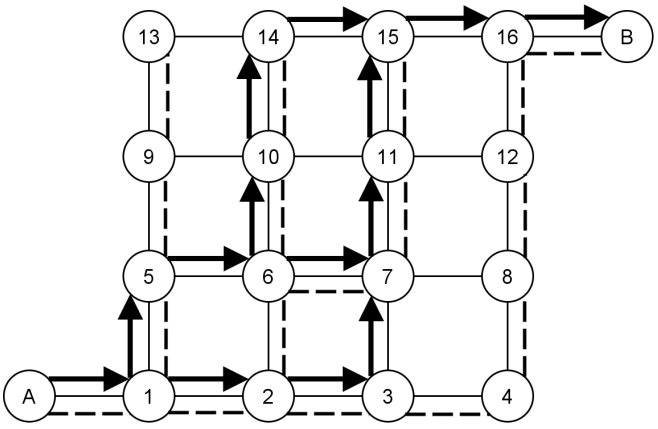

FIGURE 3.3: PAS Identification

node 6 over the flow carrying links until it meets a node that is part of the shortest path from A to 6. In this case there is only one possible route using the flow carrying links and the backtracking search will meet the shortest path at node 1, resulting in the PAS $\{[1, 5, 6], [1, 2, 6]\}$. Similarly, the critical link $(14, 15)$ will lead to the PAS $\{[6, 10, 14, 15], [6, 7, 11, 15]\}$. The advantage of using a breadth first search will be demonstrated by critical link $(15, 16)$. A naive backtracking procedure may result in a needlessly long PAS such as $\{[1, 5, 6, 7, 11, 15, 16], [1, 2, 3, 4, 8, 12, 16]\}$. The breadth first search visits the flow carrying links in the order $(14, 15)$, $(11, 15)$, $(10, 14)$, $(7, 11)$, $(6, 10)$, $(6, 7)$, $(3, 7)$. At this point a link contained in the shortest path from the origin to the merge node is found, resulting in the PAS $\{[3, 7, 11, 15, 16], [3, 4, 8, 12, 16]\}$.

On page 1030 of the original paper [3] the procedure for adding an origin as relevant to an existing PAS is described as follows. When a critical link a and its shortest path alternative a' are found, the algorithm searches the PAS list for PASs whose segments end with links a and a'. If such a PAS is found, the origin is added as relevant. This method is incomplete, as can be seen in figure 3.4. There is a positive flow on paths $[A, 3, 4, C]$ and $[B, 1, 2, 4, C]$ and the shortest paths are again given by the dashed lines. Let's assume that there currently is one PAS $\{[3, 4, C], [3, 2, C]\}$ for origin A and we are searching for a PAS for the critical link $(4, C)$ and shortest path alternative $(2, C)$ for origin B. According to the method described above, origin B can be added as relevant to existing PAS. However, links $(3, 4)$ and $(3, 2)$ can never carry flow from origin B, so this PAS can never be used for a flow shift for origin B. Therefore we add some extra conditions that must be satisfied by the PAS for an origin p to be added as relevant: link a should be part of the high cost segment s_1 and all other links in s_1 should carry flow from p. Also the PAS should be cost and flow effective for p, to be defined in section 3.3.

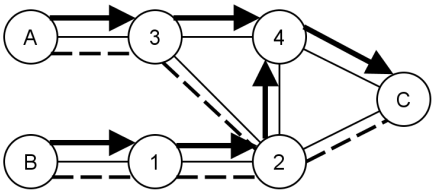

FIGURE 3.4: Origin incorrectly denoted as relevant

If there exists a PAS that satisfies these requirements, we can add p as a relevant origin to the PAS.

3.2 Flow Shifts

Flow shifts between segments of PASs are the easiest part of the algorithm. Only effective PASs are considered for flow shifts, so that the algorithm does not spend any time on ineffective flow shifts. For a flow shift on PAS P with segments s_1 and s_2 we first need to set segment s_1 as the segment with the highest cost. Then for each relevant origin $p \in O_P$ we determine the maximal flow f_p^* that can be shifted, so that all OB flows remain nonnegative:

$$f_p^* = \min_{a \in s_1} f_{pa}$$

The total amount of flow that can be shifted is:

$$f_{max} = \sum_{p \in O_P} f_p^*$$

If the maximum allowed amount of flow is shifted, we would get a new link flow solution $\boldsymbol{f}'$:

$$f'_a = \begin{cases} f_a - f_{max} & a \subseteq s_1 \\ f_a + f_{max} & a \subseteq s_2 \\ f_a & \text{otherwise} \end{cases}$$

If shifting all allowed flow still leaves segment s_2 as shortest segment we can simply shift all allowed flow. Otherwise, we determine the optimal step size $0 \leq \lambda \leq 1$ by performing a line search. We solve the problem

$$\min_{0 \leq \lambda \leq 1} \sum_{a \in A} \int_0^{(1-\lambda) f_a + \lambda f'_a} c_a(\omega) \, \mathrm{d}\, \omega \tag{3.2.1}$$

Flows on links that are not part of one of the segments of the PAS do not change, so solving equation 3.2.1 is equivalent to solving

$$\min_{0\leq\lambda\leq 1} \sum_{a\in s_1} \int_0^{f_a-\lambda f_{max}} c_a(\omega)\,\mathrm{d}\,\omega + \sum_{a\in s_2} \int_0^{f_a+\lambda f_{max}} c_a(\omega)\,\mathrm{d}\,\omega \tag{3.2.2}$$

This problem can be solved by taking the derivative and finding the root using for instance Newton's method. The last step is to update OB flows and link costs:

$$f_{pa} = f_{pa} - \lambda f_p^* \quad \forall\, a \in s_1$$

$$f_{pa} = f_{pa} + \lambda f_p^* \quad \forall\, a \in s_2$$

Empirical evidence shows that the PAS searching procedure takes much more time than the flow shifting procedure. Also, in later iterations when the PAS set is almost completely covering and the search for new PASs only introduces a few new PASs, it would be wise to spend more time on flow shifts. Bar-Gera proposes an inner loop for the flow shifts, as can be seen in the general structure of TAPAS in algorithm 1. Bar-Gera uses 20 flow shift inner loops per iteration in his implementation. Our prototype uses a slightly different approach: less time is spent on flow shifts in early iterations, when the solution is far from being equilibriated. A substantial amount of flow that is shifted during the early iterations will be shifted away from the segments of the existing PASs when new ones are found in later iterations. Therefore the number of inner loops is chosen to be $\min(i, 20)$, where i is the number of iterations, so that the amount of flow shifts is increased gradually.

3.3 Cost and Flow Effectiveness of PASs

As was pointed out earlier, flow shifts between any pair of routes can be accomplished by a covering PAS set. However, during the execution of the algorithm, flows and link costs may change in such a way that existing PASs may lose their efficiency. There are two major causes for this to happen: either the cost difference between the segments is very small and only a small amount of flow can be shifted before segment costs are equal, or the amount of flow on one of the links of the high cost segment is very small, so that shifting all allowed flow still leaves a big cost difference between the critical link and its shortest path alternative. These two cases, called cost effectiveness and flow effectiveness respectively are treated in the original paper and solutions for each case are proposed.

A PAS is cost effective for origin p if the cost difference between the segments is large enough. To assess this formally, we again need the reduced cost for the last link a of segment s_1 of the PAS, with a_h as the merge node of the PAS. As a reminder, the reduced cost rc_{pa} is defined for each origin link combination as:

$$rc_{pa} = \pi_{pa_t} + c_a(\boldsymbol{f}) - \pi_{pa_h}$$

The reduced cost can be seen as the minimal improvement in travel time that could be made by a traveler if the traveler stops using the current route that approaches node a_h using link a and instead uses the shortest path from p to a_h. If a PAS with link a as the last link of the high cost segment has a cost difference that is smaller than a factor $\mu = 0.5$ of the reduced cost, then both segments will have equal cost when only a small fraction of the maximum allowed flow is shifted. This is a problem especially when the PAS is part of a sequence of PASs that is used to shift flow from one path to another. In this case a lot of iterations are needed to shift large amounts of flow from one path to the other as can be seen in the example below. The PAS is considered not cost effective and a new PAS has to be found with a more efficient low cost segment. Formally, a PAS, given by s_1 and s_2, with segment s_1 as high cost segment and link a as last link of s_1 is considered cost effective for origin p if

$$c_{s_1} - c_{s_2} \geq \mu \cdot rc_{pa}$$

For the flow effectiveness of a PAS for origin p we look at the amount of flow f_{pa} on the last link a of segment s_1. We want to be able to shift all or most of the flow from segment s_1 to segment s_2. However, the nonnegativity constraints restrict the maximal flow shift for origin p to the minimal flow $f_p^* = \min_{a \in s_1} f_{pa}$ on s_1. If f_p^* is small, we can only shift a very small amount of flow. If this PAS is part of a chain, we get a so called cascading effect, as can be seen in the example below. To counter this phenomenon, we define a PAS to be flow effective if the ratio between the minimal flow f_p^* and the flow f_{pa} on the last link of s_1 is greater than a factor $\nu = 0.25$. We formally define a PAS with segment s_1 as high cost segment and link a as last link of s_1 to be flow effective for origin p if

$$f_p^* \geq \nu \cdot f_{pa}$$

If a PAS is not flow effective, we have to find a new PAS that uses another high cost segment.

We can show the problems that arise when flow is shifted using ineffective PASs

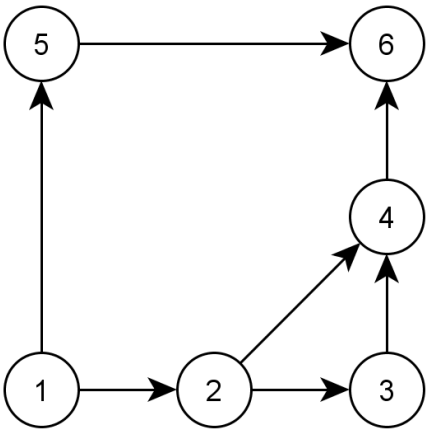

FIGURE 3.5: Example of ineffective PASs

with the network shown in figure 3.5 where flow is sent from origin 1 to destination 6. There are three possible PASs: $P_1 = \{[1,5,6],[1,2,3,4,6]\}$, $P_2 = \{[1,5,6],[1,2,4,6]\}$ and $P_3 = \{[2,3,4],[2,4]\}$. Any two of these PASs forms a covering PAS set and be used to perform all flow shifts between routes in this network. Assume that PAS P_2 and P_3 are used and links have the following costs: $c_{(1,5)} = c_{(5,6)} = 2$, $c_{(1,2)} = c_{(4,5)} = 1$, $c_{(2,3)} = c_{(3,4)} = 0.5$ and $c_{(2,4)} = (2-m) + f_{(2,4)}$ where m is an arbitrary number between 0 and 1. Also assume that there is a flow of 1 on route $[1,5,6]$, the flow is zero everywhere else. In the flow shift procedure first PAS P_2 is used to shift m units of flow from segment $[1,5,6]$ to $[1,2,4,6]$. Now both segments have equal costs and PAS P_3 can be used to shift m units of flow to segment $[2,3,4]$. This sequence will be repeated until all flow is shifted to route $[1,2,3,4,6]$. When m is chosen close to 0, this procedure could in theory take infinitely many iterations. In this case PAS P_2 is not cost effective, because the reduced cost of link $(5,6)$ is 1, and $c_{s1} - c_{s2} = m \leq \mu \cdot 1$. The algorithm will stop using PAS 2 and add PAS 1 which will find the UE solution in one flow shift.

Now we change link costs to 1 for all links except link $(2,4)$, which will have cost $c_{(2,4)} = (2-m)+f_{(2,4)}$. There is a flow of 1 on path $[1,2,3,4,6]$ and zero everywhere else. Even though both PASs are cost effective we get a similar behavior: first PAS P_3 is used to shift m units of flow to segment $[2,4]$ and then PAS P_2 is only allowed to shift a maximum of m units of flow to segment $[1,5,6]$, because otherwise the non-negativity constraints are violated. The sequence is then repeated until all flow is shifted. This process can take arbitrarily long if we choose m small enough. PAS P_2 is not flow effective and the algorithm will use PAS P_1 instead.

There are two possible ways to implement cost and flow effectiveness. In the first method effectiveness is determined on PAS level: when one or more relevant origins satisfy the effectiveness requirements for both effectiveness measures, the whole

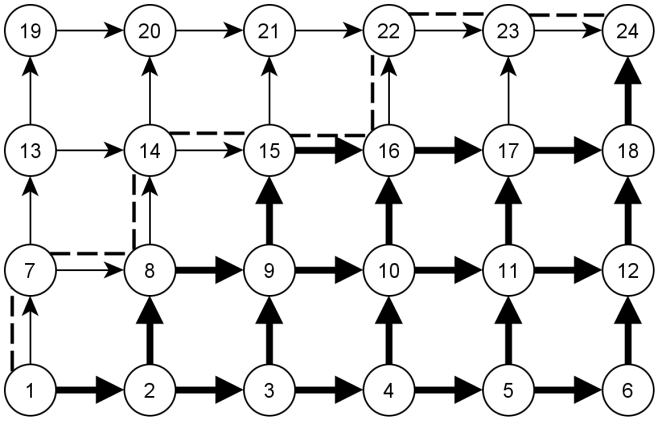

FIGURE 3.6: Example of a branch

PAS is declared effective. Flow shifts are then performed on all relevant origins. In the second method effectiveness is determined separately for each relevant origin. Using this method flow shifts are only performed on the relevant origins that are both cost and flow effective. If a relevant origin fails to satisfy one or both of the requirements the PAS is declared not effective for that relevant origin and a new PAS has to be found for the origin. In the original paper this issue is not addressed, but in our prototype we choose to use the second method. This is because in the first method a PAS may not meet both effectiveness requirements for some relevant origins, while it may still be considered effective because there is another relevant origin that does meet both requirements. This can severely slow down convergence for the noneffective relevant origins, as shown in the example in figure 3.5. In such a case we are better of searching a new PAS for these origins.

3.4 Branch Shift

For a new PAS for origin p with critical link a it always holds that $c_{s_1} - c_{s_2} \geq rc_{pa}$, so any new PAS generated by the algorithm is always cost effective. This may not be the case for flow effectiveness. New PASs are checked for flow effectiveness and if a new PAS is not flow effective, the breadth first search will continue until another diverge node is found. In later iterations, when flow is highly spread out, there is a possibility that no flow effective PAS exists for critical link a. This is a scenario that is likely to happen, since TAPAS attempts to maintain a proportionalized flow solution (described in section 3.6). In this case a new mechanic called a branch shift is performed. Unlike a flow shift where flow is shifted from one route segment to another, in a branch shift flow is shifted from all route segments from p to a_h that carry flow from origin p simultaneously. The set of links that are part

of a route segment from an origin p to the merge node a_h that carries flow from p and that has the critical link a as last link is called a branch. An example of a branch is shown in figure 3.6. Here the critical link is link $(18, 24)$. The least cost route from origin 1 to node 24 is represented by the dashed line. All links that are part of any flow carrying route segment that ends with the critical link are in bold. Note that this is not necessary all flow to node 24; there may be flow that enters node 24 through link $(23, 24)$. No description of the calculation of a branch shift is given in [3], so an original method is stated here. We use S to denote the set of all flow carrying route segments that start at p and enter a_h through a. In a branch shift flow is shifted from all route segments in S. The maximal allowed flow shift is the OB flow on the critical link: $f_p^* = f_{pa}$. Using the segment flows for each route segment s in S, we can calculate the proportion $pr_{pa'}$ of f_p^* that passes through each link a' in the branch. We call this proportion the branch proportion of link a'. We can calculate $pr_{pa'}$ by dividing the sum of the segment flows on all segments in S that contain a' by the max flow shift. The flow on each segment s in S can be calculated using

$$k_{ps} = \left(\prod_{b \in s} \alpha_{pb} \right) \cdot f_p^*$$

So the proportion of branch flow that is on link a' is

$$pr_{pa'} = \frac{\sum_{(s \in S | a' \in s)} k_{ps}}{f_p^*} \tag{3.4.1}$$

$$= \frac{\sum_{(s \in S | a' \in s)} \left(\prod_{b \in s} \alpha_{pb} \right) \cdot f_p^*}{f_p^*} \tag{3.4.2}$$

$$= \sum_{(s \in S | a' \in s)} \prod_{b \in s} \alpha_{pb} \tag{3.4.3}$$

This formula requires that all possible path segments in the branch are calculated. Alternatively, we can calculate the branch proportion recursively by setting the branch proportion to $pr_a = 1$ for the critical link and to $pr_b = 0$ for all links b that are not part of the branch. We can then traverse backward through the branch using

$$pr_{pa'} = \alpha_{pa'} \cdot \sum_{a_{suc} \in OUT_{a'_h}} pr_{a_{suc}}$$

This way we only need to visit each link once. We can then calculate the optimal step size λ using

$$\min_{0\leq\lambda\leq 1} \sum_{a\in A} \int_0^{f_{pa}+\delta_{a,spt}\lambda f_p^* - \delta_{a,b}\lambda pr_a f_p^*} c(\omega)\mathrm{d}\omega$$

where $\delta_{a,spt}$ is 1 if a is part of the shortest path segment and 0 otherwise, $\delta_{a,b}$ is 1 if a is part of the branch and 0 otherwise. The flow update is performed using

$$f_{pa',new} = f_{pa'} + \delta_{a',spt}\lambda f_p^* - \delta_{a',b}\lambda\, pr_{a'} f_p^*$$

3.5 Cyclic Flow Removal

Cyclic flow can be generated in the network due to the structure of PAS flow shifts. This flow has to be eliminated from the network, because UE can not be achieved when there is cyclic flow present on the network. Several algorithms that identify cycles in networks exist. The most efficient of these algorithms is Tarjan's strongly connected components algorithm, which has to be altered slightly in order to be able to identify cycles instead of connected component. A connected component is a set of nodes $N' \subseteq N$ such that each node $n \in N'$ can be reached by all other nodes $\{n' \in N' | n' \neq n\}$. A connected component can contain multiple cycles and some edges can be included in more than on cycle. Pseudocode for the adapted version of Tarjan's algorithm can be found in appendix B. This cyclic flow removal algorithm differs from Tarjan's algorithm in that this version visits links instead of nodes. Another difference is that instead of finding a complete connected component, this algorithm stops searching when a cycle is found and removes cyclic flow immediately. After removing the cyclic flow all links in the cycle are flagged as unvisited so that the procedure can visit these links again in case they are part of more cycles. The procedure resumes the search for cycles at the link of the cycle it first encountered. If a flow cycle $C = [a_1, a_2, \ldots, a_i]$ is found for origin p, we determine the total flow to be removed

$$f_{max} = \min_{a\in C} f_{pa} \tag{3.5.1}$$

The OB flows will be updated using

$$f_{pa,new} = f_{pa} - f_{max} \tag{3.5.2}$$

Now at least one link a in C has OB flow $f_{pa} = 0$, so the cycle has been removed.

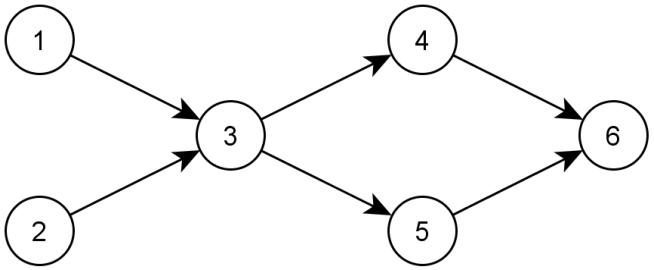

FIGURE 3.7: Proportionality example

3.6 Proportionality

While the User Equilibrium conditions guarantee a unique optimal solution of Beckmanns problem (BE) in terms of link flows, this is not the case for route flows (see remark r1 in section 2.3). When there are OD pairs that have a common pair of flow carrying segments, there are infinitely many route flow solutions, as can be seen in the example in figure 3.7. Here the flow demand is 1 for OD pairs $(1, 6)$ and $(2, 6)$ and each link carries one unit of flow in the UE solution. Each OD pair has two UE routes that have a common pair of segments $s_1 = [3, 4, 6]$ and $s_2 = [3, 5, 6]$. Now any feasible OB flow that has segment flows $k_{1,s_1} = 1 - k_{2,s_1}$ and $k_{1,s_2} = 1 - k_{2,s_2}$ is a UE solution. TAPAS tries to find a realistic route solution $\boldsymbol{h}$ by introducing two properties for route flow solutions: route consistency and route proportionality. A UE solution is called consistent if for all OD pairs $\{p, q\}$ all UE routes, i.e. all routes r between p and q with $f_a > 0 \, \forall a \in r$, have route flow $h_r > 0$. This condition is a natural extention to the User Equilibrium principles: users are indifferent between UE routes, so all of these routes are used by some portion of the flow. For the proportionality condition we first introduce the notion of alternative routes. Let r_1 be a route from origin p_1 to destination q_1 and let r_2 be a route from p_2 to q_2. Assume PAS P with segments s_1, s_2 has equal cost for both segments and let node d and m be the diverge node and merge node of the segments of P. The origin based proportion of a PAS is defined as

$$\rho_{p,s_1} = \frac{k_{p,s_1}}{k_{p,s_1} + k_{p,s_2}}$$

Routes r_1 and r_2 are called a pair of alternative routes if they consist of the following route segments:

$$r_1 = [r_{p_1,d}, s_1, r_{m,q_1}]$$
$$r_2 = [r_{p_2,d}, s_2, r_{d,q_2}]$$

Here $r_{p_1,d}$ is the route segment from p_1 to the diverge node of PAS P and r_{m,q_1} is the route segment from the merge node of PAS P to q_1. We define the route proportion ρ_{r_1,r_2} as

$$\rho_{r_1,r_2} = \rho_{p,s_1}$$

We call a UE solution proportional if for any two pairs of alternative routes (r_1, r_2), (r_3, r_4) that use PAS P, with routes r_1 and r_3 using segment s_1 and routes r_2 and r_4 using segment s_2, we have that $\rho_(r_1, r_2) = \rho_(r_3, r_4)$.

Finding a completely consistent UE solution is difficult, but TAPAS tries to satisfy the consistency condition by adding many relevant origins to a PAS. This way, each relevant origin will carry flow on both segments of the PAS by the proportionality condition. It is well-known that for given UE $(\boldsymbol{f}^*, \boldsymbol{h})$ with uniquely determined link flow $\boldsymbol{f}^*$ the corresponding pathflow h satisfies consistency and the proportionality conditions if (for given $\boldsymbol{f}^*$) the part $\boldsymbol{h}$ is a maximizer of the following entropy maximization problem (see Larsson et al. [24]):

$$\begin{aligned}
\max \quad & \boldsymbol{E}(\boldsymbol{h}) = -\sum_{r \in \boldsymbol{R}} h_r \cdot \log \frac{h_r}{d_{pq}} \\
\text{s.t.} \quad & \sum_{r \in R_{pq}} h_r = d_{pq} \qquad \forall p \in N_o, q \in N_d(p) \\
& \sum_{r \in \boldsymbol{R}: a \subseteq r} h_r = f_a^* \qquad \forall a \in A \\
& \boldsymbol{h} \geq 0
\end{aligned}$$

TAPAS solves the proportionality problem approximately on PAS level by iteratively applying a flow shift between the relevant origins of a PAS that keeps the total link flows identical and maximizes entropy for the PAS, as described in Bar-Gera [3]. We want to find the vector of origin-based PAS adjustments Δf_{pa} such that

$$\Delta f_{pa} = \begin{cases} \delta_p & a \subseteq s_1 \\ -\delta_p & a \subseteq s_2 \\ 0 & \text{otherwise} \end{cases}$$

We can find Δf_{pa} by calculating the solution to the maximum entropy problem for a single PAS. Here $\boldsymbol{f}$ is the vector of the OB flows f_{pa}:

$$\begin{aligned} \max \quad & \boldsymbol{E}(\boldsymbol{f}+\Delta\boldsymbol{f}) = -\sum_{p\in N_o}\sum_{a\in A}(f_{pa}+\Delta f_{pa})\cdot\log\left(\frac{f_{pa}+\Delta f_{pa}}{g_{pa_h}(\boldsymbol{f}+\Delta\boldsymbol{f})}\right) \\ \text{s.t.} \quad & \sum_p \delta_p = 0 \\ & \boldsymbol{f}+\Delta\boldsymbol{f} \geq 0 \end{aligned}$$

Here the first constraint ensures that the total link flow on each link does not change. The second constraint ensures that all OB flows remain nonnegative. The Lagrangian for this problem is

$$\max \mathcal{L} = \boldsymbol{E}(\boldsymbol{f}+\Delta\boldsymbol{f}(\boldsymbol{\delta})) + \lambda\cdot\left(\sum_p \delta_p\right) \tag{3.6.1}$$

$$\text{s.t.} \quad \boldsymbol{f}+\Delta\boldsymbol{f}(\boldsymbol{\delta}) \geq 0 \tag{3.6.2}$$

To find the derivative of equation 3.6.1 with respect to δ_p we look at each origin link combination separately:

$$E_{pa}(\boldsymbol{f}+\Delta\boldsymbol{f}(\boldsymbol{\delta})) = -(f_{pa}+\Delta f_{pa}(\boldsymbol{\delta}))\cdot\log\left(\frac{f_{pa}+\Delta f_{pa}(\boldsymbol{\delta})}{g_{pa_h}(\boldsymbol{f}+\Delta\boldsymbol{f}(\boldsymbol{\delta}))}\right) \tag{3.6.3}$$

$$\boldsymbol{E}(\boldsymbol{f}+\Delta\boldsymbol{f}(\boldsymbol{\delta})) = \sum_{p\in N_o}\sum_{a\in A} E_{pa}(\boldsymbol{f}+\Delta\boldsymbol{f}(\boldsymbol{\delta})) \tag{3.6.4}$$

First we look at the derivative of the last links of the segments $a_1 \subseteq s_1$ and $a_2 \subseteq s_2$, with merge node $n = a_{1_h} = a_{2_h}$. Here we have that $g_{pn}(\boldsymbol{f}+\Delta\boldsymbol{f}(\boldsymbol{\delta})) = g_{pn}$, because the adjustment on s_1 cancels out the adjustment on s_2. So we get

$$\begin{aligned} \frac{\partial E_{pa_1}}{\partial\delta_p} &= -\log\left(\frac{f_{pa_1}+\delta_p}{g_{pn}}\right) - 1 \\ \frac{\partial E_{pa_2}}{\partial\delta_p} &= \log\left(\frac{f_{pa_2}+\delta_p}{g_{pn}}\right) + 1 \end{aligned}$$

All other OB flows on links that have n as head node are unchanged, so the total contribution to the derivative of links that end at the merge node is

$$\sum_{a|a_h=n}\frac{\partial E_{pa}}{\partial\delta_p} = -\log\left(\frac{f_{pa_1}+\delta_p}{g_{pn}}\right) + \log\left(\frac{f_{pa_2}-\delta_p}{g_{pn}}\right) \tag{3.6.5}$$

For other links $a'|a'_h = j \neq n$ in s_1 the change in origin based link flow $f_{pa'}$ and in the origin based node flow g_{pj} is δ_p. So the contribution to the derivative of origin

link combination pa' is

$$\frac{\partial E_{pa'}}{\partial \delta_p} = -\log\left(\frac{f_{pa'} + \delta_p}{g_{pj} + \delta_p}\right) - 1 + \frac{f_{pa'} + \delta_p}{g_{pj} + \delta_p} \tag{3.6.6}$$

All other links a'' that end at node j also contribute to the derivative, because the node flow g_{pj} is changed. The contribution of these links is

$$\frac{\partial E_{pa''}}{\partial \delta_p} = \frac{f_{pa''}}{g_{pj} + \delta_p} \tag{3.6.7}$$

Adding equations 3.6.6 and 3.6.7 together gives the total contribution of links ending at j:

$$\sum_{a|a_h=j} \frac{\partial E_{pa}}{\partial \delta_p} = -\log\left(\frac{f_{pa'} + \delta_p}{g_{pj} + \delta_p}\right) \tag{3.6.8}$$

Along the same line we can get the contribution to the derivative of a link $a' \subseteq s_2$ and all other links that end at node $a'_h = j \neq n$

$$\sum_{a|a_h=j} \frac{\partial E_{pa}}{\partial \delta_p} = -\log\left(\frac{f_{pa'} - \delta_p}{g_{pj} - \delta_p}\right) \tag{3.6.9}$$

We can get the total entropy derivative by adding the derivative components of each node in the PAS using equations 3.6.5, 3.6.8 and 3.6.9:

$$\begin{aligned}
\frac{\partial E}{\partial \delta_p} &= -\sum_{a \subseteq s_1} \log\left(\frac{f_{pa} + \delta_p}{g_{pa_h}(\delta_p)}\right) + \sum_{a \subseteq s_2} \log\left(\frac{f_{pa} - \delta_p}{g_{pa_h}(\delta_p)}\right) \\
&= -\log\left(\frac{k_{ps_1}(\boldsymbol{\delta})}{g_{ps_{1h}}(\boldsymbol{\delta})}\right) + \log\left(\frac{k_{ps_2}(\boldsymbol{\delta})}{g_{ps_{2h}}(\boldsymbol{\delta})}\right) \\
&= -\log\left(\frac{k_{ps_1}(\boldsymbol{\delta})}{k_{ps_2}(\boldsymbol{\delta})}\right)
\end{aligned}$$

Here $k_{ps}(\boldsymbol{\delta})$ is the flow on segment s from origin p given the adjustment $\boldsymbol{\delta}$. So the optimality conditions of 3.6.1 are:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \delta_p} &= -\log \frac{k_{ps_1}(\delta_p)}{k_{ps_2}(\delta_p)} + \lambda = 0 \\
\frac{k_{ps_1}(\delta_p)}{k_{ps_2}(\delta_p)} &= \exp(\lambda)
\end{aligned}$$

An equivalent system for the unknowns δ_p, λ is

$$\rho_p(\delta_p) := \frac{k_{ps_1}(\delta_p)}{k_{ps_1}(\delta_p) + k_{ps_2}(\delta_p)} = (1 + \exp(-\lambda))^{-1} \qquad \forall p \in O_P$$

In [3] Bar-Gera proposes a method for solving this system of KKT-conditions for δ_p, λ subject to the constraint $f + \Delta(f) \geq 0$.

3.7 Proof of Convergence

This section gives a formal proof of convergence for TAPAS. This proof is along the lines of the proof given by Bar-Gera in section 6.4 of [3], but is worked out in greater detail. For this proof we use a simplified version of TAPAS, shown in algorithm 2, where in each iteration only critical origin-link combinations (p, a) that have $\xi(f_{pa}, rc_{pa}) > \epsilon$ are considered. This ϵ is halved after each iteration and the algorithm stops if ϵ becomes smaller than a given ϵ_s. For each critical origin-link combination either a flow shift or a branch shift is performed. This algorithm is different from the normal version of TAPAS, but it is easier to prove convergence for. Equilibrium is reached if ϵ reaches 0, i.e. if there are no more critical origin-link combinations. In this case, for each origin p and for all flow carrying links a the reduced cost $rc_{pa} = 0$, so all flow carrying links are part of the shortest path tree of p. We show that we can find a lower bound on the reduction of the objective function for flow shifts and branch shifts for the critical link. For the proof we assume that the cost functions $c_a(\boldsymbol{f})$ are Lipschitz continuous with Lipschitz constant $\mathcal{L}$:

$$|c_a(\boldsymbol{f} + \delta) - c_a(\boldsymbol{f})| \leq \mathcal{L}\delta \quad \forall a, \text{ feasible } \boldsymbol{f} \tag{3.7.1}$$

This is not a strong condition, because demands are finite. Furthermore we use the objective function $z(\boldsymbol{f})$ given in equation 2.3.5, the effectiveness definitions given in section 3.3, the reduced cost defined in section 3.1 and $\xi(f_{pa}, rc_{pa})$, which is defined as

$$\xi(f_{pa}, rc_{pa}) := \min\left\{\frac{\mu^2 rc_{pa}^2(\boldsymbol{f})}{8\mathcal{L}|A|}, \frac{\mu\nu}{2} f_{pa} rc_{pa}(\boldsymbol{f})\right\} \tag{3.7.2}$$

First we prove that a flow shift on an efficient PAS reduces the objective function by at least $\xi(f_{pa}, rc_{pa})$.

Lemma 3.1. *Let $\boldsymbol{f}^k \to \boldsymbol{f}^{k+1}$ be the result of a flow shift for an efficient PAS for origin p and critical link a. Then $z(\boldsymbol{f}^{k+1}) \leq z(\boldsymbol{f}^k) - \xi(f_{pa}, rc_{pa})$*

Algorithm 2: Simplified version of TAPAS used for convergence proof

Initialize: set $\epsilon = \epsilon_0$, $\epsilon_s \ll \epsilon_0$, find a feasible flow $\boldsymbol{f}^0$, set μ and ν as described in 3.3
Set $k = 0$
while $\epsilon \geq \epsilon_s$ **do**
  while *for* $\boldsymbol{f}^k$, $\exists a, p$ *such that* $\xi(f^k_{pa}, rc_{pa}) \geq \epsilon$ **do**
    if $\exists$ *an efficient PAS* P *for* p *and* a **then**
      Perform a flow shift for P
    end
    else
      Perform a branch shift for p and a
    end
    $k++$
    Set $\boldsymbol{f}^k$ as the new flow vector
  end
  $\epsilon = \epsilon/2$
end

Proof. Let $\delta > 0$ be the amount of flow shifted on the PAS. Then

$$\begin{aligned} f'_{pa} &= f_{pa} - \delta \quad \forall a \in s_1 \\ f'_{pa} &= f_{pa} + \delta \quad \forall a \in s_2 \\ f'_{pa} &= f_{pa} \quad \forall a \in A \setminus (s1 \cup s2) \end{aligned}$$

We have that

$$-\mathcal{L}|\delta| \leq c_a(f+\delta) - c_a(f) \leq \mathcal{L}|\delta| \quad \forall a, f \tag{3.7.3}$$

By the mean value theorem we get that for each link $a \in s_1$ for some $\tilde{f}_a \in (f_a - \delta, f_a)$

$$\begin{aligned} \int_{f_a-\delta}^{f_a} c_a(\omega)\mathrm{d}\omega &= c_a(\tilde{f}_a)\delta && (3.7.4)\\ &\geq \delta(c_a(f_a) - \mathcal{L}|\tilde{f}_a - f_a|) && (3.7.5)\\ &\geq \delta(c_a(f_a) - \mathcal{L}\delta) && (3.7.6) \end{aligned}$$

and similarly for each link $b \in s_2$

$$\begin{aligned} \int_{f_b}^{f_b+\delta} c_b(\omega)\mathrm{d}\omega &= c_b(\tilde{f}_b)\delta && (3.7.7)\\ &\leq \delta(c_b(f_b) + \mathcal{L}|\tilde{f}_b - f_b|) && (3.7.8)\\ &\leq \delta(c_b(f_b) + \mathcal{L}\delta) && (3.7.9) \end{aligned}$$

So the difference in the objective function is, using 3.7.6 and 3.7.9

$$\Delta := z(f) - z(f') = z_{s_1}(f) + z_{s_2}(f) - z_{s_1}(f') - z_{s_2}(f') \tag{3.7.10}$$

$$= \sum_{a \in s_1} \int_{f_a - \delta}^{f_a} c_a(\omega) \mathrm{d}\omega - \sum_{b \in s_1} \int_{f_b}^{f_b + \delta} c_b(\omega) \mathrm{d}\omega \tag{3.7.11}$$

$$\geq \delta(c_{s_1}(\boldsymbol{f}) - c_{s_2}(\boldsymbol{f}) - 2|A|\delta\mathcal{L}) \tag{3.7.12}$$

$$\geq \delta(\mu r c_{pa}(\boldsymbol{f}) - 2|A|\delta\mathcal{L}) \tag{3.7.13}$$

Here the last inequality is due to the fact that the PAS is cost effective. Now define

$$\delta^* := \frac{1}{2} \frac{\mu \, r c_{pa}(\boldsymbol{f})}{2|A|\mathcal{L}} \tag{3.7.14}$$

We can choose our flow shift $\delta = \min\{\delta^*, f_p^*\}$, where $f_p^* = \min_{b \in s_1} f_{pa}$. Here we have that $f_p^* \geq \nu f_{pa}$, due to the fact that the PAS is flow effective. If $\delta^* \leq \nu f_{pa}$, a flow shift of δ^* is feasible and yields

$$\Delta \geq \delta^* \frac{1}{2} \mu \, r c_{pa}(\boldsymbol{f}) = \frac{\mu^2 \, r c_{pa}^2(\boldsymbol{f})}{8|A|\mathcal{L}} \tag{3.7.15}$$

Otherwise we can shift νf_{pa} amounts of flow, yielding

$$\Delta \geq \nu f_{pa} \frac{\mu \, r c_{pa}(\boldsymbol{f})}{2} \tag{3.7.16}$$

□

We will now show that a similar reduction in objective function can be achieved by a branch shift.

Lemma 3.2. *The reduction Δ in the objective function z of a branch shift is*

$$\Delta \geq \min \left\{ f_{pa} \frac{r c_{pa}(\boldsymbol{f})}{2}, \frac{r c_{pa}^2(\boldsymbol{f})}{8|A|\mathcal{L}} \right\} \tag{3.7.17}$$

Proof. Let a be a critical link for p with respect to flow $\boldsymbol{f}$ and B_p be the branch spanned by all segments $s^1, .., s^k$ from p to a_h through a. Suppose s_2 is a shortest path segment from p to a_h. By definition we have for all s^i

$$c_{s^i}(\boldsymbol{f}) - c_{s_2}(\boldsymbol{f}) \geq r c_{pa}(\boldsymbol{f}) \tag{3.7.18}$$

Let

$$f_p^*(i) := \min_{b \in s_i} f_{pb} \tag{3.7.19}$$

Now since all flow f_{pa} comes via one of the segments $s^1, .., s^k$ in B_p we have that

$$\sum_{i=1}^{k} f_p^*(i) \geq f_{pa} \tag{3.7.20}$$

So we can choose flows f_p^i for each segment in B such that $f_p^i \leq f_p^*(i)$ and $\sum_i f_p^i = f_{pa}$. Now consider simultaneous flow shifts δ_i from s^i to s_2 denoted by $\boldsymbol{f} \to \boldsymbol{f}'$. We define $\delta = \sum_i \delta_i$. By the analysis used in the proof of lemma 3.1 and using equation 3.7.18 we find the difference in objective function values

$$\begin{aligned}
\Delta &:= z(\boldsymbol{f}) - z(\boldsymbol{f}') && (3.7.21)\\
&= \sum_i \sum_{a \in s^i} \int_{f_a - \delta_i}^{f_a} c_a(\omega) \mathrm{d}\omega - \sum_{b \in s_2} \int_{f_b}^{f_b + \delta} c_b(\omega) \mathrm{d}\omega && (3.7.22)\\
&\geq \sum_i \delta_i (c_{s^i}(\boldsymbol{f}) - |A|\mathcal{L}\delta_i) - \delta(c_{s_2}(\boldsymbol{f}) + |A|\mathcal{L}\delta) && (3.7.23)\\
&= \sum_i \delta_i [(c_{s^i}(\boldsymbol{f}) - c_{s_2}(\boldsymbol{f})) - 2|A|\mathcal{L}\delta] && (3.7.24)\\
&\geq \sum_i \delta_i (r c_{pa}(\boldsymbol{f}) - 2|A|\mathcal{L}\delta) && (3.7.25)
\end{aligned}$$

In order to satisfy nonnegativity constraints, we can choose flow shifts δ_i such that

$$\delta_i \leq f_p^i, \quad \delta \leq \delta^* := \frac{1}{2} \frac{r c_{pa}(\boldsymbol{f})}{2|A|\mathcal{L}} \tag{3.7.26}$$

In case $\sum_i f_p^i \geq \delta^*$ we can choose $\delta_i \leq f_p^i$ such that $\sum_i \delta_i = \delta^*$ and we can shift δ_i from s^i to s_2 to obtain

$$\begin{aligned}
\Delta &\geq \sum_{i=1}^{k} \delta_i \frac{r c_{pa}(\boldsymbol{f})}{2} && (3.7.27)\\
&= \delta^* \frac{r c_{pa}(\boldsymbol{f})}{2} && (3.7.28)\\
&= \frac{r c_{pa}(\boldsymbol{f})}{8|A|\mathcal{L}} && (3.7.29)
\end{aligned}$$

In case $\sum_i f_p^i \leq \delta^*$ we can choose $\delta_i = f_p^i$ and perform this shift to obtain

$$\begin{aligned}
\Delta &\geq \sum_{i=1}^{k} f_p^i \frac{r c_{pa}(\boldsymbol{f})}{2} && (3.7.30)\\
&= f_{pa} \frac{r c_{pa}(\boldsymbol{f})}{2} && (3.7.31)
\end{aligned}$$

□

Now we arrive at the convergence theorem, which follows from lemmas 3.1 and 3.2.

Theorem 3.3. *Let $\epsilon_s = 0$, let $\boldsymbol{f}^k$ be the sequence of feasible flows generated by algorithm 2 and let $z(\boldsymbol{f}^k)$ be the sequence of objective functions induced by $\boldsymbol{f}^k$. Then the sequence $z(\boldsymbol{f}^k)$ converges to the minimum value of Beckmann's program (BE). Moreover the sequence $\boldsymbol{f}^k$ has a (at least one) limit point $\boldsymbol{f}^*$. Each such limit point $\boldsymbol{f}^*$ is a UE.*

Proof. Since the sequence $z(\boldsymbol{f}^k)$ is monotonically decreasing we only have to show that $\boldsymbol{f}^*$ is a UE. Suppose that $\boldsymbol{f}^*$ is not a UE. Then the value $\xi(f^*_{pa}, rc_{pa}(\boldsymbol{f}^*)) := \sigma > 0$. But then according to algorithm 2 in each step k of this algorithm we have $\xi(f^k_{pa}, rc_{pa}(\boldsymbol{f}^*)) \geq \delta/2$ and by lemma 3.1 and 3.2 in each step the value of $z(\boldsymbol{f}^k)$ must be reduced by $\delta/2$. Since we have done infinitely many steps this gives a contradiction since by assumption a minimizer of (BE) exists, i.e. the minimum value of z is bounded. □

Chapter 4

Traffic Assignment Problem with Asymmetric Costs

Introducing asymmetric cost functions to TAP makes the problem much more difficult to solve efficiently, as shown in section 2.4. This chapter describes how OmniTRANS deals with junction modeling and proposes several methods that can be added to TAPAS in order to be able to handle asymmetric cost functions. Section 4.1 explains how junction modeling is treated in OmniTRANS and shows the mathematical difficulties that arise from asymmetrical cost functions. Section 4.2 describes the changes that need to be made to incorporate junction modeling in TAPAS. In section 4.3 some heuristics for finding a descent direction using the asymmetric cost functions are treated.

4.1 Junction Modelling in OmniTRANS

In OmniTRANS junction delays are modeled by expanding each junction node as can be seen in figure 4.1. Each arm of the junction is represented by a node and for each possible turn a link between two arms is added, called a turn. Since OmniTRANS supports both three way junctions and four way junctions, each junction node will be blown up to at most four nodes and twelve links. The links that are added when junction nodes are expanded are called turns. When turns are added we divide the set A in two sets: the set of turns, called A_t and the set of regular links, called A_l, such that $A \setminus A_l = A_t$. OmniTRANS has put several restictions on paths containing turns: a link entering an junction must be followed by a turn of that junction and this turn has to be followed by an exit link of the junction. Illegal turn movements such as traveling along a turn of a

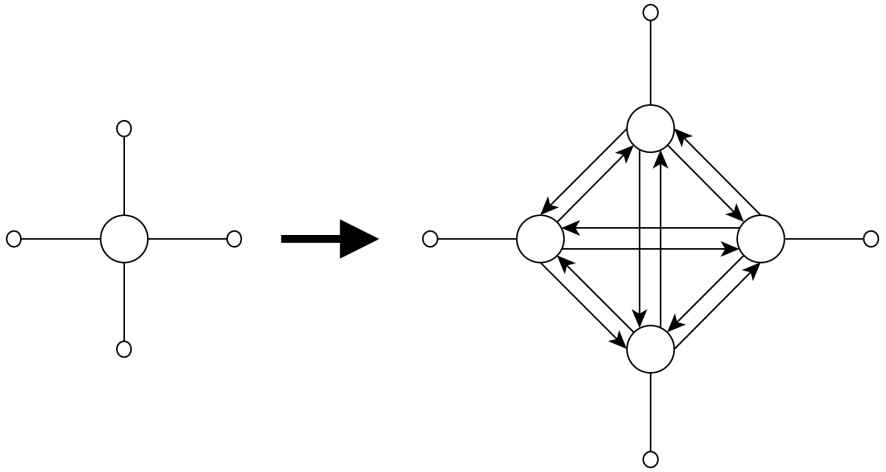

FIGURE 4.1: A junction node being expanded

junction and then immediately traveling along another turn in the same junction are prevented by saving a list of allowed turns for each entry link and for each exit link of the junction and a list of allowed entry links and a list of allowed exit links for each turn. Each turn has a cost function that depends on the type of the turn, the load on the turn itself and conflicting turns and several other user defined parameters such as number of lanes or free flow speed. OmniTRANS supports seven types of junctions: equal junctions, priority junctions with the main road going straight or turning, all stop junctions, signalized junctions and signalized and unsignalized roundabouts. Some examples of capacity and delay functions of junctions in OmniTRANS are given in Appendix A.

The objective function of the Beckmann notation, as described in section 2.3, does not exist anymore if we introduce asymmetric link costs. Therefore most traffic modeling software developers use approximations of the turn functions in order to be able to calculate a reasonable solution. Several of these approximations are described in section 4.3. The current implementation of FW in OmniTRANS uses a very simple and pragmatic approximation of the turn functions for its step size calculation: the integral is omitted and instead the turn cost function is used:

$$z(\boldsymbol{f}) = \sum_{a \in A_l} \int_0^{f_a} c_a(\omega) \mathrm{d}\omega + \sum_{a \in A_t} c_a(f_a) \tag{4.1.1}$$

Due to this crude approximation the calculated step size may actually result in a worse solution. Due to this effect the FW sometimes has problems converging to an equilibrium, as shown by Muijlwijk [19]. She also showed that VA gives better results for assignments with junction modeling. This is due to the fact

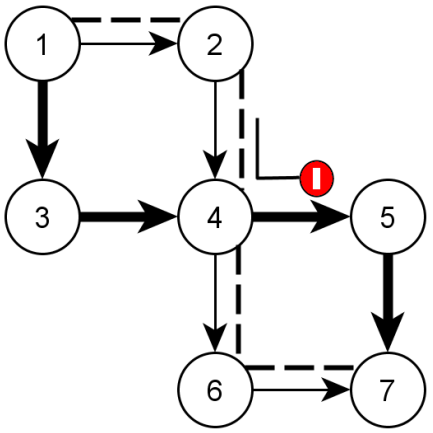

FIGURE 4.2: Example network with a banned turn

that VA uses predetermined step sizes and as such is not affected by errors in the approximations of turn cost function.

4.2 Adjustments to TAPAS

In this section we cover all adjustments needed for TAPAS to be implemented in OmniTRANS. First some examples are given to show the problems that can arise when the original methods of TAPAS are used. Then a solution is proposed.

One of the problems with using TAPAS on a network that uses Junction Modeling is that paths generated by PASs as described by Bar-Gera [3] may contain illegal turn movements. We illustrate this problem using the example in figure 4.2. Here turn $[2, 4, 5]$ is banned. In this network the demand from node 1 to node 7 is 3 units of flow. Link cost functions are as follows: $c_a = 2 + f_a$ for $a = (1, 2)$ and $a = (2, 4)$, $c_a = 3 + f_a$ for $a = (4, 6)$ and $a = (6, 7)$, and $c_a = 1 + 2f_a$ for all other links. The UE solution has 1.5 units of flow on each link. Figure 4.2 shows the situation just before the first iteration of TAPAS. The initializing AON assignment is shown as bold arrows, the shortest path from node 1 to 7 after the initialization is shown by the dashed line. There are two critical links, $(3, 4)$ and $(5, 7)$, leading to two obvious PASs, $P_1 = \{[1, 2, 4], [1, 3, 4]\}$ and $P_2 = \{[4, 5, 7], [4, 6, 7]\}$. Both PASs are allowed by the definitions stated in section 3.1, but when we perform a flow shift on PAS P_1 we may introduce a new path $[1, 2, 4, 5, 7]$ that uses the banned turn. We can prevent this by requiring that the flow on section $[1, 2, 4]$ does not exceed the flow on section $[4, 6, 7]$. But this introduced a new problem when we want to shift flow. Segment $[4, 6, 7]$ has no flow, so shifting flow on PAS P_1 would create an invalid path and we treat P_2 first. Shifting $1\frac{2}{3}$ units of flow

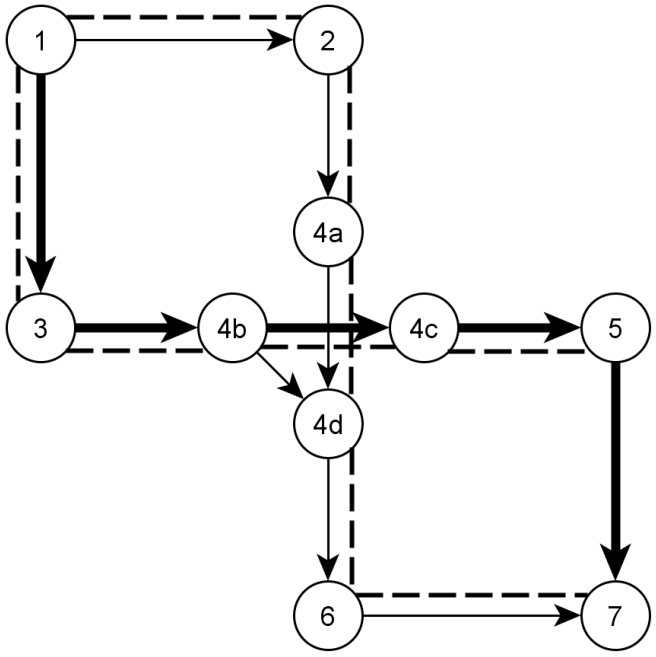

FIGURE 4.3: Example network of fig 4.2 with expanded node

on PAS P_2 equilibriates both segments of the PAS. Now we can perform the shift on P_1. The maximum allowed flow shift is $1\frac{1}{3}$, which still leaves segment $[1, 2, 4]$ the short segment. If we shift this flow TAPAS reaches a deadlock, because P_2 is equilibriated and flow shifts on P_1 are limited by the flow on segment $[4, 6, 7]$. We can solve this issue by requiring that nodes that have a banned turn are expanded, so they contain all allowed turns (which have zero cost). This can be seen in figure 4.3. Introducing turns on node 4 has ensured that the flow and the shortest path cross but don't meet at node 4. Now there is only one critical link $(5, 7)$, which leads to the PAS $\{[1, 3, 4b, 4c, 5, 7], [1, 2, 4a, 4d, 6, 7]\}$ that can reach equilibrium in one flow shift. With this approach we can circumvent all problems described above and also any potential problems that could arise in a branch shift.

Another problem that arises in the PAS construction procedure described in section 3.1 is finding the incorrect merge node or diverge node. We use the example in figure 4.4 to show what happens. Figure 4.4(a) contains a simple network with flow from node 1 to node 8 on the bolded lines and the shortest path tree indicated with the dashed lines. In this network there are three critical links: $(2, 4)$, $(4, 5)$ and $(7, 8)$. Note that link $(4, 5)$ is a critical link because the shortest path tree is on the link in the opposite direction $(5, 4)$. The PAS generation procedure identifies three PASs: $\{[1, 2, 4], [1, 3, 5, 4]\}$, $\{[1, 2, 4, 5], [1, 3, 5]\}$ and $\{[4, 5, 7, 8], [4, 6, 8]\}$. When nodes 4 and 5 are blown up, as shown in figure 4.4(b), the PAS construction fails. Here link $(4a, 5a)$ is one of the critical links, and the corresponding PAS is $\{[1, 2, 4b, 4a, 5], [1, 3, 5b, 5a]\}$. Shifting flow along this PAS creates the path

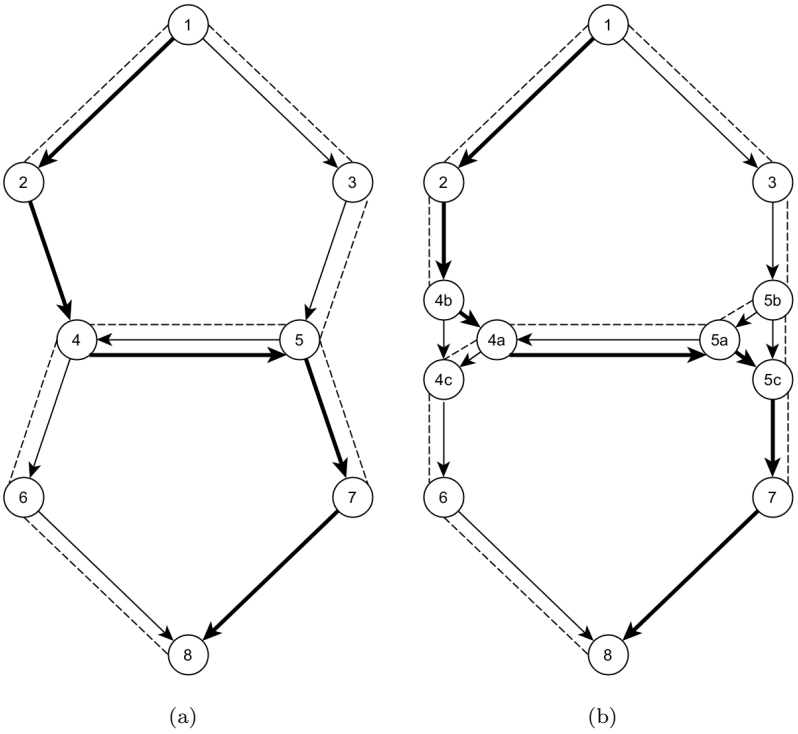

FIGURE 4.4: Example network that identifies the merge node incorrectly

$[1, 3, 5b, 5a, 5c, 7, 8]$ which includes an illegal turn movement. Similar behavior can occur with selecting a diverge node when the first link of one segment is a link and the first link of the other segment is a turn. Part of this problem is already solved by the way the shortest route generation is implemented in OmniTRANS. OmniTRANS takes a link-based approach to saving shortest paths: for each link a the cost of the shortest path from the origin to the end node a_h via link a is stored along with the predecessor of link a with this shortest path. So the shortest path tree in OmniTRANS is not really a tree anymore, because shortest path information is stored for all links, so nodes are reachable through multiple paths. We take advantage of this link-based approach by changing the backtracking phase of the PAS generation procedure as follows: when link a is visited, instead of checking whether node a_t is part of the shortest path from the origin to the critical link, we check whether one of the predecessor links of link a is part of the shortest path. This ensures that both segments of the generated PAS are part of an allowed path. In the example of figure 4.4 we can see this procedure working. Link $(5b, 5a)$ can not be the shortest path alternative of link $(4a, 5a)$, because none of the successor links of either link have both $(5b, 5a)$ and $(4a, 5a)$ as predecessor links. In fact, link $(4a, 5a)$ will no longer be a critical link, because the only shortest path alternative of link $(4a, 5a)$ is itself, so it will always be part of the shortest path from the origin to the end node $5a$.

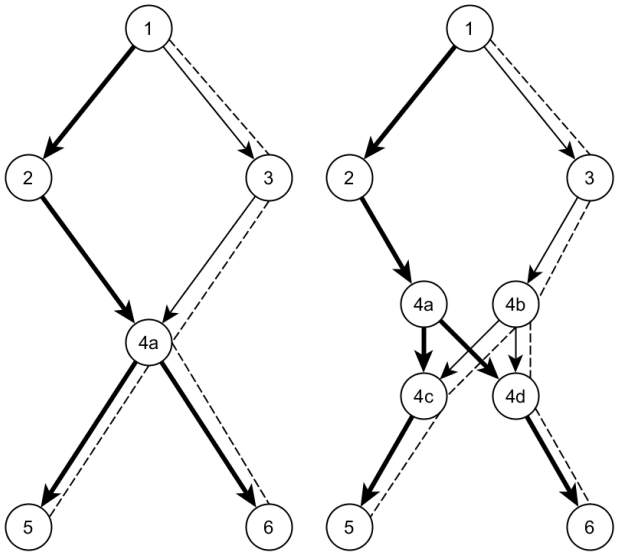

FIGURE 4.5: A network that shows the lost PAS efficiency from introducing junctions

A disadvantage of TAPAS compared to FW is that TAPAS has higher memory usage due to the OB flows that are stored: FW uses $|A|$ entries to store its link flows and TAPAS needs $|A|\cdot|O|$. Blowing up junctions increases this problem. A possible solution for this is using a sparse matrix representation for the link flows. Empirical evidence has shown that up to 80 percent of OB flows are zero, so a sparse matrix could be a great improvement in memory usage. Blowing up junctions also gives a problem in the efficiency of PAS construction. In figure 4.5(a) we see a network without junctions where flow goes from origin 1 to destinations 5 and 6. As always, the flow is on bolded lines and the shortest path tree is represented by the dashed lines. Here one PAS $\{[1,2,4],[1,3,4]\}$ is needed and flow is shifted from paths $[1,2,4,5]$ and $[1,2,4,6]$ to paths $[1,3,4,5]$ and $[1,3,4,6]$ simultaneously. If node 4 is blown up, as can be seen in figure 4.5(b), this advantage is lost. Now two PASs are needed: $\{[1,2,4a,4c],[1,3,4b,4c]\}$ and $\{[1,2,4a,4d],[1,3,4b,4d]\}$. While this behavior does not compromise the ability of TAPAS to converge, more PASs are needed and so more memory is needed and computation time goes up.

4.3 Solutions for Asymmetric Cost Functions

As described in section 2.3, adding turn delays to a traffic assignment problem makes it impossible to solve using the traditional Beckmann formulation, because the cost functions are asymmetric and the integral objective function does not

exist anymore. Subsections 4.3.1 and 4.3.2 cover two heuristics that tackle this problem. Also multiple User Equilibria may exist, which makes comparing solutions obtained by making small changes to the network or OD matrix or solutions generated by different algorithms very difficult. In section 4.3.3 a method is described that tries to 'push' the solution to the same equilibrium regardless of the starting state of the network.

4.3.1 Diagonalization

The diagonalization algorithm for asymmetric cost TAP was introduced by Florian and Spiess [18]. The idea behind the algorithm is simple: in the cost function for each link all variables of conflicting junctions are fixed, so the link function only depends on the flow on the link itself. This way the link cost functions become separable and we can use the Beckmann program to find an optimal step size given the fixed costs of conflicting turns. To this end a new cost function is defined for links with asymmetric costs:

$$\tilde{c}_a^i(f_a) = c_a(f_1^{i-1}, f_2^{i-1}, \dots, f_{a-1}^{i-1}, f_a, f_{a+1}^{i-1}, \dots) \tag{4.3.1}$$

Here i is the iteration number and f_b^{i-1} is the flow on link b at the end of the iteration $i-1$. In each iteration the UE is approximated by the problem with the diagonalized turn costs $\tilde{c}_a^i(f_a)$. Then the cost functions are updated, leading to improved approximations of the cost functions. This is repeated until $f_a^i = f_a^{i-1}$ for all $a \in A_t$. Algorithm 3 shows an overview of the diagonalization algorithm. An alternative approach, called the streamlined diagonalization algorithm was given by Sheffi [25]. Here only one inner iteration of the used assignment algorithm is performed between consecutive updates of diagonalized cost functions. This method saves a lot of calculation time in early iterations, because the difference between f_a^{i-1} and f_a^i is still large, so a lot of time is wasted calculating a converged solution for the diagonalized turn costs $\tilde{c}_a^i(f_a)$.

A modified version of the streamlined diagonalization algorithm can be used efficiently in TAPAS, due to the structure of flow shifts in PASs. Before a flow shift we determine the diagonalized cost function $\tilde{c}_a(f_a)$ for each turn that is part of one of the segments of the PAS. There are never more than two turns of the same junction that are part of a PAS. When a junction is the merge node or the diverge node of the PAS, there is one turn a_1 that is part of segment 1 and one turn a_2 that is part of segment 2, so there are at most two junctions that have two of its

Algorithm 3: Diagonalization algorithm

Find an initial solution f_a^0 using an AON assignment
Set $i = 1$
while *not converged* **do**
  Set turn costs $\tilde{c}_a^i(f_a)$ using equation 4.3.1
  Use solution f_a^{i-1} of iteration $i-1$ as initial solution for iteration i
  Inner iterations
  while *UE given turn costs* $\tilde{c}_a^i(f_a)$ *not found* **do**
    Perform an iteration of assignment algorithm of your choice
  end
  Store link flows f_a^i, update link costs $c_a(\boldsymbol{f})$
  Set $i = i + 1$
end

turns in the segments of the PAS. For these turns the diagonalized cost function $\tilde{c}_{a_1}^i(f_{a_1})$ differs only from $c_{a_1}(\boldsymbol{f})$ in the term for a_2 if flow is shifted from segment 1 to segment 2. For all other junctions that are part of the PAS, there is exactly one turn a that is part of one of the segments of the PAS, so for turn a we have that $\tilde{c}_a^i(f_a) = c_a(\boldsymbol{f})$. Therefore the diagonalized turn cost functions will be a reasonable approximation of the real turn cost functions. We split up the terms in equation 3.2.2 so that for each segments the turns and links are grouped. Before each flow shift all turn cost functions are diagonalized using equation 4.3.1. This results in the function that we use to determine the optimal step size:

$$\begin{aligned}\min_{0\le\lambda\le 1} &\sum_{a\in s_1|a\in A_l}\int_0^{f_a-\lambda f_{max}} c_a(\omega)\mathrm{d}\omega + \sum_{a\in s_1|a\in A_t}\int_0^{f_a-\lambda f_{max}} \tilde{c}_a(\omega)\mathrm{d}\omega \\ &+ \sum_{a\in s_2|a\in A_l}\int_0^{f_a+\lambda f_{max}} c_a(\omega)\mathrm{d}\omega + \sum_{a\in s_1|a\in A_t}\int_0^{f_a+\lambda f_{max}} \tilde{c}_a(\omega)\mathrm{d}\omega\end{aligned} \tag{4.3.2}$$

After the flow shift the link flows and costs of all links in the PAS are updated. This way each flow shift uses a turn cost function that is up to date.

4.3.2 VISUM Solution

An alternative method is implemented in the VISUM traffic modeling software by PTV [26]. It shares similarities with the diagonalization algorithm in that flows on conflicting turns are fixed in the turn cost functions during an iteration. For each turn the cost function is replaced by a piecewise linear function. The flows on conflicting turns are fixed and three values are calculated by varying the flow on the turn itself: it uses the current flow, one flow that is lower and one

that is higher. These three values yield a simple and easy to use piecewise linear function that depends only on the flow of the turn itself. VISUM then performs a few iterations of its assignment algorithm before updating the turn functions. Calculations using these approximation functions are fast, but the approximations may not be very good, since only three points are used for the piecewise linear functions. Therefore this method is not implemented in TAPAS.

4.3.3 Finding Consistent Solutions

One method for tackling the problem of multiple equilibria is already implemented in OmniTRANS. Ideally we would like to find the same equilibrium every time we do a traffic assignment, regardless of the initial state of the network. This is not the case with asymmetric cost functions, as shown by Muijlwijk [19]. This is particularly problematic for variant studies in which consultants compare solutions of two networks that differ slightly. We want to be sure that all differences between the solutions in the variant study are the result of the changes in the network and not of different equilibria calculated for the variants. So we try to push the assignment algorithm to the same equilibrium by gradually introducing the turn delays throughout the iterations of the algorithm. The idea is that during the first iterations the turn delays are omitted and the algorithm calculates a solution that is close to the user equilibrium without junction modeling. Then the turn delays are introduced gradually, so that a UE with junction modeling that is 'close' to the UE without junction modeling is found. To achieve this, we introduce a vector $\boldsymbol{\mu} = [\mu_1, \mu_2, \ldots, \mu_n]$, where n is the maximum number of iterations and $0 \leq \mu_i \leq 1$, $\mu_i \leq \mu_{i+1}$ for $1 \leq i \leq n-1$ and $\mu_n = 1$. We then change the objective function in the i-th iteration to

$$z_i(f) = \sum_{a \in A_l} \int_0^{f_a} c_a(\omega) \, \mathrm{d}\, \omega + \mu_i \sum_{a \in A_t} \int_0^{f_a} c'_a(\omega) \, \mathrm{d}\, \omega \tag{4.3.3}$$

Here c'_a is one of the approximation functions described in previous sections, since the integral of the actual turn delay function does not exist. The values in $\boldsymbol{\mu}$ are project and user specific and can be chosen by the user based on his or her preferences.

Chapter 5

Results

This chapter contains some results generated by the prototype of TAPAS made in OmniTRANS. All solutions calculated by TAPAS are compared with solutions calculated by FW and VA, the algorithms currently available in OmniTRANS. Section 5.1 contains a description of the prototype, with explanations of some of the design choices made. Section 5.2 shows some results of TAPAS, FW and VA without junction delays. In section 5.3 results of the three algorithms with junction delays incorporated are treated.

5.1 Prototype

A prototype was made in the OmniTRANS software package. Due to time constraints only the essential parts of TAPAS described in chapter 3 were implemented and both the branch shift and proportionality procedures were skipped. This does not impair the ability of the prototype to converge. Proportionality is an auxiliary method that produces more realistic path flow solutions but is not needed for convergence. Branch shifts are needed for guaranteed speedy convergence, but can be replaced by using the PAS that least violates the flow effectiveness condition. As for junction modeling, i.e. for dealing with asymmetric costs the methods described in section 4.3 were not implemented. Instead a simpler bisection method is implemented that iteratively approaches the step size for which the difference in cost between both segments of the PAS is minimal. The method is outlined in algorithm 4. As always we determine the maximal allowed flow shift f_{max} and try to determine the fraction λ of f_{max} that is optimal. For this procedure we define $\delta(c_\lambda)$ as the absolute cost difference between segment s_1 and s_2 if we would shift the fraction λ of the maximal allowed flow shift f_{max}. In the iteration scheme we

store λ_{low} and λ_{high}, that are initialized at 0 and 1 respectively, as the current lower and upper bound of the step size we want to perform. These bounds are then tightened by calculating $\delta(c_\lambda)$ for two points halfway between λ_{low} and λ_{high} and then choosing new upper and lower bounds based on the cost differences.

While this method is less precise than a direct calculation using for instance the diagonalization method, it does fit well in the iteration scheme of TAPAS where flow is shifted on a PAS multiple times in order to reach equilibrium. The main drawback of this method is that segment costs need to be calculated many times in each PAS shift, so calculation time of the algorithm goes up dramatically compared to TAPAS without junctions and therefore this method can only be used for demonstration purposes.

Algorithm 4: Procedure for finding the step size of flow shifts on a PAS with junctions

determine maximal allowed flow shift f_{max}
set $i = 0$
set $\lambda_{low} = 0$
set $\lambda_{high} = 1$
set $\lambda_1 = 1/3$
set $\lambda_2 = 2/3$
while $i < 20$ **do**
  calculate $\delta(c_{\lambda_{low}})$, $\delta(c_{\lambda_{high}})$, $\delta(c_{\lambda_1})$ and $\delta(c_{\lambda_2})$
  determine $\delta_{min} = \min\{\delta(c_{\lambda_{low}}), \delta(c_{\lambda_{high}}), \delta(c_{\lambda_1}), \delta(c_{\lambda_2})\}$
  if $\delta_{min} = \delta(c_{\lambda_{low}})$ **then**
    $\lambda_{high} = \lambda_1$
  end
  else if $\delta_{min} = \delta(c_{\lambda_1})$ **then**
    $\lambda_{high} = \lambda_2$
  end
  else if $\delta_{min} = \delta(c_{\lambda_2})$ **then**
    $\lambda_{low} = \lambda_1$
  end
  else
    $\lambda_{low} = \lambda_2$
  end
  $\lambda_1 = \lambda_{low} + 1/3(\lambda_{high} - \lambda_{low})$
  $\lambda_2 = \lambda_{low} + 2/3(\lambda_{high} - \lambda_{low})$
end
$\lambda = \arg\min\{\delta(c_{\lambda_{low}}), \delta(c_{\lambda_{high}}), \delta(c_{\lambda_1}), \delta(c_{\lambda_2})\}$

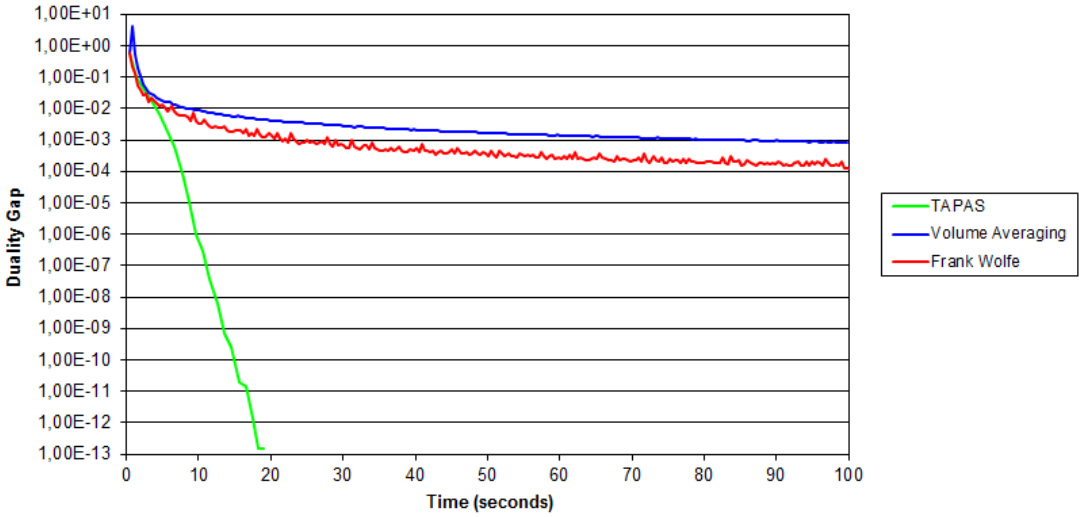

FIGURE 5.1: Convergence of algorithms in OmniTRANS without junction delays on the Amsterdam network

5.2 TAPAS without Junction Modeling

Figure 5.1 shows the performance of TAPAS, VA and FW on the medium sized network of Amsterdam. Amsterdam is a network consisting of 280 zones and 4000 links. It contains no junctions. This figure the duality gap achieved by each algorithm throughout the running time. It can be seen clearly that initially the performance of all three algorithms is similar, but while the convergence rate of FW and VA slows down after a few seconds, TAPAS keeps on converging at a fast rate. It stops after 19.04 seconds when it has reached a duality gap of $< 1.0 \cdot 10^{-12}$ after 19 iterations. In the same time that TAPAS needed to converge to a duality gap of 10^{-12}, VA and FW performed 50 iterations each and reached a duality gap of $4.5 \cdot 10^{-3}$ and $1.2 \cdot 10^{-3}$ respectively. So, while iterations take more time in TAPAS than in VA and FW, each iteration is much more efficient. This result is comparable to results shown by Bar-Gera [3] and Inoue and Maruyama [12].

5.3 TAPAS with Junction Modeling

Figure 5.2 shows the performance of TAPAS, VA and FW on a small rectangular test network consisting of 4 zones, 60 links and 32 nodes, of which 11 contain junctions. In this figure we can clearly see that FW performs much worse than VA due to the bad approximation of the objective function of FW programmed

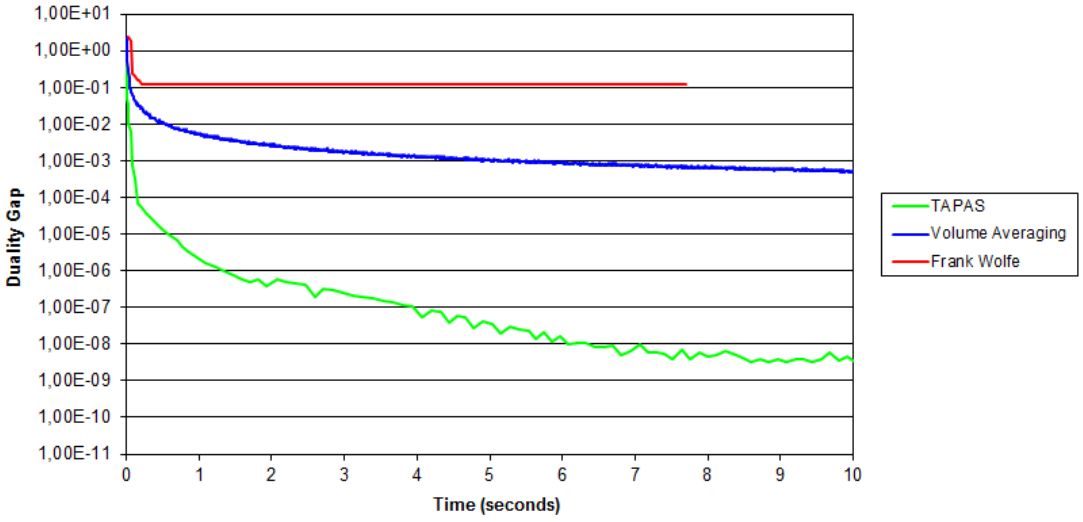

FIGURE 5.2: Convergence of algorithms in OmniTRANS with junction delays on a small test network

in OmniTRANS. FW reaches a duality gap of $1.2 \cdot 10^{-1}$ after 8 iterations and fails to improve upon that in later iterations. Also TAPAS performs less well when compared to the case without junction modeling. After a few iterations convergence slows down and TAPAS fails to reach the duality gap $1.0 \cdot 10^{-12}$ that is stated as its stop condition but instead stops after 100 iterations at a duality gap of $3.2 \cdot 10^{-9}$. Also it can be clearly seen that the procedure described in section 5.1 makes flow shifts much slower in TAPAS. In the case without junction modeling TAPAS iterations where approximately 2.5 times slower than VA and FW (19 iterations for TAPAS compared to 50 for FW and VA), with junction modeling TAPAS needs 10.8 seconds to perform 100 iterations, in the same amount of time VA can perform 1300 iterations, making iterations of TAPAS 13 times slower than VA iterations. However, the duality gap achieved by TAPAS is a great improvement over VA, that reaches a duality gap of $3.2 \cdot 10^{-4}$ after 2000 iterations. This result of TAPAS is very promising when considering that this could be improved both in precision and in calculation time when exact methods are implemented.

Figure 5.3 shows the performance of TAPAS, VA and FW on the Delft network which consists of 25 zones, 1172 links and 472 nodes, of which foughly 100 contain junctions. For this network TAPAS shows the same behavior as FW: in 8 iterations the final duality gap $8.6 \cdot 10^{-7}$ is reached and the algorithm is stuck in a deadlock. It is not sure if this is due to the way turn functions are defined or to the inaccuracy of the approximation procedure described in section 5.1, so more research is needed

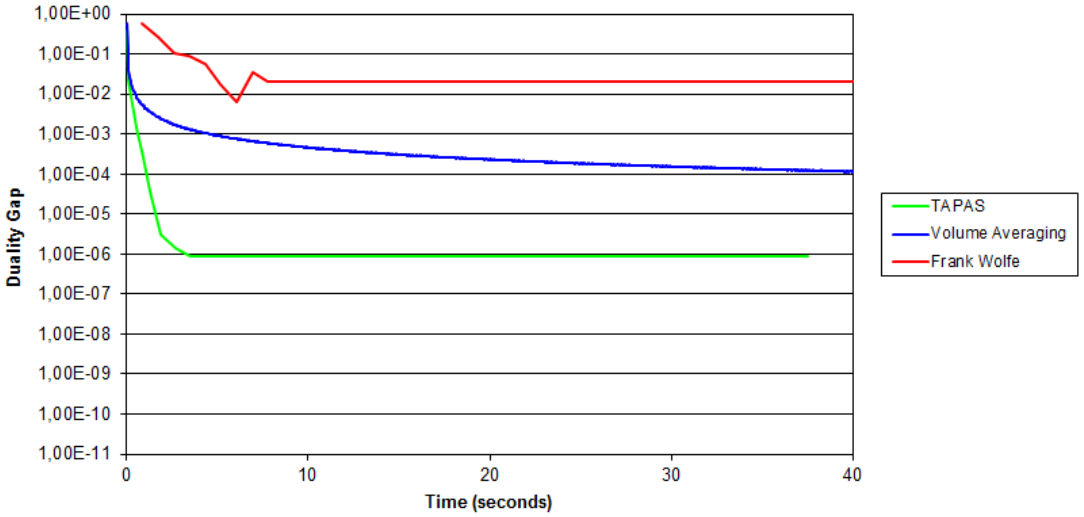

FIGURE 5.3: Convergence of algorithms in OmniTRANS with junction delays on the Delft network

on this topic. The duality gap calculated by TAPAS is still a lot better than VA, that reaches a duality gap of $1.3 \cdot 10^{-4}$ after 1000 iterations, and FW, that reaches a duality gap of $2.0 \cdot 10^{-2}$.

Chapter 6

Discussion

This thesis gives a complete and thorough description of TAPAS, along with proposed changes to incorporate Junction Modeling and results generated with a prototype. There is still room for improvement though. Section 6.1 describes some features of the traffic assignment problem that are only partially treated in this thesis or not at all, but could improve the assignments calculated by TAPAS. Section 6.2 treats the shortcomings of the prototype and the solutions offered in this thesis. Finally, section 6.3 gives recommendations for further research.

6.1 Elements of TAP

One of the elements of TAPAS described in this thesis, but not implemented in the prototype is proportionality. This does not affect the convergence rate of TAPAS, but this procedure adds paths to the path flow solution, which is a big advantage to consultants who make great use of path flow solutions in their analyses. Currently it is not clear how many paths with flow would be added to the path flow solution. Extra research is needed to find out exactly how many paths the proportionality procedure adds and how much calculation time is added by this procedure.

OmniTRANS is able to calculate multimodal traffic assignments. It is possible to define multiple modes of transport, each with their own OD matrix and network parameters, and also different purposes for each mode. This is used to model differences in travel incentive and route choice of different travelers: a commuter may want to choose the quickest way home using the main highways in terms of travel time, whereas recreational traffic may choose to cross the countryside, and also some roads are closed for certain kinds of transport. No implementation of

this is currently available in TAPAS and this topic falls out of the scope of this thesis, but this could be a topic of further research as it adds realism to the model.

Another question one can ask is what duality gap one wants to achieve. TAPAS is able to reach highly converged solutions with a duality gap less than $1.0 \cdot 10^{-12}$, but is it worth the calculation time to reach a solution that is this far converged? This question was addressed by Boyce et al. [27]. In the research of Boyce et al. a variant study was performed in which one variant used a network that represented the current road network of a major city and the other variant used a network with a road that was planned. The differences in the calculated solutions were used to make an informed decision on whether to add the planned road or not. The goal of the research was to find the duality gap for which the convergence of the solutions no longer affected the decision made and all changes in solutions could be attributed to changes in the network. They found that for duality gaps lower than $1.0 \cdot 10^{-7}$ the difference between link flows was so small that the decision was no longer affected and they concluded that solutions converged to this duality gap were sufficient for all practical analysis.

6.2 Prototype and results

As shown in chapter 5 TAPAS can calculate solutions that are converged much better than the current methods available in OmniTRANS, but there are still some issues that need to be dealt with before TAPAS is a viable alternative for OmniTRANS to use. First of all an improved method for performing flow shifts for PASs with junctions needs to be implemented. Several alternatives are mentioned in section 4.3, and diagonalization appears to be the best option based on the way the structure of diagonalization fits the flow shift procedure of PASs. An implementation of all alternatives is needed to make an informed decision on which option is the best for TAPAS. If these options are implemented it also can be checked whether the deadlock state, as seen in figure 5.3, is the result of the approximation procedure described in section 5.1 or of the way the turn functions are defined. Also the branch shift and proportionality procedures need to be implemented in TAPAS to see how these procedures affect the performance and calculated solutions of TAPAS. Another point of ongoing research is the application of sparse matrices for storing link flows and the effects this implementation has on memory usage and performance of TAPAS. OmniTRANS is used for calculating assignments for networks of up to 5000 zones and 400000 links, so the origin based link flows of such a network take up 14.9 Gb of memory space, an

amount that is not available on most computers. Empirical evidence on medium sized networks shows that approximately 80% of OB flows are zero, so a sparse matrix implementation could make it possible to use TAPAS for networks of such sizes. This does of course come at the cost of reduced performance, since reading and writing data in a sparse matrix takes more computation time.

6.3 Recommendations

An interesting branch of research is to see whether the list of PASs that was used to calculate an equilibrium solution can be used as an analysis tool for consultants. One example of this is that the PAS list may be used as a memory efficient way to represent path flows. Path flows are one of the main tools for consultants to perform analysis with, but FW and VA use aggregated link flows, so path flows need to be stored explicitly in order to be able to use them for analysis. Since in theory a covering PAS set can be used to calculate all used paths in a post processing step, this would result in a more memory efficient method. Some research is needed on the PAS removal procedure to ensure that TAPAS does not remove PASs that are part of the covering set of PASs. A second example is when one wants to compare the differences between two solutions of traffic assignment problems that have a slightly different network or OD matrix. FW has to calculate both solutions separately and cannot use information obtained in the calculation of the first assignment to speed up the calculation of the second assignment. TAPAS on the other hand can store the PAS list used in the first calculation to get a warm start for the second assignment. This way TAPAS starts with a nonempty PAS list in its first iteration and, assuming that the eventual PAS lists for both assignments have a lot of overlap, a lot less time is needed for managing the PAS list. A similar approach is described by Dial [8] for algorithm B that uses the bushes created in a previous assignment to be able to calculate an assignment with a slightly changed OD matrix in as little as 25% of the time of the first calculation.

Chapter 7

Conclusion

In this thesis the structure of the traffic assignment algorithm TAPAS was studied and detailed descriptions of each part of the algorithm were given. For all parts where the original paper gave an incomplete description or left room for interpretation, a solution was proposed. Also a complete proof of convergence was given. It was shown that several changes need to be made to TAPAS in order to be able to deal with junction modeling in OmniTRANS. A prototype of TAPAS was made in OmniTRANS that was used to compare the performance of TAPAS to the two algorithms currently implemented in OmniTRANS, Frank Wolfe and Volume Averaging. The prototype replicated the results of earlier publications on networks without junction modeling and showed that TAPAS is able to calculate highly converged solutions with a duality gap of up to $1.0 \cdot 10^{-12}$. Using an approximation procedure that could perform flow shifts for PASs that contain junctions, it was shown that TAPAS is able to calculate solutions that are converged to a much higher level than VA and FW can. However the approximation method is slow and not stable enough for practical applications and needs to be replaced with a more accurate method described in 4.3.

Appendix A

Junction functions

There are seven different types of junctions modeled in OmniTRANS: equal junctions, priority junctions with the main road going straight, priority junctions with the main road turning, all stop junctions, signalized junctions, unsignalized roundabouts and signalized roundabouts. This appendix shows the structure of junction functions in OmniTRANS by giving a simplified version of capacity and delay functions for equal junctions, priority junctions and unsignalized roundabouts, that keep the structure of the formulas but omit some of the parameters for readability. These three junctions share most of the terms of their functions and differ only in the calculation of the geometric delay. For a complete overview of the calculation of all capacity and delay functions for all junction types we refer to the guide of junction functions in OmniTRANS[28]. For each junction all turns are numbered, as shown in figure A.1 for an equal junction with four arms. For each junction a conflict matrix is created. When a turn t conflicts with another turn t', the delay function of turn t will contain a term that represents the delay created by flow on turn t' for turn t. Turn conflicts need not be symmetrical: traffic on a minor road of a priority junction does experience delay from traffic on the main road, but this is not true the other way around.

The effect that traffic on conflicting turns has on the delay of turn t is modeled by reducing the capacity q_t of turn t. The capacity of each turn is calculated using

$$q_t = \max(\sigma_t - \sum_{b \in Y_t} f_b, q_{min,t})$$

with

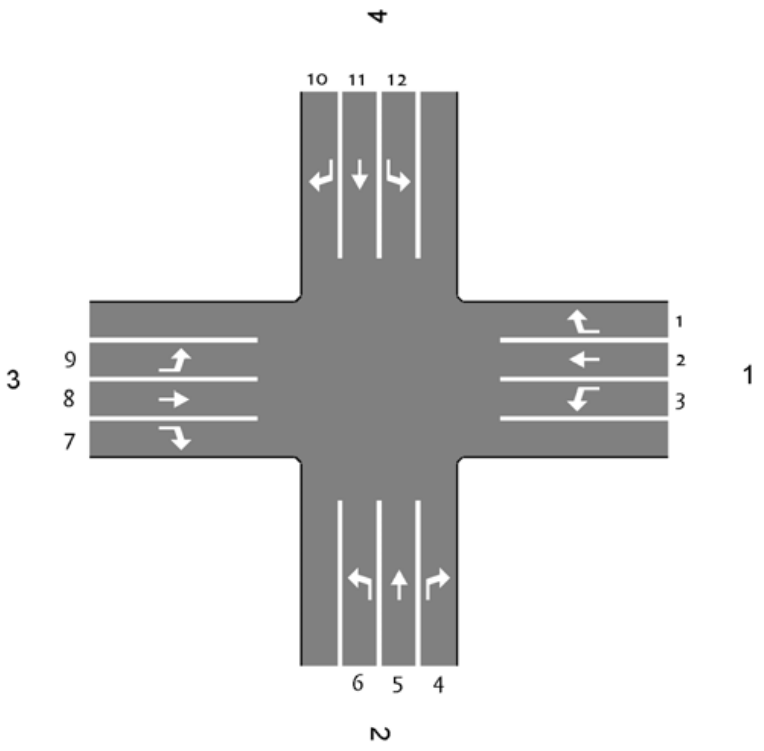

FIGURE A.1: An equal junction with its turns numbered

q_t	capacity of turn t
σ_t	saturation flow of turn t
Y_t	set of conflicting turns for turn t
f_b	flow on turn b
$q_{min,t}$	minimal capacity of turn t

Here the saturation flow is the capacity the turn can handle if there are no conflicting turn movements. The minimal capacity is used to prevent complete total congestion. The capacity is then used to calculate the delay for each turn

$$c_t = \min(c_{1,t} + c_{2,t} + c_{3,t}, c_{max,t})$$

with

c_t	delay of turn t
$c_{1,t}$	uniform delay of turn t
$c_{2,t}$	incremental delay of turn t
$c_{3,t}$	geometric delay on turn t
$c_{max,t}$	maximal delay on turn t

The uniform delay is calculated using

$$c_{1,t} = \frac{1}{q_t}$$

The incremental delay is calculated using

$$c_{2,t} = \begin{cases} \left(\frac{f_t}{q_t} - 1\right) + \sqrt{\left(\frac{f_t}{q_t} - 1\right)^2 + \frac{f_t}{q_t^2}} & \text{if } \frac{f_t}{q_t} \geq \alpha \\ 0 & \text{if } \frac{f_t}{q_t} \leq \alpha \end{cases}$$

with

f_t	flow on turn t
q_t	capacity of turn t
α	parameter usually set to 0.5

The geometric delay is calculated differently for each unsignalized junction. For equal junctions the geometric delay is

$$c_{3,t} = \begin{cases} 1 & \text{if } f_t > 0 \\ 0 & \text{if } f_t = 0 \end{cases}$$

For priority junctions the geometric delay is

$$c_{3,t} = \begin{cases} 1 & \text{if } f_t > 0 \text{ and turn is on major road} \\ \frac{f_{adj}}{f_t} & \text{if } f_t > 0 \text{ and turn is on minor road} \\ 0 & \text{if } f_t = 0 \end{cases}$$

For unsignalized roundabouts the geometric delay is

$$c_{3,t} = \beta$$

with

$$f_{adj} = \beta f_{left} + f_{through} + \beta f_{right}$$

and

f_{left}	flow on left turn movement
$f_{through}$	flow on straight turn movement
f_{right}	flow on right turn movement
f_t	flow on turn t
β	parameter usually set to 7

Appendix B

The Cycle Removal Algorithm

This appendix contains a description in pseudocode for the cycle removal algorithm described in 3.5.

```
initialize linkStack
cycleCounter = 0
removeCycles()
{
  for (each origin p)
  {
    for (each link a)
    {
      a.lowestIndexOfCycle = -1
      a.inStack = false
    }
    for (each link a)
    {
      if (f(pa) > 0 and a.cycleIndex == -1)
      {
        findCycleForLink(a, p)
      }
    }
  }
}
```

```
findCycleForLink(a, p)
{
  a.cycleIndex = cycleCounter
  a.lowestIndexOfCycle = cycleCounter
  a.inStack = true
  cycleCounter++
  linkStack.add(a)

  for (each successor a' of a)
  {
    findCycleForLinkInternal(a, a', p)
  }
  if (a.lowestIndexOfCycle == a.cycleIndex)
  {
    removeLinkFromStack(a, p)
  }
}

findCycleForLinkInternal(a, a', p)
{
  if (f(a',p) > 0)
  {
    if (a'.inStack == true)
    {
      a.lowestIndexOfCycle = a'.cycleIndex
    }
    else if (a'.cycleIndex == -1)
    {
      findCycleForLink(a', p)
      if (a'.lowestIndexOfCycle >= 0)
      {
        a.lowestIndexOfCycle = min(a.lowestIndexOfCycle,
                                   a'.lowestIndexOfCycle)
      }
    }
  }
}
```

```
removeLinkFromStack(a, p)
{
  initialize cycleList
  while (linkStack.back != a)
  {
    a' = linkStack.back
    cycle.add(a')
    linkStack.remove(a')
    a'.inStack = false
  }
  if (linksInCycle.size > 1)
  {
    minFlow = min {f(p,a')}
    for (each a in cycle)
    {
      f(p,a') -= minFlow
      a'.cycleIndex = -1
      a'.lowestIndexOfCycle = -1
    }
    if (f(p,a) > 0)
    {
      findCycleForLink}(a, p)
    }
  }
}
```

Abbreviations

TAP	Traffic Assignment Problem
TAPAS	Traffic Assignment by Paired Alternative Segments
PAS	Pair of Alternative Segments
OD pair	Origin Destination pair
OB flow	Origin Based flow
SPT	Shortest Path Tree
AON	All Or Nothing assignment
VA	Volume Averaging
FW	Frank Wolfe
OBA	Origin Based Algorithm
Alg B	Algorithm B
LUCE	Linear User Cost Algorithm
PG	Projected Gradient Algorithm

Symbols

Symbols

N	Set of nodes
A	Set of links
n	Node
a	Link
p	Origin
q	Destination
N_o	Set of origins
$N_d(p)$	Set of destinations for origin p
r	Route
s	Route segment
R_{pq}	Set of routes connecting origin p to destination q
$\boldsymbol{R}$	Set of all routes connecting any origin to any destinations
IN_n	Set of links that end at node n
OUT_n	Set of links that start at node n
a_t	Tail (begin node) of link a
a_h	Head (end node) of link a
d_{pq}	Demand for OD pair pq
f_a	Total flow on link a
f_{pa}	Flow on link a coming from origin p
$\boldsymbol{f}$	Vector of OB flows f_{pa}
	or vector of link flows f_a
h_r	Flow on route r

$\boldsymbol{h}$	Vector of route flows h_r
g_{pn}	Flow from origin p entering node n
α_{pa}	Proportion of flow that enters a_h through link a
k_{ps}	Flow on segment s from origin p
$c_a(\boldsymbol{f})$	Link cost for link a
π_{nm}	Cost of shortest path between nodes n and m
$\mathcal{P}$	Set of PASs
O_P	Set of relevant origins for PAS P
A_t	Set of turns
A_l	Set of regular links

Bibliography

[1] J. Wardrop. Some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers*, Part 2:325–378, 1952.

[2] M. Beckmann, C. B. McGuire, and C. B. Winsten. Studies in the economics of transportation. Technical report, Yale University, 1956.

[3] H. Bar-Gera. Traffic assignment by paired alternative segments. *Transportation Research Part B: Methodological*, 44(89):1022 – 1046, 2010.

[4] O. van der Kooi. Tapas: a new algorithm for traffic assignment. *Not published*, 2013.

[5] J. Ortúzar and L. G. Willumsen. *Modelling Transport*. Wiley, 2001.

[6] L. J. LeBlanc, E. K. Morlok, and W. P. Pierskalla. An efficient approach to solving the road network equilibrium traffic assignment problem. *Transportation Research*, 9(5):309 – 318, 1975.

[7] H. Bar-Gera. Origin-based algorithm for the traffic assignment problem. *Transportation Science*, 36(4):398–417, 2002.

[8] R. B. Dial. A path-based user-equilibrium traffic assignment algorithm that obviates path storage and enumeration. *Transportation Research Part B: Methodological*, 40(10):917 – 936, 2006.

[9] G. Gentile. Linear user cost equilibrium: a new algorithm for traffic assignment. *submitted to Transportation Research B*, 2009.

[10] M. Florian, I. Constantin, and D. Florian. A new look at projected gradient method for equilibrium assignment. *Transportation Research Record: Journal of the Transportation Research Board*, 2090(-1):10–16, 2009.

[11] J. Rosen. The gradient projection method for nonlinear programming. part i. linear constraints. *Journal of the Society for Industrial and Applied Mathematics*, 8(1):181–217, 1960.

[12] S. Inoue and T. Maruyama. Computational experience on advanced algorithms for user equilibrium traffic assignment problem and its convergence error. *Procedia-Social and Behavioral Sciences*, 43:445–456, 2012.

[13] S. C. Dafermos. An extended traffic assignment model with applications to two-way traffic. *Transportation Science*, 5(4):366–389, 1971.

[14] S. C. Dafermos. Traffic equilibrium and variational inequalities. *Transportation science*, 14(1):42–54, 1980.

[15] M. J. Smith. An algorithm for solving asymmetric equilibrium problems with a continuous cost-flow function. *Transportation Research Part B: Methodological*, 17(5):365–371, 1983.

[16] M. J. Smith. The existence and calculation of traffic equilibria. *Transportation Research Part B: Methodological*, 17(4):291–303, 1983.

[17] S. Lawphongpanich and D. W. Hearn. Simplical decomposition of the asymmetric traffic assignment problem. *Transportation Research Part B: Methodological*, 18(2):123–133, 1984.

[18] M. Florian and H. Spiess. The convergence of diagonalization algorithms for asymmetric network equilibrium problems. *Transportation Research Part B: Methodological*, 16(6):477–483, 1982.

[19] H. Muijlwijk. Static traffic assignment with junction modelling. *not published*, 2012.

[20] M. Klein. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Management Science*, 14(3):205–220, 1967.

[21] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research*, 15(3):430–466, 1990.

[22] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[23] E. Dijkstra. *A discipline of programming*, volume 1. Prentice-Hall, Inc., 1976.

[24] T. Larsson, J. T. Lundgren, C. Rydergren, and M. Patriksson. Most likely traffic equilibrium route flows analysis and computation. In *Equilibrium Problems: Nonsmooth Optimization and Variational Inequality Models*, pages 129–159. Springer, 2004.

[25] Y. Sheffi. *Urban transportation networks: equilibrium analysis with mathematical programming methods*. Prentice-Hall, Inc., 1985.

[26] PTV. *VISUM 12.5 Fundamentals*. epubli GmbH, 2012.

[27] D. Boyce, B. Ralevic-Dekic, and H. Bar-Gera. Convergence of traffic assignments: how much is enough? *Journal of Transportation Engineering*, 130(1): 49–55, 2004.

[28] F. Brandt and M. Schilpzand. Explanation of junction modeling. *not published*, 2007.