WORKING PAPER

ITLS-WP-18-21

General solution scheme for the Static Link Transmission Model

By
Mark P.H. Raadsen[1] and Michiel C.J. Bliemer[1]

[1]Institute of Transport and Logistics Studies (ITLS), The University of Sydney Business School, Sydney

November 2018

ISSN 1832-570X

INSTITUTE of TRANSPORT and LOGISTICS STUDIES

The Australian Key Centre in Transport and Logistics Management

The University of Sydney

Established under the Australian Research Council's Key Centre Program.

NUMBER: Working Paper ITLS-WP-18-21

TITLE: **General solution scheme for the Static Link Transmission Model**

ABSTRACT: Until the present day most static traffic assignment models are neither capacity constrained nor storage constrained. Recent studies have resulted in novel approaches that consider capacity constraints and sometimes storage constraints. We build upon the results of these works and the model formulated in our companion paper Bliemer and Raadsen (2018a) which comprises a static assignment model formulation that is both capacity constrained as well as storage constrained. The formulation of this model is derived from a continuous time dynamic network loading model proposed in Bliemer and Raadsen (2018b). The prospect of being able to capture spillback effects in static assignment provides new opportunities for making this modelling method more capable. It is well known that the absence of spillback typically results in significant underestimation of path travel times. This is especially true for paths that do not traverse bottleneck(s) directly, but that are affected by the space occupied of queues that are spilling back. Similar to Smith (2013) and Smith et al. (2013), Bliemer and Raadsen (2018a) did not provide a solution algorithm. In this paper, we take their model formulation and propose a general solution scheme suitable for large scale networks.

KEY WORDS: ***Network loading, static traffic assignment, link transmission model, storage capacity, capacity constrained, network modelling, strategic transport planning***

AUTHORS: **Raadsen, Bliemer**

Acknowledgements: -

CONTACT: INSTITUTE OF TRANSPORT AND LOGISTICS STUDIES (H73)

The Australian Key Centre in Transport and Logistics Management

The University of Sydney NSW 2006 Australia

Telephone: +612 9114 1813

E-mail: business.itlsinfo@sydney.edu.au

Internet: http://sydney.edu.au/business/itls

DATE: November 2018

1 Introduction

Transport planning models are the most widely used method to forecast (future) traffic conditions on transport networks. They can broadly be categorised in three distinct categories based on their planning horizons. *Strategic planning* methods concerned with long term predictions, i.e. typically spanning 5 or more years into the future. This in contrast to *operational planning* approaches which look at short-term forecasts, ranging from a few hours to a few months. *Tactical planning* methods bridge the gap between strategic and operational models. Because strategic transport planning models adopt a long planning horizon, their forecasts are inherently subject to significant uncertainties. The adopted modelling techniques reflect this in the sense that they typically adopt a more aggregate approach than their short-term counterparts, mainly because it makes little sense to try and capture micro-effects such as vehicle acceleration, lane choice, and/or driver heterogeneity in this context. Consequently, traditional *static assignment* models still dominate this application area, even though their inception stems from the 1950s.

Traditional static assignment models based on the seminal work of Beckmann *et al.* (1956) have a number of appealing properties that suit strategic transport planning very well. They guarantee the existence and uniqueness of – equilibrium based – approaches when adopting a strictly monotone (increasing) cost function, also known as a link performance function. The readily available solution algorithms are computationally attractive compared to other more sophisticated assignment methods, and results, in terms of link flows and path costs, are easy to interpret due to the lack of an endogenous time dimension. However, the reasons that make this method so attractive are also its biggest drawback. While mathematically elegant, the traditional static assignment model is in many ways unrealistic compared to how traffic flow behaves. Its most notable drawbacks are: (i) the model allows link flows to exceed link capacities making the model *capacity restrained*, rather than *capacity constrained*. Therefore, queues that should form due to a lack of supply (in reality) are absent in this model. (ii) The adopted link performance functions cannot capture travel time based on – nor are they consistent with - queue formation. Instead, they estimate delay based on the v/c ratio, i.e. the ratio between flow (volume) and capacity. The resulting travel times are therefore not consistent with traffic flow theory for all but uncongested conditions. (iii) Link performance functions require additional calibration for their physically infeasible component (when flow exceeds capacity) which is both a time-consuming exercise and inherently inaccurate. (iv) Traditional static models capture delays inside the bottleneck rather than in front of the bottleneck, resulting in the delay being imposed in the wrong location. (v) Because queues are not modelled, spillback effects where queues grow beyond a link's available storage space, are not considered. Therefore, this model is not *storage constrained* either.

To get around some of these limitations alternative model formulations have been proposed over the last few decades, a discussion of which is to be found in Section 2. However, it has proven difficult to incorporate support for spillback in a static context. Yet, the importance of being able to capture spillback effects is readily apparent. Not doing so typically results in underestimation of path travel times. This is most obvious for travellers who are delayed due to queues that spill back but do not traverse any of the bottleneck(s) directly. In the absence of spillback effects this delay is not accounted for. In our companion paper Bliemer and Raadsen (2018a) a mathematical model formulation for a static model that does consider spillback is proposed. As far as the authors are aware, the only other *static* models attempting to capture spillback are found in Smith *et al.* (2013) and *Smith* (2013). However, in their work, they assume a stable queue that is not a result of the considered steady-state flow rates, necessitating the assumption that their (stable) queue originates from an unmodelled preceding period. In our work, the queues are consistent with the steady-state flow rates obviating the need to "choose" the queue, it simply follows from the link inflow and outflow rates.

1.1 Original contributions

Similar to Smith (2013) and Smith *et al.* (2013), Bliemer and Raadsen (2018a) do not provide a solution algorithm for their mathematical model formulation. In this study we propose a general solution scheme for this model that is suitable for general networks. This is the main contribution of this work. The resulting algorithm solves the static network loading problem capable of capturing spillback, i.e. is *storage constrained*, does not allow flows to exceed capacity, i.e. is *capacity constrained*, does not utilise link performance functions but adopts a Fundamental Diagram (FD) based approach, and places queues in front of bottlenecks instead of in them by incorporating a node model. This solution scheme is termed the *Static Link Transmission Model* (sLTM). It can be regarded as the static equivalent within the family of (dynamic) link transmission models that exist in the literature, even though the proposed solution scheme bears no resemblance to the existing dynamic LTM solution methods.

In some ways, the results of this study can be regarded as an extension to the point queue model formulation (and solution scheme) originally presented in Bliemer *et al.* (2014). There are however two major differences. First, the model in Bliemer *et al.* (2014) does not capture spillback effects, allowing for a far simpler solution method, mainly because traffic flow interactions only occur in a downstream direction through path flow propagation. By introducing spillback effects, new complex interactions occur via upstream propagating queues. Now an average steady-state solution with both forward and backward interaction effects needs to be constructed such that they are consistent with the underlying FD and model formulation. This is a far from trivial exercise as will become clear in the remainder of the text. Second, the novel model formulation can be relaxed to reduce to the special case that is the model in Bliemer *et al.* (2014). However, the resulting – simplified - solution algorithm is more capable than the original algorithm proposed in Bliemer *et al.* (2014) because it solves previously unsolvable networks as demonstrated via a special – rare - case where multiple solutions exist.

1.2 Assumptions and outline

To keep a clear focus, we make the following simplifying assumptions in the remainder of this work: (i) we only consider network loading, i.e. path flows are assumed known and given such that path choice, mode choice and/or departure time choice remain exogenous, (ii) we only consider a single user class, i.e. private vehicles, (iii) queue formation is conditioned on the prevailing link inflow and outflow rates, (iv) steady-state traffic flow is assumed to be consistent with a two-regime FD, and (v) intersections are modelled through a first order node model in which we do not consider signals.

This study is organised as follows: In Section 2 we discuss the current state of the literature with respect to capacity and/or storage constrained network loading models, followed by the model formulation of sLTM in Section 3. Section 4 describes the rationale behind the solution scheme that is being proposed. In Section 5 we demonstrate that a simplified version of this scheme can be regarded as a more capable version of the point-queue solution algorithm of Bliemer *et al.* (2014). We then proceed to discuss the base solution algorithm by including storage constraints in Section 6. Sections 7 and 8 introduce the two model extensions required to solve general networks. Parameter calibration and a conceptual comparison to existing static models is the topic of Section 9, while the large scale Gold Coast case study is discussed in Section 10. Conclusions and future research comprise the last section of this work. If the reader is mainly interested in implementing the proposed solution method, we kindly refer him/her to Appendix D, where we provide the full algorithm, including the extensions.

2 Capacity and storage constrained network loading in the literature

In this section we discuss the literature from two perspectives, the perspective where traditional static models are extended to incorporate additional features and the perspective where dynamic models – which already consider capacity and storage constraints - are simplified to yield static equivalents.

Early extensions to the traditional static model formulation introduced additional capacity constraints to the optimisation formulation, examples of which can be found in (Shahpar et al., 2008; Nie et al., 2004; Larsson and Patriksson, 1995; Yang and Yagar, 1994; Hearn, 1980). The resulting constrained optimisation problems were typically solved using penalty-based solution methods (inner penalties, outer penalties) or Lagrange multipliers. These models yield flows that no longer violate a link's physical capacity, but also do not actively withhold excess flows and instead reroute them to other links. In such models the inflow rate still equates to the outflow rate and queues are not considered. Lam and Zhang (2000) and Bifulco and Crisalli (1998) were among the first to explicitly model *residual queues*. Lam and Zhang withhold excess flows on so called shadow links, while Bifulco and Crisalli (1998) consider a stochastic user equilibrium setting and adopt an iterative solution scheme to solve their model. An early operational static approach with capacity constraints is found in Hungerink (1989), but this method lacks insight in any of the underlying model properties and is therefore difficult to assess. These methods extend or build upon the traditional modelling paradigm where both queues and/or delays emerge only within bottlenecks and storage constraints remain unconsidered. The only methods underpinned by a mathematical model and that do consider storage constraints are found in Smith (2013) and Smith *et al.* (2013), but unlike the model proposed in our companion paper, their queue formation is not consistent with the adopted flow rates, nor do they provide a solution algorithm for general networks. Further, there exist a few algorithm driven static

(and/or semi-dynamic[1]) models that do consider spillback (Davidson *et al.*, 2011; Bundschuh, 2006; Bakker *et al.*, 1994; van Vliet, 1982; Hall *et al.*, 1980), but due to their algorithmic nature little is known about the underlying model properties similar to their non-storage constrained counterparts, and they also place queues inside bottlenecks.

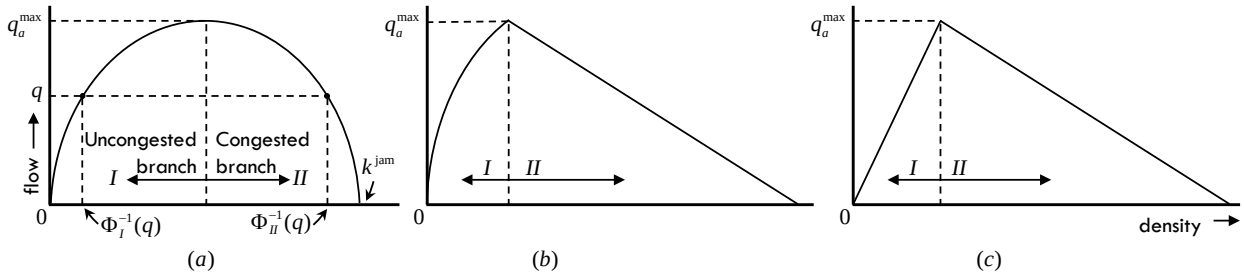
Recently, novel static model formulations are proposed that are directly derived from existing dynamic modelling approaches. This has the benefit that (dynamic) model features such as capacity and storage constraints can be transferred to the static context. One of the main motivations to do so is the fact that static models yield a single average result as a solution, while dynamic models do not. This coarser result dimension simplifies analysis and is a good fit with respect to applications that mainly consider long-term traffic forecasts. In such applications the time varying aspect of dynamic models is of less importance but being able to capture the effects of capacity and storage constraints are still desired. The first static assignment method to be derived explicitly from a dynamic model is found in Bliemer *et al.* (2014), as was demonstrated in Bliemer and Raadsen (2018b). This model incorporates a first order node model ensuring that residual queues emerge in front of bottlenecks instead of inside them. This type of model is gaining traction in applications as well, recent examples of which can be found in Brederode *et al.* (2018), or Tajtehranifard *et al.* (2018). However, the model proposed in Bliemer *et al.* (2014) is a point queue model and as such does not yet consider storage constraints. Another interesting method is proposed in Himpe *et al.* (2016) which is effectively the dynamic link transmission model, but one can adjust the discretised time step size beyond the commonly restricting Courant, Friedrich and Lewy (CFL) conditions (Courant *et al.*, 1928), which is achieved by conducting an iterative solution scheme. Hence, choosing a step size equal to the entire period should theoretically result in a static model result, although the model is clearly not formulated with this purpose in mind and might not solve either under such an extreme configuration.

3 Methodology and model formulation

Let us now briefly reiterate the original model formulation and the necessary notation required to discuss the sLTM solution scheme. For a more in-depth discussion of the model formulation we refer the interested reader to our companion paper (Bliemer and Raadsen, 2018a).

3.1 Network and Fundamental diagram

Consider transport network $\mathcal{G}(\mathcal{N},\mathcal{A})$, with links $a \in \mathcal{A}$ and nodes $n \in \mathcal{N}$. Each link has a length ℓ_a [km], maximum speed $\vartheta_a^{\max}$ [km/h], maximum throughput, i.e. capacity, $q_a^{\max}$ [veh/h], and maximum density, also known as jam density k_a^{jam} [veh/km]. We adopt a general concave two-regime concave FD. The FD has an uncongested branch (I) where density increases with increasing flow, and a congested branch (II) where density increases with decreasing flows. Density [veh/km] is denoted via inverse flux functions $\Phi_{I,a}^{-1}(q_a), \Phi_{II,a}^{-1}(q_a)$, for the uncongested and congested branch respectively, with flow rate [veh/h] denoted by $q_a \in [0, q_a^{\max}]$, see Figure 1(a). We utilise these inverse formulations because they allow us to formulate the model in terms of flow rates rather than densities.



(*a*) (*b*) (*c*)

Figure 1: (a) General concave two-regime fundamental diagram, with an uncongested branch and congested branch, (b) Quadratic-Linear (QL) fundamental diagram, (c) Triangular fundamental diagram.

Having two regimes, or branches, allows one to adopt specific functional forms of the FD for each branch separately. For example, one can construct a Quadratic-Linear form (Bliemer and Raadsen, 2018b; Smulders, 1987, 1990) as depicted in Figure 1(b), or the popular triangular FD (Newell, 1993). Even link

[1] In these semi-dynamic models consider multiple time slices, where each time slice is modelled via a static assignment procedure which has some form of interaction with its neighbouring period(s) by means of transferring demand and/or residual queues between consecutive periods.

performance functions can be considered as a special type of FD. In that case the congested branch becomes non-existent because density only increases with increasing flows, again highlighting why such an approach is inconsistent with how (congested) traffic flow operates.

3.2 Network loading

Analogous to most contemporary dynamic models, the network loading consists of a *link model* and a *node model*. The link model is responsible for the propagation of traffic flows along a homogeneous road section, i.e. link, while the node model distributes the incoming traffic flows, obtained from the link model, across the available outgoing links' capacities at intersections. Because we explicitly consider queue formation, we distinguish link inflow rates, denoted u_a [veh/h] from link outflow rates, denoted v_a [veh/h]. Consequently, the portion $\alpha_a \in [0,1]$ of accepted flow is given by $\frac{v_a}{u_a}$. In traditional static models, $\alpha_a = 1, \forall a \in \mathcal{A}$ since $u_a = v_a$. In our approach this is no longer necessarily the case.

3.2.1 Link Model

Travel occurs between origins and destinations via paths. Origin-destination pairs are captured via $w \in \mathcal{W}$ such that any path p connecting an origin to a destination is defined through $p \in \mathcal{P}^w$, with overall path set $\mathcal{P} = \bigcup_{w \in \mathcal{W}} \mathcal{P}^w$. Given that we only consider network loading, desired path flows are assumed known and given via $f_p, \forall p \in \mathcal{P}$. Further, path incidence indicator δ_{ap} yields zero if link a is not on path p and one otherwise. Link set $\mathcal{A}_{ap}$ contains all links preceding link a on path p. The link model is then formulated as follows:

$$u_{ap} = \delta_{ap} f_p \prod_{a' \in \mathcal{A}_{ap}} \alpha_a, \quad \text{with } \alpha_a = \tfrac{v_a}{u_a}, \quad \forall a \in \mathcal{A}, \forall p \in \mathcal{P}, \tag{1}$$

$$u_a = \sum\nolimits_{p \in \mathcal{P}} u_{ap}, \quad \forall a \in \mathcal{A}, \tag{2}$$

where the link-path inflow rate u_{ap} [veh/h] depends on the preceding path links and their acceptance factors. The construction of α depends on the node model function, denoted $\Gamma_n(\cdot)$.

3.2.2 Node model

The node model function distributes competing *sending flows* $\mathbf{s}_n = [s_a]_{a \in \mathcal{A}_n^-}$ [veh/h] from node n its incoming links $a \in \mathcal{A}_n^-$ over outgoing links $a \in \mathcal{A}_n^+, n \in \mathcal{N}$. It does so conditional on the available *receiving flows* $\mathbf{r}_n = [r_b]_{b \in \mathcal{A}_n^+}$ [veh/h] on the aforementioned outgoing links. The *splitting rates* $\boldsymbol{\varphi}_n = [\varphi_{ab}]_{a \in \mathcal{A}_n^-, b \in \mathcal{A}_n^+}$ convert the link sending flows to turn sending flows. Hence, the implicit (general) node model function's input and outputs are defined as follows:

$$\varphi_{ab} = \tfrac{1}{u_a} \sum\nolimits_{p \in \mathcal{P}} \delta_{bp} u_{ap} \quad \forall a \in \mathcal{A}_n^-, \forall b \in \mathcal{A}_n^+, \forall n \in \mathcal{N}, \tag{3}$$

$$(\mathbf{u}_n, \mathbf{v}_n) = \Gamma_n(\mathbf{s}_n, \mathbf{r}_n, \boldsymbol{\varphi}_n), \quad \mathbf{u}_n = [u_b]_{b \in \mathcal{A}_n^+}, \mathbf{v}_n = [v_a]_{a \in \mathcal{A}_n^-}, \mathbf{s}_n = [s_a]_{a \in \mathcal{A}_n^-}, \mathbf{r}_n = [r_b]_{b \in \mathcal{A}_n^+}, \boldsymbol{\varphi}_n = [\varphi_{ab}]_{a \in \mathcal{A}_n^-, b \in \mathcal{A}_n^+}, \tag{4}$$

where for any node $n \in \mathcal{N}$, the node model yields the incoming links' accepted outflows $\mathbf{v}_n$ and the outgoing links inflows $\mathbf{u}_n$, respectively. We make no assumptions on what node model function is adopted except for the fact that we assume it is compliant with a first order macroscopic node model following the conditions outlined in Tampère *et al.* (2011).

3.2.3 Link and node model interaction

Interaction between the link and node model occurs by transforming the (outgoing) link inflow rates to (downstream) sending flows. Because this is a static model and flow propagation is instantaneous, the sending flow is identical to the inflow rate and is only restricted by the link capacity. Receiving flows are constructed based on the link outflow rates, the available storage capacity, and are similarly bounded by the link capacity as well, via:

$$s_a = \min\{u_a, q_a^{\max}\}, \quad \forall a \in \mathcal{A}, \tag{5}$$

$$r_a = \min\{v_a + \tfrac{1}{T} \ell_a \Phi_{II,a}^{-1}(v_a), q_a^{\max}\}, \quad \forall a \in \mathcal{A}. \tag{6}$$

It is Equation (6) that makes this model storage constrained. The fact that the available receiving flow depends on the link's outflow rate results in an interdependency between the downstream and upstream

propagation of flows and queues. This dependency is a direct result of the derivation of this formulation from a dynamic model. In a dynamic context this interaction takes time to propagate due to the endogenous time dimension. Here however, the interactions are instantaneous and the time dimension is only used for scaling via $\frac{1}{T}$, where T [h] denotes the simulation duration. Intuitively, this equation is best interpreted as follows: the possible receiving flow must be larger or equal than the link's outflow rate v_a due to the conservation of vehicles. At the same time, the *congested* density on the link – in case of an explicit queue - is given by $\Phi_{II,a}^{-1}(v_a)$. Therefore, the maximum number of vehicles on the link in spillback conditions, i.e. the storage capacity is given by $\ell_a \Phi_{II,a}^{-1}(v_a)$. To make sure a link - with outflow rate v_a - that is in spillback has as many vehicles on the link as its storage capacity dictates, it must hold that the inflow rate equates to the receiving flow, and the receiving flow must exceed the outflow rate by exactly the storage capacity which in turn must be scaled by the simulation duration.

4 Solution scheme concept

The network loading model in equations (1)-(6) can be formulated as a fixed-point problem via the flow acceptance factor vector $\boldsymbol{\alpha}$. Bliemer and Raadsen (2018b) demonstrated this through $\boldsymbol{\alpha} = \Psi(\boldsymbol{\alpha}), \boldsymbol{\alpha} = [\alpha_a]_{a \in A}$. In their formulation, Ψ denotes the implicit (composite) self-mapping function. However, to introduce the reader to the rationale behind the proposed solution scheme it is preferable to provide a slightly modified fixed point formulation by decomposing the flow acceptance factor into its two original components, i.e. the inflow and outflow rates respectively. The resulting self-mapping is graphically depicted in Figure 2.

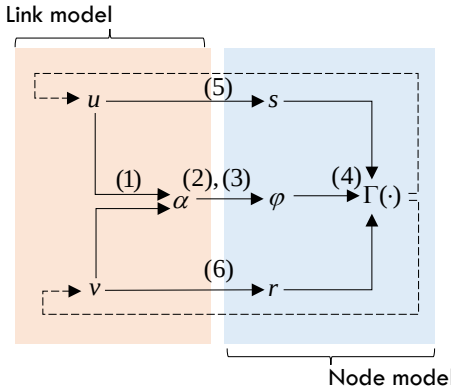


Figure 2:Self-mapping of inflow/outflow rates and the composition of variables (and node model function) involved, including the relevant equation references.

Due to the instantaneous propagation of flows, the link model is a straightforward conversion of desired path flows to link inflows. Instead, the complexity is found in the node model function where each of the three inputs independently influences the incoming links' outflows and outgoing links' inflows. A naïve solution approach would be to construct initial inflows and outflows which lead to the initial node model inputs, then conduct a node model update, which lead to updated inflows and outflows, which lead to updated node model inputs etc. etc., to the point convergence is reached. This approach would be in line with a basic iterative fixed-point algorithm. This is the method adopted in Bliemer *et al.* (2014) for solving their capacity constrained point queue model without storage constraints where $r_a = q_a^{\max}, \forall a \in A$. However, because the receiving flow is fixed there is relatively little instability in the system and therefore this naïve method works in all but some very rare cases, see also the next section. However, it was quickly found that when one considers explicit storage constraints, i.e. the receiving flow become dependent on the outflow, such that this solution scheme no longer suffices because it often does not converge due to flip-flopping. Significant additional effort is needed to address the instabilities caused by the - network wide - upstream and downstream interactions. These interactions are exacerbated by the static nature of the model because they all occur instantaneously unlike in dynamic models where there is an endogenous time lag.

In dynamic traffic assignment, some solution algorithms fix the splitting rates φ, either temporarily or during the whole simulation, even though this leads to (temporal) inconsistencies with the path flows (Raadsen *et al.*, 2016; Gentile, 2010). The benefit however, is found in a more stable model that solves more quickly. Following this line of thought, we propose to decompose the original fixed-point problem into two separate fixed-point problems, denoted $\mathbf{s} = \vec{\Psi}(\mathbf{s}), \mathbf{r} = \overleftarrow{\Psi}_I(\mathbf{r})$, respectively. The first fixed point sub-problem $\mathbf{s} = \vec{\Psi}(\mathbf{s})$ is responsible for the *downstream (forward) propagation consistency* of inflow

rates/sending flows conditional on the adopted node model. This first sub-problem adopts temporarily fixed splitting rates and receiving flows. The second fixed point sub-problem $\mathbf{r} = \bar{\Psi}_I(\mathbf{r})$ is responsible for the *upstream (backward) propagation consistency* of outflow rates/receiving flows conditional on the adopted node model. This second sub-problem adopts fixed splitting rates and sending flows. The fact that these two sub-problems are again fixed-point problems is graphically illustrated in Figure 3(b) and (d).

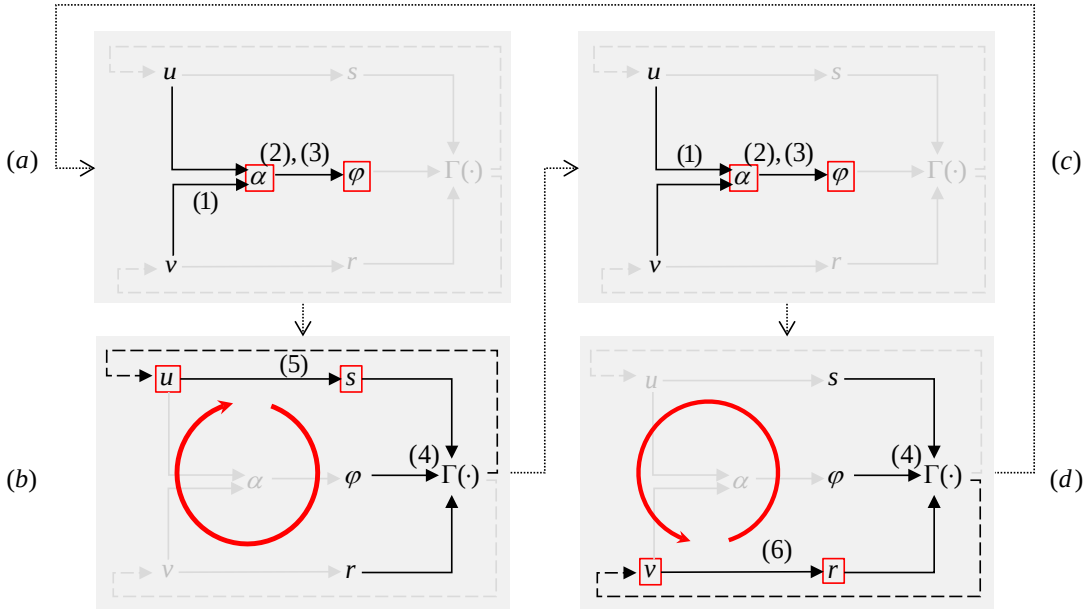


Figure 3: Decomposition of problem in two interconnected fixed point sub-problems, (a) and (c) connect the two fixed point problems via a path splitting rate update, (b) downstream propagation fixed point sub-problem, (c) upstream propagation fixed point sub-problem. Non-fixed variables highlighted in red (box).

Each sub-problem solution captures either updated inflow rates or outflow rates which in turn are used to update the splitting rates before continuing to the next sub-problem, as shown in Figure 3(a) and (c). It is worth noting that the update of splitting rates does not require an iterative process. The splitting rates follow from the path-based network loading procedure of (1) after the new flow acceptance factors have been computed. Hence, there is no involvement of the node model that causes a feedback loop. Clearly, this decomposition adds complexity to the solution algorithm, but it is needed to increase the stability of the (sub-)algorithm(s) such that we no longer experience the non-convergence one suffers when trying to solve the network loading problem. In the remainder of this work we further explore this solution approach, discuss the implementation, and justify its design choices, starting with how this solution scheme is both more general and more capable when it is deployed – in simplified form - to solve the point-queue model of Bliemer *et al.* (2014).

5 Point queue model

In a static point queue model, the steady-state inflow rates can exceed steady-state outflow rates. If so an explicit queue results which can grow infinitely without spilling back onto the upstream link boundary because $r_a = q_a^{\max}, \forall a \in \mathcal{A}$. Therefore, the steps in Figure 3(c) and (d) drop out. Instead, the sending flow update yields new inflows and outflows which in turn provide updated splitting rates, but only *after* local convergence has been reached. This, as shown in Figure 4(a), is different from the original solution algorithm in Bliemer *et al.* (2014) who propose an all-in-one iterative feedback loop in which both splitting rates and sending flows/inflows are updated simultaneously, as per Figure 4(b). Here, demand, network and paths are all symmetric, with the two paths each having a demand of 1000 veh/h. To find a solution, they initialise the flow acceptance factors α_a at 1 allowing all flow to pass through initially. Then a node model update is conducted. Here, only the two diverge nodes downstream of links 1 and 4 are affected by a lack of supply. They adopt the well-known diverge node model of Daganzo (1995) to distribute the sending flows via:

$$v_a = s_a \min_{b \in \mathcal{A}_n^+} \left\{ 1, \frac{q_b^{\max}}{s_a \varphi_{ab}} \right\} \quad a \in \mathcal{A}_n^-. \tag{7}$$

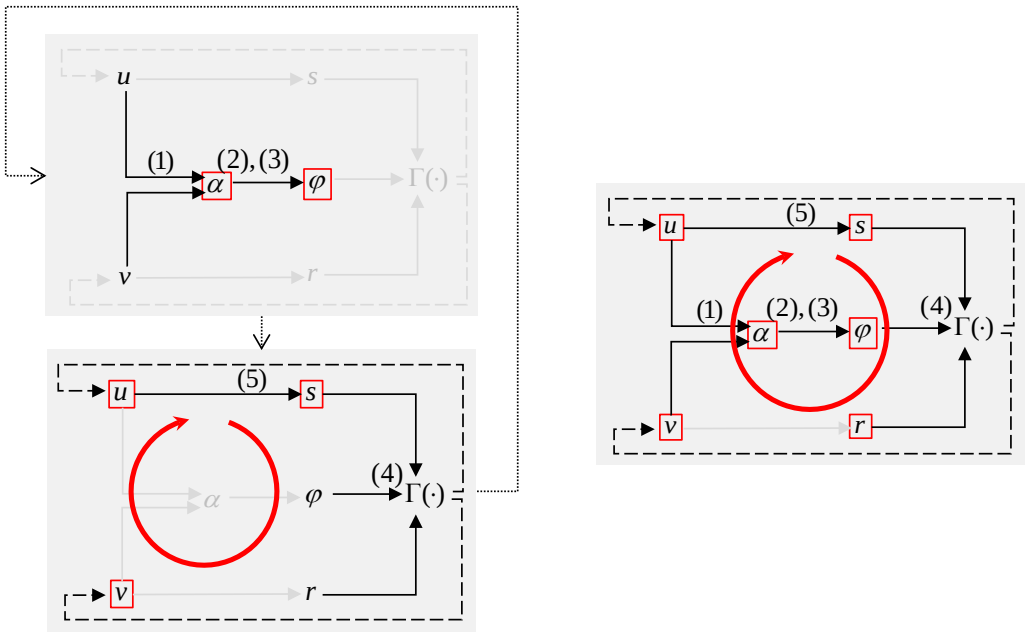


Figure 4: (a) Simplified 'two-step' point queue version of our proposed solution approach temporarily fixing splitting rates, and (b) original 'one-step' fixed point solution scheme proposed in Bliemer et al. (2014). Non-fixed variables highlighted in red (box).

In their original approach, the authors acknowledge that their solution scheme can lead to flip-flopping and non-convergence illustrated by their original example which we depict in *Figure 5*(a).

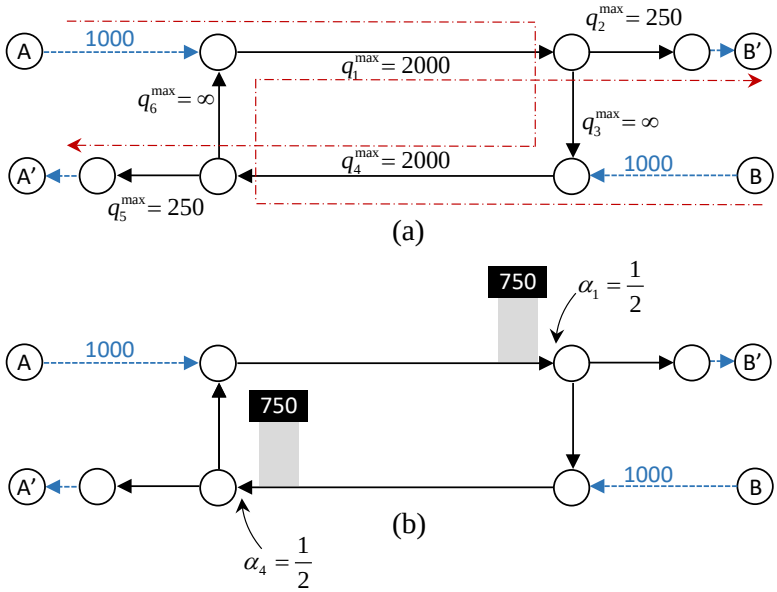


Figure 5: (a) non-converging example as per Bliemer et al. (2014), (b) which does solve utilising our simplified algorithm yielding the depicted symmetric solution.

As a result $v_1 = v_4 = 2000 \cdot \frac{250}{1000} = 500$ and based on (1) we find updated flow acceptance factors of $\alpha_1 = \alpha_4 = \frac{1}{4}$. However, since each path traverses both links 1 and 4, each path's flow is affected by both factors. Hence, the path flow is first reduced from 1000 → 250 at the first bottleneck and reduced further from 250 → 62.5 at the second bottleneck. As a result, the inflows/sending flows on the final link of each

path drop below the available capacity. Therefore, in the next iteration, the updated flow acceptance factors revert to their initial value and no solution can be found due to flip-flopping.

5.1 Solution scheme

As mentioned, we propose to avoid the above described non-convergence by temporarily fixing the splitting rates while searching for a consistent sending flow-based (intermediate) solution. By doing so, an overall solution to the above problem - which can alternatively be rewritten to $\alpha_1\alpha_4 = \frac{1}{4}$ - is found quickly. Initialisation is outlined in Algorithm 1. As before, no flow is initially withheld such that all inflows, outflows and sending flows equate to the summation of the desired path link inflows.

Algorithm 1: Point queue initialisation procedure.

Initialisation

(a) Initial link exit flow acceptance factors, $\alpha_a^0 = 1, \ \forall a \in \mathcal{A}$.

(b) Initial in/outflows via network loading, $v_a = u_a = \sum_{p \in \mathcal{P}} u_{ap}$, with $u_{ap} = \delta_{ap} f_p, \forall a \in \mathcal{A}$.

(c) Initial sending and receiving flows, $s_a = u_a, r_a = q_a^{\max}, \forall a \in \mathcal{A}$.

(d) Set iteration number $i = 1$.

(e) Continue to 1(b).

We then conduct path-based network loading conditional on the current flow acceptance factors, yielding link inflows. The new (path-based) link inflows provide updated splitting rates as shown in Algorithm 2, this constitutes the splitting rate update step of the algorithm.

Algorithm 2: Point queue splitting rate update through network loading.

Step 1 - splitting rate update

(a) Update inflows via network loading, $u_{ap} = \delta_{ap} f_p \prod_{a' \in \mathcal{A}_{pa}} \alpha_{a'}^{i-1}, \quad \forall a \in \mathcal{A}, \forall p \in \mathcal{P}$

(b) *Update splitting rates*, $\varphi_{ab} = \frac{1}{u_a} \sum_{p \in \mathcal{P}} \delta_{bp} u_{ap}$, with $u_a = \sum_{p \in \mathcal{P}} u_{ap}, \forall a, b \in \mathcal{A}$.

(c) Continue to Step 2(b).

The sending flow update that follows – and is described in Algorithm 3 - is an iterative (fixed-point) process that terminates upon convergence. Its increased stability is due to the *splitting rates remaining fixed*. Convergence, is tested via a configurable ε [-]. This internal iterative procedure does not rely on path-based network loading, but instead conducts local node model updates conditional on the splitting rates. This propagates flows through the network by updating each node's outgoing link inflows which then determine the sending flows of its subsequent downstream node(s).

Algorithm 3: Point queue sending flow update.

Step 2. sending flow fixed point – lock splitting rates

(a) Update link inflows via node model, $(\mathbf{u}_n, \mathbf{v}_n) = \Gamma_n(\mathbf{s}_n, \mathbf{r}_n, \boldsymbol{\varphi}_n), \forall n \in \mathcal{N}$.

(b) *Update sending flows*, $\tilde{s}_a = \min\{q_a^{\max}, u_a\}, \forall a \in \mathcal{A}$.

(c) Verify local convergence, $H = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} |\tilde{s}_a - s_a|$ and set $s_a = \tilde{s}_a, \forall a \in \mathcal{A}$.

(d) If $H > \varepsilon$ return to Step 2(a), otherwise continue to overall convergence check.

Overall network loading convergence is verified by comparing the previous' iteration flow acceptance factors with the updated values. If not converged, a new iteration is started until convergence has been reached as illustrated through Algorithm 4.

Algorithm 4: Point queue overall convergence check.

Verify overall convergence

(a) Update flow acceptance factors, $\alpha_a^i = \min\left\{1, \frac{v_a}{u_a}\right\}$.

(b) Verify overall convergence, $H = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \left| \alpha_a^{i+1} - \alpha_a^i \right|$.

(c) Increment iteration index $i := i + 1$.

(d) If not converged return to Step 1(a), else **done**, i.e. $H < \varepsilon$.

To illustrate that this method is able to solve the example, let us consider the results in Table 1, where we provide each algorithm step for each of the first three iterations. We only consider links 1-3 as the results

of link 4-5-6 are identical due to symmetry. Highlighted entries - in grey - reflect a change compared to a variable's previous value.

Table 1: First three iterations of solving the previously non-converging example first discussed in Bliemer et al. (2014).

Iteration	*Step*	*Description*	*Variable*	*Links* 1	2	3
$i=1$	Init	*acceptance factors*	α^0	1	1	1
		receiving flows	r	$q_1^{\max}=\infty$	250	$q_3^{\max}=\infty$
		network loading: *Inflows/sending flows*	$s=u=v$	2000	1000	1000
	1	*splitting rates*	φ	$\varphi_{1,2}=\varphi_{1,3}=\frac{1000}{2000}=\frac{1}{2}$		
	2	node model: outflows	v	500	250	250
		sending flows	$\tilde{s}=u$	2000	250	250
		node model: outflows	v	500	250	250
		sending flows[2]	$s=\tilde{s}=u$	1250	250	250
		acceptance factors	α^1	$\frac{500}{1250}=\frac{2}{5}$	1	1
$i=2$	1	network loading: inflows	u	1400	100	400
		splitting rates	φ	$\varphi_{1,2}=\frac{400}{1400}=\frac{2}{7}, \varphi_{1,3}=\frac{1000}{1400}=\frac{5}{7}$		
	2	node model: outflows	v	875	250	625
		sending flows	$\tilde{s}=u$	1250	250	625
		node model: outflows	v	875	250	625
		sending flows	$s=\tilde{s}=u$	1625	250	625
		acceptance factors	α^2	$\frac{875}{1625}=\frac{7}{13}$	1	1
$i=3$	1	network loading: inflows	u	1538.46	289.94	538.46
		splitting rates	φ	$\varphi_{1,2}=\frac{538.46}{1538.46}=\frac{7}{20}, \varphi_{1,3}=\frac{1000}{1538.46}=\frac{13}{20}$		
	2	node model: outflows	v	714.28	250	464.29
		sending flows	$\tilde{s}=u$	1625	250	464.29
		node model: outflows	v	714.28	250	464.29
		sending flows	$s=\tilde{s}=u$	1464.29	250	464.29
		acceptance factors	α^3	$\frac{714.29}{1464.29}\approx 0.49$	1	1
$i=4$	…	…	…	…	…	…

As can be seen the flow acceptance factors quickly converge towards the symmetric and intuitively logical solution of $\alpha_1=\alpha_4=\frac{1}{2}$. When the gap is chosen such that $\varepsilon=10^{-6}$ the algorithm terminates after 12 iterations, yielding point queues as depicted in Figure 5(b).

6 Capacity and Storage constraint model

Let us now discuss the full solution method by introducing storage constraints. To do so, we first provide a base solution scheme demonstrating the feasibility of our approach for a simple corridor network.

6.1 Flow acceptance factor decomposition

Before discussing the receiving flow and additional splitting rate update steps as depicted in Figure 3(c) and (d), we first provide an alternate formulation for flow acceptance factor α_a where without loss of generality we can state $\alpha_a=\frac{\beta_a}{\gamma_a}$, with $\beta_a=\frac{v_a}{r_a}$ and $\gamma_a=\frac{u_a}{r_a}, \forall a\in\mathcal{A}$, where we decompose α_a into a *flow capacity factor* $\beta_a\in[0,1]$ and *storage capacity factor* $\gamma_a\in[0,1]$, respectively. Besides our findings that this increases the flexibility in the solution method, this separation also provides a more meaningful insight in the state of the link. Whereas α_a only provides an indication whether a link is congested when $\alpha_a<1$ or uncongested when $\alpha_a=1$, we can now distinguish a range of different traffic states. When $\gamma_a=1$ for example, the inflow is constrained by the receiving flow such that the link is in spillback, with the notable exception when $\beta_a=1$, in that case the link is at capacity flow. Further, when $0<\gamma_a<\beta_a$ the link is in congestion, but not spilling back, while when both factors are zero the link is empty, see also Table 2. We

[2] Whenever we reach final (local) result we do not depict next internal iteration with identical results.

refer the reader to Appendix A for an updated visual depiction of the (conceptual) solution scheme replacing Figure 3, incorporating these two newly introduced factors.

Table 2: Link traffic states identified by (used) storage capacity factor γ_a and (used) flow factor β_a, respectively.

		Flow capacity factor		
		$\beta_a = 0$	$0 < \beta_a < 1$	$\beta_a = 1$
Storage cap. factor	$\gamma_a = 0$	No flow	-	-
	$0 < \gamma_a < 1$	Congestion, no spillback, no outflow	$\gamma_a < \beta_a$: Congestion, no spillback, non-zero outflow; $\gamma_a = \beta_a$: No congestion	-
	$\gamma_a = 1$	Spillback, no outflow	Spillback, non-zero outflow	Capacity flow

Using this alternative definition of the flow acceptance factor, the network loading in (1) is replaced by the following mathematically equivalent formulation:

$$u_{ap} = \delta_{ap} f_p \prod_{a' \in \mathcal{A}_{ap}} \alpha_a, \quad \text{with } \alpha_a = \frac{\beta_a}{\gamma_a} \text{ where } \beta_a = \frac{v_a}{r_a} \text{ and } \gamma_a = \frac{u_a}{r_a} \quad \forall a \in \mathcal{A}, \forall p \in \mathcal{P}. \tag{8}$$

6.2 Base solution scheme with storage constraints

The storage constraint solution scheme largely follows the initialisation, splitting rate, sending flow, and convergence criterion steps discussed in the previous section, i.e. Algorithms 1-4. To avoid repeating ourselves, we only discuss changes to existing algorithm components when needed. Two steps are added to ensure that storage constraints - consistent with (6) - are accounted for. First, another splitting rate update is conducted after the sending flow update. After that the - now variable - receiving flows are updated while temporarily fixing the splitting rates and sending flows. The full base solution algorithm is provided in Appendix B for the reader's convenience.

The changes required with respect to Algorithms 1-4 are minimal and mostly stem from the proposed decomposition of the flow acceptance factor, they comprise: (i) receiving flows are no longer initialised at capacity, but are conditioned on the initial outflow consistent with (6), (ii) network loading is conducted based on (8), replacing (1) and therefore the iteration specific flow acceptance factor α_a^i is replaced by γ_a^i and $\beta_a^i, \forall a \in \mathcal{A}$, accordingly, (iii) after the inflows/sending flows are found, we construct $\gamma_a^i = \frac{u_a}{r_a}$. Similarly, β_a^i becomes the result of the newly added receiving flow update step discussed in Section 6.2.2.

6.2.1 Splitting rate updates

After finding a (local) solution for the sending flows, one could attempt to immediately proceed with updating the receiving flows instead conducting another network loading/splitting rate update as proposed here, recall Figure 3(c). One could even argue that this would be a 'more natural' approach since each of the main variables, s_a, r_s, φ_a, then conducts exactly one update before repeating the procedure. However, we found this to be a suboptimal approach which often failed to converge, especially on large(r)-scale networks. The underlying reason, is likely due to the receiving flows being constructed based on splitting rates that are inconsistent with the sending flows because they have been constructed based on previous' iteration results. To avoid this, splitting rates should be updated whenever new information regarding either the inflows/sending flows or outflows/receiving flows is available. Therefore, we instead update splitting rates after the inflow/sending flow update as well as after the outflow/receiving flow update. This way, the sending flow and receiving flow sub-algorithms utilise up to date splitting rates while enjoying the added stability of temporarily fixing all but one of the node model inputs.

6.2.2 Receiving flow update

While the sending flow update is responsible for the forward propagation of inflows/sending flows, the receiving flow update conducts the backward propagation of queues by updating the outflow/receiving flows. For example, when the outflow of a link changes, its upstream receiving flow changes as well, as

per (6). Therefore, the upstream node's outflow might change which in turn affects its incoming links' receiving flows etc. The resulting fixed-point algorithm is provided in Algorithm 5.

Algorithm 5: Receiving flow update.

Step 4 - receiving flow update – lock splitting rates and sending flows

(a) Update link outflows via node model, $(\mathbf{u}_n, \mathbf{v}_n) = \Gamma_n(\mathbf{s}_n, \mathbf{r}_n, \boldsymbol{\varphi}_n), \forall n \in \mathcal{N}$.

(b) *Update receiving flows,* $r_a = \left\{ q_a^{\max}, v_a + \frac{1}{T}\ell_a \Phi_{II}^{-1}(v_a) \right\}, \forall a \in \mathcal{A}$.

(c) Verify local convergence, $H = \frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}} |\tilde{r}_a - r_a|$ and set $r_a = \tilde{r}_a, \forall a \in \mathcal{A}$.

(d) If not converged, i.e. $H > \varepsilon$, return to 4(a)

(e) *Update betas,* $\beta_a^i = \frac{v_a}{r_a}, \forall a \in \mathcal{A}$. , continue to overall convergence check

The outflows provided by the node model are utilised to iteratively find a consistent solution with respect to the receiving flows. Once this solution is found, the flow capacity factors $\beta_a^i, \forall a \in \mathcal{A}$ are updated which serve as the (partial) input for both the overall convergence check as well as the next splitting rate update. The result is a symmetric approach where each of the two internal fixed points serve one of the two propagation directions that need to be considered, separated by the distribution of these entities through intermediate splitting rate updates.

Let us illustrate how this algorithm works by considering the single path corridor network depicted in Figure 6. Depending on the adopted network loading model we find either no queues in Figure 6(a), point queues in Figure 6(b), or physical queues through sLTM in Figure 6(c).

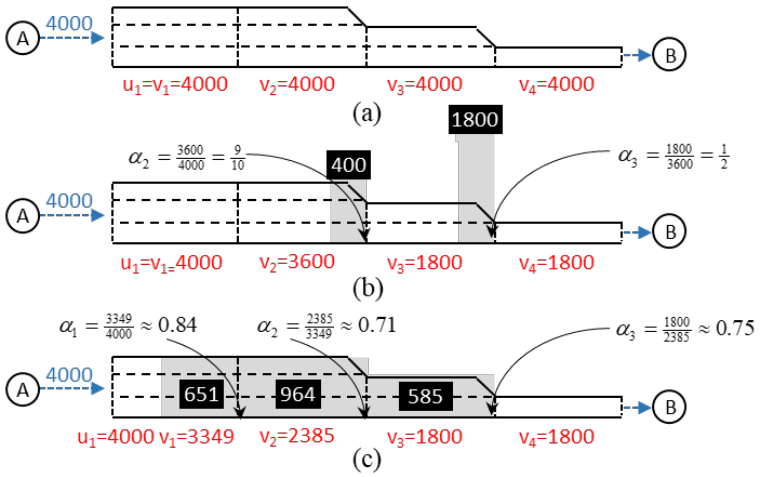


Figure 6: Corridor network, (a) traditional static assignment result, (b) point queue result following Section 5, (c) capacity and storage constrained result for the static link transmission model.

We adopt a triangular FD with $k_a^{\text{jam}} = 180$ [veh/km] and $q_4^{\max} = 1800$, $q_3^{\max} = 3600$, and $q_{1-2}^{\max} = 5400$ [veh/h]. In this simple example, the number of vehicles in both the point queue and physical queue model are identical, except for the fact that in the physical queueing model the queue takes up space and spills back as would happen in reality.

The steps to find this solution are outlined in Table 3. It is worth noting that we can easily identify that links 2 and 3 move to spillback, link 1 is merely congested, and link 4 is at capacity flow just by looking at the storage and flow capacity factors at the end of iteration 2. Further, in iteration 1, we see how the receiving flow first decreases on link 3, i.e. $r_3 = 2385$, reducing link's 2 outflow rate accordingly, in turn reducing the link 2 receiving flow, i.e. $r_2 \approx 3349$, forcing a reduction on the outflow on link 1. In other words, the receiving flow update indeed propagates information in the upstream direction in contrast to the sending flow update which propagates sending flows downstream.

Table 3: Constructing the static link transmission model solution for the corridor network in Figure 6(c).

Iteration	*Step*	*Description*	*Variable*	*Links*

				1	*2*	*3*	*4*
	Init	*acceptance factors*	$\alpha = \frac{\beta^0}{\gamma^0}$		$\frac{\beta^0}{\gamma^0} = \frac{1}{1} = 1$		
		receiving flows	r	4520	4520	3600	1800
		inflows/sending flows	$s = u = v$	4000	4000	4000	4000
	1	*splitting rates*[3]	φ		1		
	2	node model: outflows	v	4000	3600	1800	1800
		sending flows[4]	$s = \tilde{s} = u$	4000	4000	3600	1800
		storage capacity factors	$\gamma^1 = \frac{u}{r}$	$\frac{4000}{4520} \approx 0.88$	$\frac{4000}{4520} \approx 0.88$	$\frac{3600}{3600} = 1$	$\frac{1800}{1800} = 1$
	3	*acceptance factors*	$\alpha = \frac{\beta^0}{\gamma^1}$	$\approx 1.13 \rightarrow 1$	$\approx 1.13 \rightarrow 1$	1	1
$i = 1$		network loading: inflows	u	4000	4000	4000	4000
	4	node model: outflows	v	4000	3600	1800	1800
		receiving flows	$\tilde{r}$	4520	4230	2385	1800
		node model: outflows	v	4000	2385	1800	1800
		receiving flows	$\tilde{r}$	4520	3349.13	2385	1800
		node model: outflows	v	3349.13	2385	1800	1800
		receiving flows	$r = \tilde{r}$	4048.12	3349.13	2385	1800
		flow capacity factors	$\beta^1 = \frac{v}{r}$	$\frac{3349}{4048} \approx 0.83$	$\frac{2385}{3349} \approx 0.71$	$\frac{1800}{2385} \approx 0.75$	$\frac{1800}{1800} = 1$
		acceptance factors	$\alpha = \frac{\beta^1}{\gamma^1}$	≈ 0.93	≈ 0.80	≈ 0.75	1
	1	network loading: inflows	u	4000	3739.53	3009.21	2271.1
		node model: outflows	v	3349.13	2385	1800	1800
		sending flows	$s = \tilde{s} = u$	4000	3349.13	2385	1800
	2	*storage capacity factors*	$\gamma^2 = \frac{u}{r}$	$\frac{4000}{4048} \approx 0.99$	$\frac{3349}{3349} = 1$	$\frac{2385}{2385} = 1$	$\frac{1800}{1800} = 1$
	3	*acceptance factors*	$\alpha = \frac{\beta^1}{\gamma^2}$	≈ 0.84	≈ 0.71	≈ 0.75	1
$i = 2$		network loading: inflows	u	4000	3349.13	2385	1800
	4	node model: outflows	v	3349.13	2385	1800	4000
		receiving flows	$r = \tilde{r}$	4048.12	3349.13	2385	1800
		flow capacity factors	$\beta^2 = \frac{v}{r}$	$\frac{3349}{4048} \approx 0.83$	$\frac{2385}{3349} \approx 0.71$	$\frac{1800}{2385} \approx 0.75$	$\frac{1800}{1800} = 1$
		acceptance factors	$\alpha = \frac{\beta^2}{\gamma^2}$	≈ 0.84	≈ 0.71	≈ 0.75	1

7 Extension 1: Supporting overlapping alternative routes with spillback

The solution algorithm presented in the previous section can solve the sLTM on a simple corridor. However, it does not suffice as a general-purpose solution method yet due to the complexities of capturing spillback under instantaneous flow propagation. Consider the example network in Figure 7(a), where $T = 1, \ell_a = 4, \vartheta_a^{\max} = 100, \forall a \in \mathcal{A}$. Also, the maximum density is 180 veh/km/lane with all links having four lanes, except for link 2 which has only 1 lane. The network has one low capacity route, i.e. an older motorway through the city centre, and two identical high capacity routes (with equal demand), i.e. two newly built motorways around the city centre. The three paths merge on link 5 with the total demand exceeding the supply on that link.

Of interest is the merge node which distributes the sending flows s_2, s_3, s_4, respectively across the available supply. Assuming the merge node model proposed in Daganzo (1995) where we choose the priority of each incoming link based on its capacity, in line with Tampère *et al.* (2011), we find, as depicted in Figure 7(b), that only on link 2 a queue starts to form. Based on (6) we initially find that *on average* this link also spills back. It spills back because the storage capacity is reduced due to the reduced outflow which in turn puts a restriction on the receiving flow which now exceeds the initial inflow. Hence, the diverge node on the upstream border of link 2 reduces the flow into link 2, but due to the First-In-First-Out (FIFO) principle of

[3] We do not depict any further updates of splitting rates as they remain one due to corridor structure.
[4] Whenever we reach final (local) result we do not depict next internal iteration with identical results.

the model this also reduces the flows into links 3 and 4 with the same proportion. This results in a queue withholding flow for all three routes, see Figure 7(c). Consequently, the sending flow offered at the merge node now drops below the available supply, i.e. $s_1 + s_2 + s_3 \leq r_5$, dissipating the queue on link 2 to the point that it no longer spills back. When this happens, the diverge node's accepted flows revert to their initial state causing the queue on link 2 to start growing again etc. etc. In a dynamic setting this causes an infinite back-and-forth between a growing and shrinking queue state, but in a static model, this (legitimate) back-and-forth behaviour manifests itself as internal flip-flopping which complicates matters when one tries to construct a single average solution. Even though this is a very specific example constructed to demonstrate this behaviour, it should be obvious that this type of situation does occur, albeit possibly through a far more complex set of interactions than shown here.

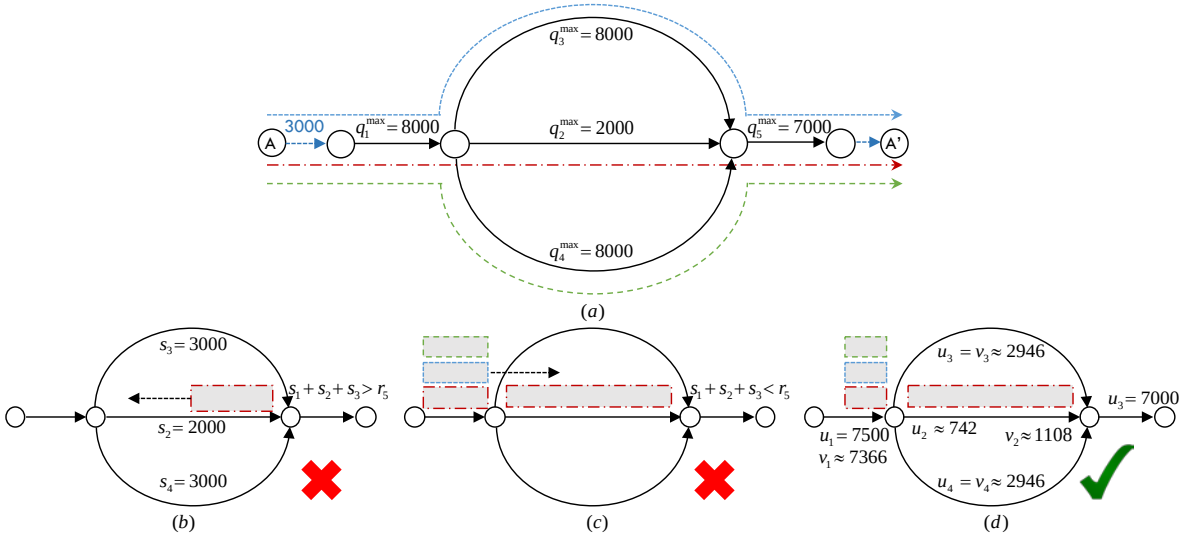


Figure 7: (a) example network with continuously growing and shrinking queue due to upstream and downstream traffic flow interactions, (b) growing queue state before spillback, (c) shrinking queue state when in spillback, (d) steady-state solution with link 2 in spillback and link 1 in congestion.

Table 4 demonstrates explicitly how the base solution algorithm suffers from the above described behaviour, switching between a growing and shrinking queue and reverting to the initial sending flows after three iterations as highlighted in red (link 5 its sending flow is slightly different, but it exercises no influence on the system).

Table 4: Lack of convergence under unstable conditions utilising the initial sLTM solution algorithm presented in Section 6.

Iteration	*Step*	*Description*	*Variable*	*Links* 1	2	3 and 4	5
$i=1$	Init	*acceptance factors*	$\alpha = \frac{\beta^0}{\gamma^0}$	$\frac{\beta^0}{\gamma^0} = \frac{1}{1} = 1$			
		receiving flows	r	7980	1740	4920	7000
		inflows/sending flows	$s = u = v$	7500	1500	3000	7500
	1	*splitting rates*[5]	φ	$\varphi_{1,2} = \frac{1500}{7500} = \frac{1}{5}$ and $\varphi_{1,3} = \varphi_{1,4} = \frac{3000}{7500} = \frac{2}{5}$			
	2	node model: outflows	v	7500	1000	3000	7000
		sending flows[6]	$s = \tilde{s} = u$	7500	1500	3000	7000
		storage capacity factors	$\gamma^1 = \frac{u}{r}$	$\frac{7500}{7980} \approx 0.94$	$\frac{1500}{1740} \approx 0.8($	$\frac{3000}{4920} \approx 0.61$	$\frac{7000}{7000} = 1$
	3	*acceptance factors*	$\alpha = \frac{\beta^0}{\gamma^1}$	$\approx 1.06 \rightarrow 1$	$\approx 1.16 \rightarrow$	$\approx 1.64 \rightarrow 1$	1
		network loading: inflows	u	7500	1500	3000	7000
	4	node model: outflows	v	7500	1000	3000	7000
		receiving flows	$\tilde{r}$	7980	1400	4920	7000
		node model: outflows	v	7000	1000	3000	7000

[5] Splitting rates remain constant in this example, so subsequent updates are ignored in this table
[6] Whenever we reach final (local) result we do not depict next internal iteration with identical results.

		receiving flows	$r = \tilde{r}$	7640	1400	4920	7000
		flow capacity factors	$\beta^1 = \frac{v}{r}$	$\frac{7000}{7640} \approx 0.92$	$\frac{1000}{1400} \approx 0.7$	$\frac{3000}{4920} \approx 0.61$	$\frac{7000}{7000} = 1$
		acceptance factors	$\alpha = \frac{\beta^1}{\gamma^1}$	≈ 0.97	≈ 0.83	1	1
	1	network loading: inflows	u	7500	1462.3	2924.6	7060.83
		node model: outflows	v	7000	1000	3000	7000
		sending flows	$\tilde{s} = u$	7500	1400	2800	7000
	2	node model: outflows	v	7000	1400	2800	7000
		sending flows	$s = \tilde{s} = u$	7500	1400	2800	7000
		storage capacity factors	$\gamma^2 = \frac{u}{r}$	$\frac{7500}{7640} \approx 0.98$	$\frac{1400}{1400} = 1$	$\frac{2800}{4920} \approx 0.57$	$\frac{7000}{7000} = 1$
	3	*acceptance factors*	$\alpha = \frac{\beta^1}{\gamma^2}$	≈ 0.93	≈ 0.71	≈ 1.07	1
$i = 2$		network loading: inflows	u	7500	1400	2800	6600
		node model: outflows	v	7000	1400	2800	6600
		receiving flows	$\tilde{r}$	7640	1672	4784	6946.29
	4	node model: outflows	v	7500	1346.29	2800	6600
		receiving flows	$r = \tilde{r}$	7980	1635.47	4784	6946.29
		flow capacity factors	$\beta^2 = \frac{v}{r}$	$\frac{7500}{7980} \approx 0.94$	$\frac{1346}{1635} \approx 0.8$	$\frac{2800}{4784} \approx 0.59$	$\frac{6600}{6946} \approx 0.95$
		acceptance factors	$\alpha = \frac{\beta^2}{\gamma^2}$	≈ 0.96	≈ 0.82	$\approx 1.03 \rightarrow 1$	1
$i = 3$	1	network loading: inflows	u	7500	1436.1	2872.18	6926.52
	2	node model: outflows	v	7500	1346.29	2800	6946.29
		sending flows	$s = \tilde{s} = u$	7500	1500	3000	6946.29

It is important to realise that in reality this scenario is unlikely to ever reach a point with stable flow rates but there does exist an *average steady-state* solution to this problem when modelled statically, see Figure 7(d). We leave it to the reader to verify that in this case indeed $u_2 = r_2 = v_2 + \ell_2 \Phi_{II}^{-1}(v_2)$. However, to find this solution we can no longer directly transfer each intermediate result to the next algorithm step. Instead, a configurable step-size is required which takes a more conservative step in the direction of the suspected solution. In existing static traffic assignment methods this is common practice with respect to finding equilibrium (in route choice), the most well-known approach being the Franke-Wolfe algorithm (Frank and Wolfe, 1956) which determines an optimal step size in each iteration to shift flows from one path to the other conditional on adopting a strictly monotone and convex cost function. Often, contemporary traffic assignment methods rely on some form of the Method of Successive Averages (MSA), see for example Brederode *et al*. (2018), which is simpler, computationally more attractive, and can also be used when the underlying objective function does not exhibit attractive mathematical properties. Although in that case the uniqueness of the result is no longer guaranteed.

Given the highly non-linear nature of our model due to the adoption of a general node model, we are not able to construct a method that determines an optimal step size. Also, an MSA like method is undesirable because in our context it often leads to an infeasible result where the averaged storage capacity is for example not consistent with the averaged outflow rate. In other words, while MSA guarantees convergence the result might not be a valid solution. Hence, we propose a straightforward fixed step-size applied after each of the three main algorithm components. The smoothing applied to the splitting rate result after Steps 1 and 3 is given by:

$$\varphi_{ab}^{i} := (1-\lambda_1)\varphi_{ab}^{i-1} + \lambda_1 \varphi_{ab}, \quad \forall a,b \in \mathcal{A}, \tag{9}$$

where $\lambda_1 \in [0,1]$ is the splitting rate specific *fixed* step-size and φ_{ab}^{i} is the *smoothed splitting rate*. Subsequently, we conduct smoothing on the result of the sending and receiving flow fixed-point result, albeit indirectly via the storage capacity and flow capacity factors respectively such that:

$$\gamma_a^i := (1-\lambda_2)\gamma_a^{i-1} + \lambda_2 \tfrac{u_a}{r_a}, \quad \text{and } \beta_a^i := (1-\lambda_3)\beta_a^{i-1} + \lambda_3 \tfrac{v_a}{r_a}, \quad \forall a \in \mathcal{A}, \tag{10}$$

with dedicated step-sizes $\lambda_1, \lambda_2 \in [0,1]$, respectively. We deliberately do not smooth the sending and receiving flows directly because in that case the (non-smoothed) end state of the sending flow update becomes inconsistent with (the smoothed sending flow of) the start state of the receiving flow update. This

inconsistency is avoided when smoothing the factors instead. Calibration of λ_1, λ_2, and λ_3 is discussed in Section 9. The solution algorithm including smoothing factors is provided in Appendix C for reference.

8 Extension 2: Supporting spillback on general networks

While the introduction of smoothing improves the likelihood of attaining overall convergence, it does not improve convergence rates of the internal fixed point sub-steps. With respect to the sending flow fixed point approach - discussed in Section 5 - this is not an issue as our method improves upon the existing, and practically applied, method in Bliemer *et al.* (2014). However, finding a solution for the receiving flow fixed point is much more difficult for general networks. This difficulty is due to the possibility of a 'snowball', or queue multiplication effect where a queue that spills back on one link might generate more congestion on its upstream links. For example, flows initially not impacted when diverting at the link's upstream node are affected once the link moves to a spillback state (whenever FIFO is partially or fully considered). Hence, a small change in receiving flows in one iteration can lead to a larger change in the next iteration violating the conditions for a contraction mapping. To avoid this, we a-priori estimate the extent of this multiplication effect and utilise it to limit the allowed change in receiving flow. In other words, we aim to avoid that a backward propagating queue - due to spillback - increases in size leading to non-convergence. To achieve this, we no longer directly update the receiving flow $\tilde{r}_a$ based on (6) but instead 'intelligently' nudge it in the right direction as much as possible via:

$$\tilde{r}_a := R_a - \frac{\Delta r_a}{\mu_a}, \quad \text{with } \Delta r_a = R_a - \min\left\{q_a^{\max}, v_a + \tfrac{1}{T}\ell_a \Phi_{II}^{-1}(v_a)\right\}, \forall a \in \mathcal{A}, \tag{11}$$

where $R_a, a \in \mathcal{A}$ is the reference flow being nudged, while the magnitude of the desired change - consistent with (6) - is given by Δr_a [veh/h]. This desired change is then dampened by $\frac{1}{\mu_a} \in [0,1]$, resulting in the nudging aiming to avoid local non-convergence. The benefit of this procedure is found in that we can, a-priori to the receiving flow fixed point, estimate the queue multiplication factor $\mu_a, \forall a \in \mathcal{A}$ [-] on a per link basis. Let us illustrate how to construct μ_a through the intersection depicted in Figure 8(a). To quantify the extent of the unwanted queue growth, we require 'regular' splitting rates (Figure 8(b)) as well as *backward splitting rates*, denoted $\overleftarrow{\varphi}_{ab} \in [0,1]$. Backward splitting rates determine the portion of flow on the outgoing link that originated from the incoming link, see Figure 8(c). We compute the backward splitting rates via:

$$\overleftarrow{\varphi}_{ab} = \frac{v_a \varphi_{ab}}{u_b}, \quad \forall a, b \in \mathcal{A}. \tag{12}$$

Now consider the situation that link 3 in the example starts to spill back leading to a reduction of its receiving flow from 200 to 100, i.e. $u_3 = r_3 = 100$. In that case[7] $v_{1,3} : 50 \rightarrow 25$, Then, due to FIFO and the conservation of turning fractions the flow on the other turn also halves, i.e. $v_{1,2} : 950 \rightarrow 475$. So, outflow on link 1 drops from $1000 \rightarrow 500$ due to a 25 veh/h reduction on the turn towards link 3. Hence, the queue multiplies by a factor 20 when propagating upstream from link 3 to 1. Similarly, on link 2, we find a queue multiplication factor of 2, i.e. $v_{2,3} : 150 \rightarrow 75$, hence $v_{2,4} : 150 \rightarrow 75$, i.e. the 75 veh/h reduction on the turn towards 3 results in 150 veh/h reduction on link 2.

[7] Assuming the node model function reduces link 1's outflow proportionally which is a justifiable assumption since it results in the worst-case multiplication factor.

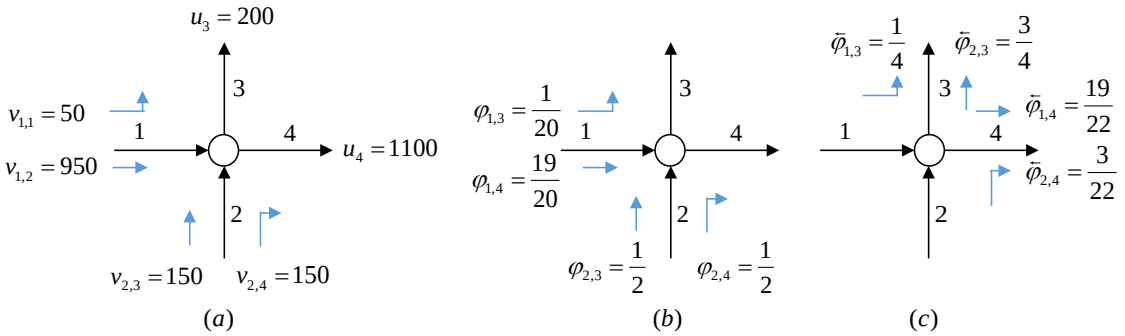


Figure 8: (a) Intersection with quantifiable 'snowballing effect' on link 1 based on regular and backward splitting rates., (b) regular splitting rates, (c) backward splitting rates.

We find that the total multiplication factor of queues due to spillback on link 3 is $(500+150)/100 = 6\frac{1}{2}$, which we can alternatively obtain via the two splitting rates, i.e. $\frac{\tilde{\varphi}_{1,3}}{\varphi_{1,3}} + \frac{\tilde{\varphi}_{2,3}}{\varphi_{2,3}} = 5 + 1\frac{1}{2} = 6\frac{1}{2}$. In general, this multiplication factor is therefore obtained via:

$$\mu_b = \sum_{a \in \mathcal{A}_n^-} \frac{\tilde{\varphi}_{ab}}{\varphi_{ab}}, \forall b \in \mathcal{A}_n^+, \forall n \in \mathcal{N}, \tag{13}$$

with $\mu_a \in [0,1], \forall a \in \mathcal{A}$. As can be seen in our example, utilising the reciprocal of this multiplication factor ensures that a *desired* change of $\Delta r_3 = 100$ veh/h, results in a *reduced propagated change* of $\tilde{r}_3 = 200 - \frac{100}{6\frac{1}{2}} \approx 185$ veh/h, and leads to combined additional queues of no more than 100 veh/h in total. With this adjustment in place receiving flow changes now 'dilute' in a similar way to how sending flow changes decrease when propagating through nodes and being distributed across exit links via splitting rates as discussed in Section 5.

The price one pays for applying this method is that the resulting fixed-point solution, even though receiving flows move towards (6), are in fact not (yet) consistent with (6). However, with each global iteration they move closer until it is considered sufficient based on the chosen gap. This therefore does require more iterations on a relatively uncongested network, but in congested networks it can avoid non-convergence in the local receiving flow fixed-point algorithm that otherwise makes it impossible to find a solution. To embed this procedure in the solution method, (12) and (13) are incorporated in the second splitting rate update (Step 3) before commencing the receiving flow update, as shown in Algorithm 6.

Algorithm 6: Second splitting rate update including backward splitting rate estimate and multiplication factor.

Step 3 - splitting rate update

(a) Update inflows via network loading, $u_{ap} = \delta_{ap} f_p \prod_{a' \in \mathcal{A}_{pa}} \alpha_{a'}$, with $\alpha_a = \frac{\beta_a^{i-1}}{\gamma_a^i}$, $\forall a \in \mathcal{A}, \forall p \in \mathcal{P}$

(b) *regular splitting rates*, $\varphi_{ab} = \frac{1}{u_a} \sum_{p \in \mathcal{P}} \delta_{bp} u_{ap}$, with $u_a = \sum_{p \in \mathcal{P}} u_{ap}, \forall a,b \in \mathcal{A}$.

(c) *Update smoothed splitting rates*, $\varphi_{ab}^i := (1-\lambda_1)\varphi_{ab}^{i-1} + \lambda_1 \varphi_{ab}, \forall a,b \in \mathcal{A}$.

(d) Estimate multiplication factor, $\mu_b = \sum_{a \in \mathcal{A}_n^-} \frac{\tilde{\varphi}_{ab}}{\varphi_{ab}}$, with $\tilde{\varphi}_{ab} = \frac{v_a \varphi_{ab}^i}{u_b}; v_a = u_a \alpha_a, \forall b \in \mathcal{A}$, continue to 4(a)

Further, (11) now drives the updated local receiving flow as shown in Algorithm 7. We choose $R_a = u_a$ for best performance. This way, when the storage capacity restricts a link's inflow, the inflow/receiving flow is nudged towards a (smaller) value consistent with (6), while otherwise the result is nudged towards a (larger) value[8] such that $\tilde{r}_a \geq u_a$. The complete solution scheme with both extensions in place is provided in Appendix D for reference. Finally, observe how the backward splitting rates – which we compute once before Step 4 - depend on the outflow rates. Given that outflow rates are updated in Step 4, we could alternatively update the backward splitting rates (and multiplication factor) within the receiving flow fixed point loop. However, we choose not to do this because (i) it is computationally more expensive, (ii) we found that an a-priori (fixed) factor solves the local non-convergence issue satisfactorily, and (iii) it defies

[8] Alternatively, one can also choose to set R_a to the previous' iteration receiving flow, but in our experience, this more often leads to non-convergence. We point out that both approaches yield the same result upon convergence because the inflow/sending flow result is only affected by the receiving flow when the receiving flow is restricting. In this case both choices for R_a lead to the same result consistent with (6).

the design rationale of fixing all but one variable within each step to maximise algorithm stability and increase the likelihood of convergence within each local fixed-point.

Algorithm 7: Receiving flow update incorporating queue growth dilution method.

Step 4 - receiving flow update – lock splitting rates and sending flows

(a) Update link outflows via node model, $(\mathbf{u}_n, \mathbf{v}_n,) = \Gamma_n(\mathbf{s}_n, \mathbf{r}_n, \boldsymbol{\varphi}_n^i), \forall b \in \mathcal{A}_n^+, \forall n \in \mathcal{N}.$

(b) *Update receiving flows,* $\tilde{r}_a = u_a - \frac{1}{\mu_a}\Delta r_a,$ with $\Delta r_a = u_a - \min\left\{q_a^{\max}, v_a + \frac{1}{T}\left(\ell \Phi_{II}^{-1}(v_a)\right)\right\}, \forall a \in \mathcal{A}.$

(c) Verify local convergence, $H = \frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}}|\tilde{r}_a - r_a|$ and set $r_a = \tilde{r}_a, \forall a \in \mathcal{A}.$

(d) If not converged, i.e. $H > \varepsilon$, return to 4(a)

(e) *Update smoothed betas,* $\beta_a^i := (1-\lambda_3)\beta_a^{i-1} + \lambda_3 \frac{v_a}{r_a}, \forall a \in \mathcal{A}.$, continue to overall convergence check

9 Model comparison and parameter calibration

In this section we calibrate the step-size variables introduced in Section 7, i.e. $\lambda_1, \lambda_2,$ and λ_3. We aim to find a combination of step-sizes which under worst-case conditions find solutions and if they do, do so in the least amount of time/iterations. A worst-case scenario in this context implies a heavily congested network that is severely affected by spillback. However, as long as queues only spillback onto upstream *uncongested* links, finding a solution is relatively straightforward, recall for example the corridor network in Section 6. However, once queues spill back to the point that they reach *their own origin* and start 'biting their own tail', finding a consistent average steady-state solution becomes truly challenging. We point out that this is not just a hypothetical scenario, it occurs frequently on for example roundabouts, in congested metropolitan areas, and under non-recurrent traffic conditions such as incidents. It is also a common side-effect of early iterations of traffic assignment models, when one has only poor estimates of route choice behaviour. In these conditions, queues that spill back cause circular dependencies that instantaneously (in our model) affect both upstream and downstream flow rates. We argue it therefore qualifies as an ideal scenario to test which step-sizes are still viable to use. To construct such a scenario, we utilise a square grid network of configurable size as depicted in Figure 9.

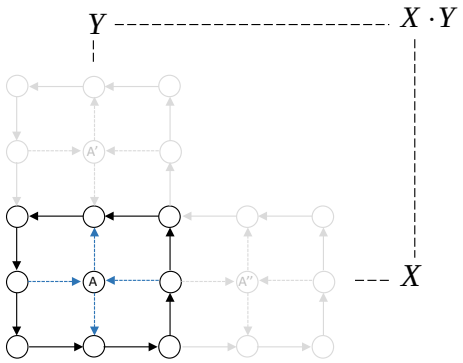


Figure 9: Generic grid network structure with uni-directional links and XY zones/centroids, A, A', A'' etc. Dashed links are connector links and solid lines represent physical links.

Links have two lanes, adopt a triangular FD, have a capacity $q_a^{\max} = 2 \cdot 1800 = 3600,$ maximum density $k_a^{\text{jam}} = 2 \cdot 180 = 360$ maximum speed $\vartheta_a^{\max} = 50,$ length $\ell_a = 1, \forall a \in \mathcal{A},$ except for connector links which have infinite length to ensure that all demand can enter the network. Simulation duration is set to $T = 2,$ while the convergence gap is set to a commonly argued acceptable value of $\varepsilon = 10^{-6}$. The node model function is based on Tampère *et al.* (2011). Od-pairs are initialised with a random number of trips $\in [0,1],$ and are (uniformly) scaled up to the point that the network exhibits severe congestion in its centre. We adopt an All-Or-Nothing (AON) routing strategy where each od-pair only selects a single fastest route to load its demand on, since we leave the incorporation of route choice for future research.

9.1 Illustrative example of adopting different model types

Before discussing the results of the step-sizes let us illustrate the significant differences in results that can occur between various static modelling paradigms. For this example we adopt a grid with 100 zones, i.e. 10,000 od-pairs, where $X = Y = 10.$ Under our chosen demand scenario a traditional static model identifies a number of bottleneck links where demand exceeds capacity, indicated by the red dots in Figure 10(a). With capacity constraints in place, the point-queue model discussed in Section 5 yields explicit non-spatial

queues (that do not spill back) in front of the bottleneck links depicted in grey, see Figure 10(b). There are less bottlenecks in this model since flow is withheld from propagating further when supply is insufficient. The results of sLTM are depicted in Figure 10(c). Because the (point) queues exceed the storage capacity on their respective links, it results in queues spilling back when imposing storage constraints. In this highly saturated example this occurs to the point that they spillback onto their origin causing circular dependencies in both directions and severe degradation of the network throughput. This is similar to what one would observe when adopting a dynamic network loading model, only without the explicit time dynamics. It demonstrates how the inclusion of storage constraints can drastically affect modelling results, providing benefits in terms of capturing real-world phenomena such as spillback, but bringing with it the complexities of increased flow interdependencies.

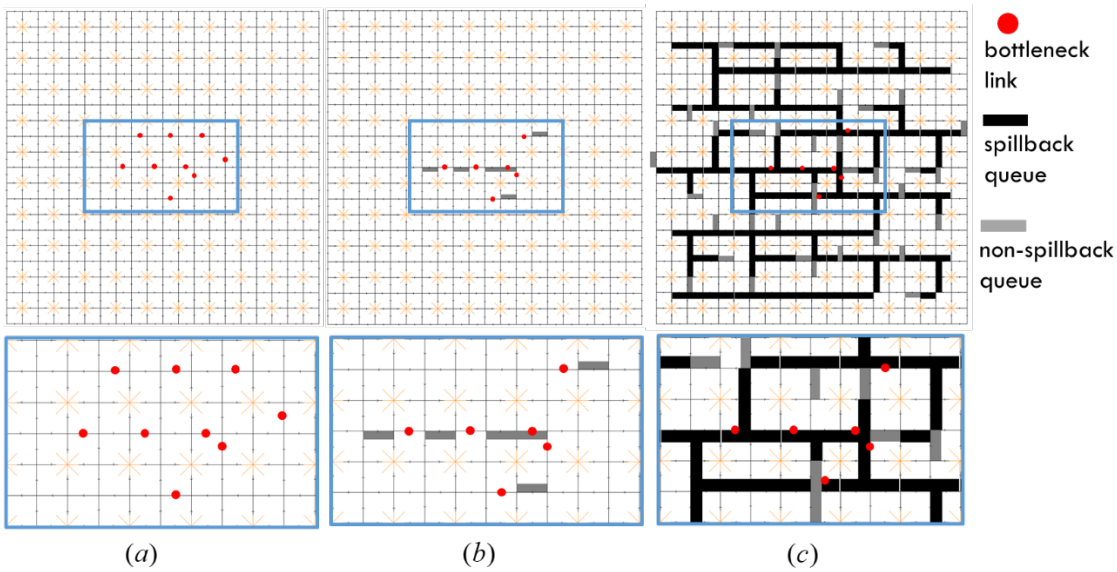


Figure 10: Comparing impact of link saturation in different models, (a) traditional static assignment with bottleneck links where demand exceeds capacity, (b) capacity constrained point-queue model solution as per Section 5, where links have physical queues that do not spill back, (c) static link transmission model result, queues do spill back when exceeding storage constraints.

9.2 Parameter calibration

To calibrate our step-size variables we adopt a similarly congested scenario, only with a larger grid network comprising 400 zones, i.e. 160,000 od-pairs, where $X = Y = 20$. Modelling a total of $\approx 208,000$ trips equating to an average of 1.3 trips per od-pair. We considered the following step-sizes, $\lambda_1 \in \{0.1, 0.25, 0.5, 0.75, 1\}$ for splitting rate smoothing and $\lambda_2, \lambda_3 \in \{0.1, 0.2, \cdots, 1.0\}$, for both the sending flow (applied to storage capacity factor γ_a) and receiving flow (applied to flow capacity factor β_a) smoothing. In total 500 simulations were conducted of which the results are depicted in Figure 11, where each grid depicts the results for all combinations of λ_2, λ_3, given some fixed value of λ_1.

Clearly, too aggressive step-sizes lead to non-convergence which is to be expected given our discussion in Section 7. Non-convergence correlates most with respect to splitting rate smoothing, i.e. λ_1, where it becomes very difficult to solve for $\lambda_1 \geq 0.5$, whereas more aggressive choices for λ_2, λ_3 have a comparatively less detrimental effect. For a conservatively chosen λ_1, i.e. $\lambda_1 = 0.1$, we can find a solution irrespective of the values for the other two smoothing variables in this case study. Further, it is also worth noting that even though we cannot guarantee uniqueness analytically, we found the exact same solution across all converging step-size combinations. For this particular worst-case scenario, the best result was found to take 134 iterations $(\lambda_1 = 0.1, \lambda_2 = 0.2, \lambda_3 = 0.3)$ equating to $\approx 63s$ of simulation time[9], see Figure 11(a). The main takeaway here is that the convergence of the algorithm mainly depends on the extent to which splitting rates are altered between iterations. When adopting fixed step-sizes we therefore recommend: (i) a conservative splitting rate step-size $0.1 \leq \lambda_1 \leq 0.25$, (ii) sending flow (storage capacity factor) step-size $0.2 \leq \lambda_2 \leq 0.5$, and (iii) receiving flow (flow capacity factor) step-size $0.2 \leq \lambda_3 \leq 0.3$.

[9] All simulations are conducted on Windows 10 with an Intel Xeon E-2186G CPU 3.80GHz (single core execution). Solution algorithm implemented in Visual Studio C++ (2008 SP1 compiler).

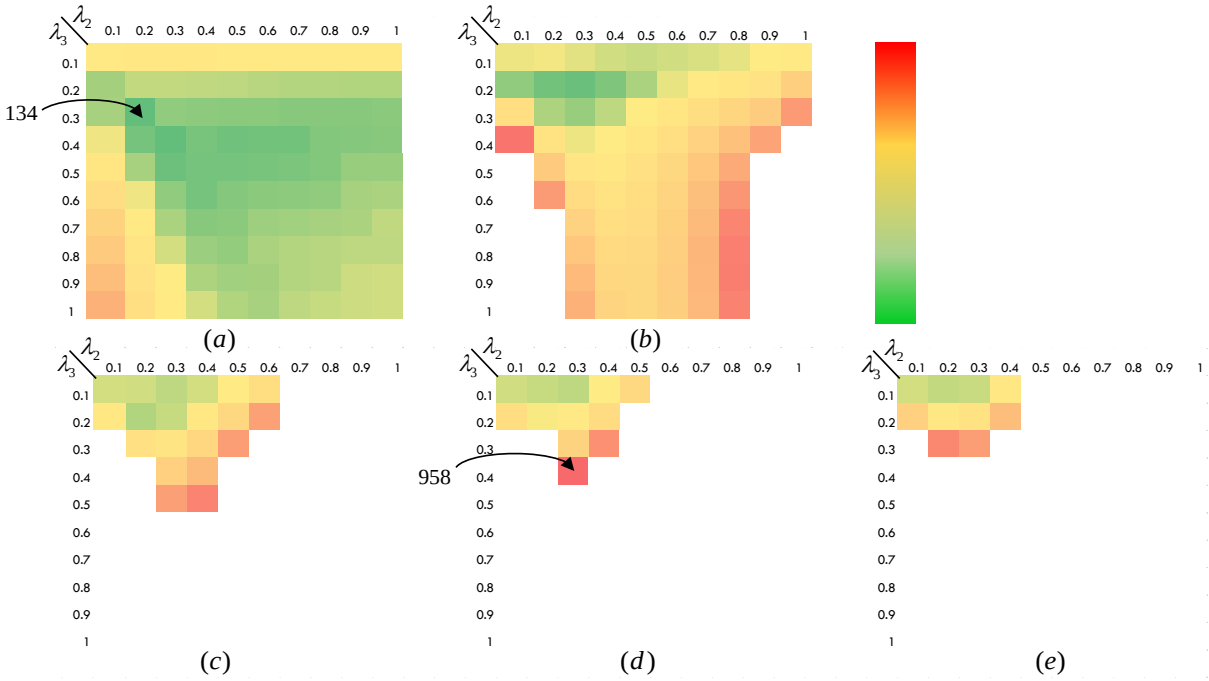


Figure 11: Iterations needed for convergence with $\varepsilon \leq 10^{-6}$ on grid network with various splitting rate step-sizes, (a) $\lambda_1 = 0.1$, (b) $\lambda_1 = 0.25$, (c) $\lambda_1 = 0.5$, (d) $\lambda_1 = 0.75$, and (e) $\lambda_1 = 1$. Red cells indicate a larger number of iterations required to solve while green cells require less (no colour means no solution possible or a solution requiring more than 1000 iterations).

10 Real world Case study

Let us now showcase the suitability of our method on the large-scale real-world network of Gold Coast (Australia) during a fictitious morning peak period (kindly provided by Veitch Lister Consultancy). An impression of the network and its characteristics is provided in Figure 12. We should point out that our goal here is not to equilibrate nor calibrate this model. We merely demonstrate suitability of the solution algorithm – given the adopted link characteristics - provide insight in to what extent the existence of congestion and spillback impacts on the difficulty of solving the network and expose possible limitations of our current implementation.

Infrastructure:
- Links : 11,140 (total length 2,818 km)
- Nodes : 4,675
- Zones : 1,068

Trips:
- Max. AM peak trips : ≈ 139,246
- Paths : 1,104,374 (a-priori generated)

Simulation:
- Duration (T) :1 hour
- FD :Quadratic-Linear
- Route choice : MNL with logit scale parameter of -5

Figure 12: Gold Coast network characteristics.

We consider demand scenarios with severe spillback where we load the maximum demand as well as scenarios with some or no congestion at all by uniformly downscaling the maximum demand in incremental steps. Results are provided in Table 5, and confirm the expectation that when a network moves from largely

uncongested to extremely congested, i.e. more links in spillback state, the forward and backward interactions make it more difficult to find a solution, even with conservative step-sizes.

Table 5: Gold Coast network loading results under various demand levels, where the percentages indicate scaling from the maximum demand considered.

Demand level	0-30%	40%	50%	60%	70%	80%	90%	Max.
Iterations	2	39	79	99	-	-	-	180
Network loading [s]	26	266	595	826	-	-	-	1093

We also see that under certain demand levels we are unable to find a solution. The reason for this is largely due to the coding of the network's infrastructure. This network has over 3,000 very short links < 0.1 km as it frequently models slip lanes and approach lanes separately and explicitly, something uncommon in strategic models, see Figure 12. The effect this has on the algorithm is the following: the difference between a link in uncongested state and the same link in spillback state for these short links - on average over the considered time period - can be as little as a few vehicles causing small changes in flows between subsequent iterations to cause a reversal to the initial link state. This makes it challenging to find a solution. Also, in congested conditions the number of separate queues existing in the network is highest and therefore this phenomenon is most likely to manifest itself here. Fortunately, most strategic transport planning models are coarser, and its links are generally longer. Nevertheless, this is a noteworthy shortcoming of the current method. To confirm this theory, we solved the model again, but now by introducing a configurable relaxing of a link's storage capacity via:

$$r_a = \min\{v_a + \tfrac{1}{T}\tilde{\ell}_a \Phi^{-1}_{II,a}(v_a), q_a^{\max}\}, \quad \tilde{\ell}_a = \min\{\ell_a^{\min}, \ell_a\}, \forall a \in \mathcal{A}. \tag{14}$$

with $\ell_a^{\min} = 0.2$ [km]. The results are provided in Table 6. As can be seen, the introduction of $\ell_a^{\min}$ ensures a minimum amount of storage on very short - and therefore potentially unstable – links. Subsequently, we now find a solution across the demand levels and the same demand levels can be solved more quickly and in less iterations. To give an indication of how congested the resulting network is, we also provide the number of links that are in congestion and/or spillback for each demand level, and a graphical impression of the 60%,80%, and maximum demand levels in Figure 13.

Table 6: Gold Coast network loading results under various demand levels and relaxing storage constraints on links with a length small than $\ell_a^{\min}$, where the percentages indicate scaling from the maximum demand considered.

Demand level	0-30%	40%	50%	60%	70%	80%	90%	Max.
Iterations	2	37	51	74	207	91	91	105
Network loading [s]	27	150	275	466	2907	566	501	567
Congested links	0	6	26	108	350	522	790	1063

Lastly, we point out that for the 70% demand level a significant higher number of iterations is needed, illustrating that the algorithm is still somewhat vulnerable to certain (local) cases of demand-supply interactions that are particularly hard to solve.

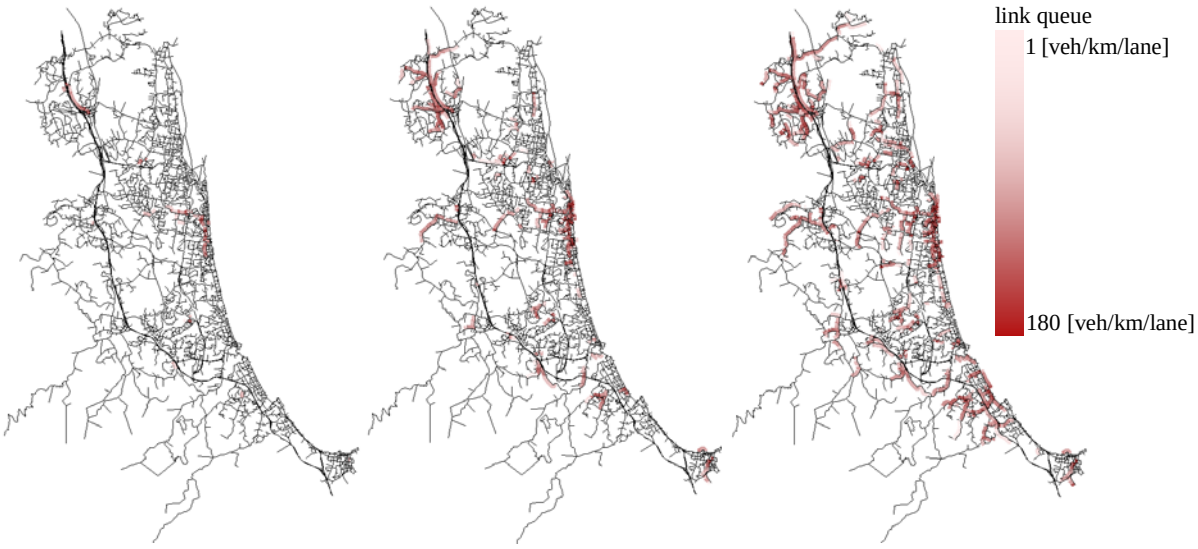


Figure 13:links in congestion or spillback with normalised queue densities for (a) 60% demand level, (b) 80% demand level, (c) maximum demand.

11 Conclusions and future research

In this study we proposed the first solution algorithm capable of solving the path based Static Link Transmission Model which captures both capacity constraints as well as storage constraints, the underlying theory of which originates from Bliemer and Raadsen (2018a). The algorithm revolves around the separate treatment of each of the node model's inputs, namely the splitting rates, receiving flows, and sending flows. Each algorithm component temporarily fixes two of the three inputs to update the remaining non-fixed variable. We argue that at least two extensions to the base algorithm are required to make it suitable for general purpose large-scale networks, either to minimise the risk of local or global non-convergence. Further, we calibrated the algorithm's smoothing parameters, where each of the three different algorithm components adopts its own smoothing: (i) splitting rate smoothing should be chosen conservatively, i.e. $0.1 \leq \lambda_1 \leq 0.25$, while the sending and receiving flow smoothing is recommended to be set as $0.2 \leq \lambda_2 \leq 0.5$, and $0.2 \leq \lambda_3 \leq 0.3$, respectively.

Finally, we conducted a large-scale case study on the Gold Coast network to verify the suitability of our method. We found that the proposed solution method is capable of finding solutions under different demand scenarios and that the difficulty of finding a solution correlates heavily with the amount of congestion/spillback that occurs on the network (as is to be expected). We also identified two limitations of the method in its current form: (i) the smaller the links in the network, the smaller the change in flow required to let a link switch states from complete spillback to free flow (uncongested). Hence, the existence of many such links increases instability because we only consider the situation on each link separately in our method. We found that imposing a (virtual) minimum link length of 0.2 km and above results in sufficient "slack" in storage capacity to avoid global non-convergence in this case study. (ii) The computational cost of finding a static steady-state path-based solution to the network loading model is more in line with a dynamic model than with a traditional static model, or even a capacity constrained point-queue model. The difficulty of incorporating storage constraints with instantaneous propagating flows is the main reason for this.

11.1 Future research

Our first objective is to combine our network loading method with route choice to be able to equilibrate a and perform a complete traffic assignment model. We also hope to address the mentioned method limitations in future work, possibly by moving away from a (single) link-based methodology. We also hope that this work contributes to providing insights in how one might construct solution methods for static assignment models with storage constraints and generate interests in the wider community to develop alternative solution methods for this relatively new type of modelling approach.

Appendix A

Conceptual visualisation of base solution scheme after decomposing the flow acceptance factor. The main benefit of this decomposition besides additional insight in the traffic state is found in the fact that the direct connection between inflow/outflow and flow acceptance factor is removed. This allows for selective smoothing of the two factors (see Section 7), while not having to smooth the inflows/sending flows or outflows/receiving flows.

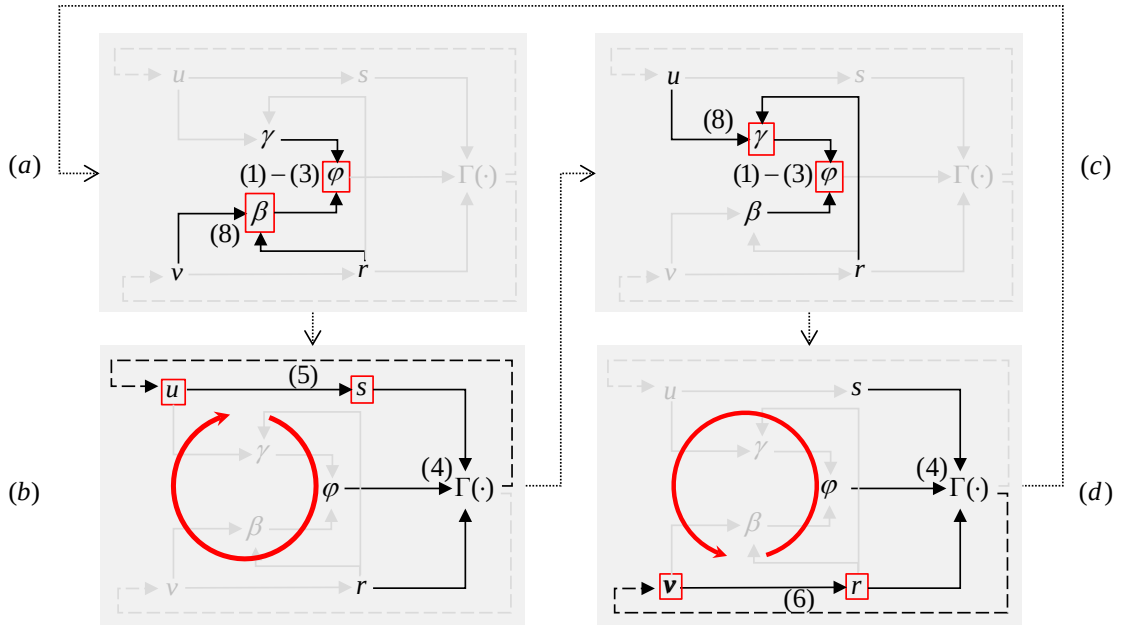


Figure A.1: Updated conceptual solution scheme of solving the static LTM through two interconnected fixed point sub-problems, (a) and (c) connect the two fixed point problems via a path splitting rate update, (b) downstream propagation fixed point sub-problem, (c) upstream propagation fixed point sub-problem. Non-fixed variables highlighted in red (box).

Appendix B

Base solution algorithm:

Initialisation

(a) Set iteration number $i = 1$.

(b) Initial link flow and storage factors, $\beta_a^0 = \gamma_a^0 = 1, \forall a \in \mathcal{A}$.

(c) Initial in/outflows via network loading, $v_a = u_a = \sum_{a \in A} u_{ap}$, given $u_{ap} = \delta_{ap} f_p$.

(d) Initial sending and receiving flows, $s_a = u_a, r_a = \min\left\{q_a^{\max}, v_a + \frac{\ell \Phi_{II}^{-1}(v_a)}{T}\right\}, \forall a \in A$.

(e) Continue to 1(b).

Step 1 - splitting rate update

(a) Update inflows via network loading, $u_{ap} = \delta_{ap} f_p \prod_{a' \in \mathcal{A}_{pa}} \alpha_{a'}$, with $\alpha_{a'} = \frac{\beta_{a'}^{i-1}}{\gamma_{a'}^{i-1}}$, $\forall a \in \mathcal{A}, \forall p \in \mathcal{P}$

(b) *Update splitting rates*, $\varphi_{ab} = \frac{1}{u_a} \sum_{p \in \mathcal{P}} \delta_{bp} u_{ap}$, with $u_a = \sum_{p \in \mathcal{P}} u_{ap}, \forall a, b \in \mathcal{A}$.

(c) Continue to 2(b).

Step 2 - sending flow update – lock splitting rates and receiving flows

(a) Update link inflows via node model, $(\mathbf{u}_n, \mathbf{v}_n) = \Gamma_n(\mathbf{s}_n, \mathbf{r}_n, \boldsymbol{\varphi}_n), \forall b \in \mathcal{A}_n^+, \forall n \in \mathcal{N}$.

(b) *Update sending flows*, $\tilde{s}_a = \min\left\{q_a^{\max}, u_a\right\}, \forall a \in \mathcal{A}$.

(c) Verify local convergence, $H = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \left|\tilde{s}_a - s_a\right|$ and set $s_a = \tilde{s}_a$.

(d) If not converged, i.e. $H > \varepsilon$ return to 2(a).

(e) *Update gammas*, $\gamma_a^i = \frac{u_a}{r_a}$, continue to 3(a).

Step 3 - splitting rate update

(a) Update inflows via network loading, $u_{ap} = \delta_{ap} f_p \prod_{a' \in \mathcal{A}_{pa}} \alpha_{a'}$, with $\alpha_{a'} = \frac{\beta_{a'}^{i-1}}{\gamma_{a'}^{i}}$, $\forall a \in \mathcal{A}, \forall p \in \mathcal{P}$

(b) *Update splitting rates*, $\varphi_{ab} = \frac{1}{u_a} \sum_{p \in \mathcal{P}} \delta_{bp} u_{ap}$, with $u_a = \sum_{p \in \mathcal{P}} u_{ap}, \forall a, b \in \mathcal{A}$.

(c) Continue to 4(a).

Step 4 - receiving flow update – lock splitting rates and sending flows

(a) Update link outflows via node model, $(\mathbf{u}_n, \mathbf{v}_n) = \Gamma_n(\mathbf{s}_n, \mathbf{r}_n, \boldsymbol{\varphi}_n), \forall b \in \mathcal{A}_n^+, \forall n \in \mathcal{N}$.

(b) *Update receiving flows*, $r_a = \left\{q_a^{\max}, v_a + \frac{1}{T} \ell \Phi_{II}^{-1}(v_a)\right\}, \forall a \in \mathcal{A}$.

(c) Verify local convergence, $H = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \left|\tilde{r}_a - r_a\right|$ and set $r_a = \tilde{r}_a, \forall a \in \mathcal{A}$.

(d) If not converged, i.e. $H > \varepsilon$, return to 4(a)

(e) *Update betas*, $\beta_a^i = \frac{v_a}{r_a}, \forall a \in \mathcal{A}$. , continue to overall convergence check

Verify overall convergence

(a) Verify overall convergence, $H = \frac{1}{|\tilde{\mathcal{A}}|} \sum_{a \in \tilde{\mathcal{A}}} \left|\frac{\beta_a^i}{\gamma_a^i} - \frac{\beta_a^{i-1}}{\gamma_a^{i-1}}\right|$, $\tilde{\mathcal{A}} = \left\{a \in \mathcal{A} \middle| \min\left\{\frac{\beta_a^i}{\gamma_a^i}, \frac{\beta_a^{i-1}}{\gamma_a^{i-1}}\right\} < 1\right\}$.

(b) Increment iteration index $i := i + 1$.

(c) If not converged return to 1(a), else **done**, i.e. $H < \varepsilon$.

Appendix C

Base solution algorithm and extension 1; smoothing of internal-step results

Initialisation

(a) Set iteration number $i = 1$.

(b) Initial link flow and storage factors, $\beta_a^0 = \gamma_a^0 = 1, \forall a \in \mathcal{A}$.

(c) Initial in/outflows via network loading, $v_a = u_a = \sum_{a \in A} u_{ap}$, given $u_{ap} = \delta_{ap} f_p$.

(d) Initial sending and receiving flows, $s_a = u_a, r_a = \min\left\{q_a^{\max}, v_a + \frac{1}{T}\ell_a \Phi_{II}^{-1}(v_a)\right\}, \forall a \in \mathcal{A}$.

(e) Continue to 1(b).

Step 1 - splitting rate update

(a) Update inflows via network loading, $u_{ap} = \delta_{ap} f_p \prod_{a' \in \mathcal{A}_{pa}} \alpha_{a'}$, with $\alpha_{a'} = \frac{\beta_{a'}^{i-1}}{\gamma_{a'}^{i-1}}$, $\forall a \in \mathcal{A}, \forall p \in \mathcal{P}$

(b) *Determine splitting rates*, $\varphi_{ab} = \frac{1}{u_a} \sum_{p \in \mathcal{P}} \delta_{bp} u_{ap}$, with $u_a = \sum_{p \in \mathcal{P}} u_{ap}, \forall a, b \in \mathcal{A}$.

(c) *Update smoothed splitting rates*, $\varphi_{ab}^i := (1-\lambda_1)\varphi_{ab}^{i-1} + \lambda_1 \varphi_{ab}, \forall a, b \in \mathcal{A}$ $i > 1$, continue to 2(b).

Step 2 - sending flow update – lock splitting rates and receiving flows

(a) Update link inflows via node model, $(\mathbf{u}_n, \mathbf{v}_n) = \Gamma_n(\mathbf{s}_n, \mathbf{r}_n, \boldsymbol{\varphi}_n^i), \forall b \in \mathcal{A}_n^+, \forall n \in \mathcal{N}$.

(b) *Update sending flows*, $\tilde{s}_a = \min\left\{q_a^{\max}, u_a\right\}, \forall a \in \mathcal{A}$.

(c) Verify local convergence, $H = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \left|\tilde{s}_a - s_a\right|$ and set $s_a = \tilde{s}_a$.

(d) If not converged, i.e. $H > \varepsilon$ return to 2(a).

(e) *Update smoothed gammas*, $\gamma_a^i := (1-\lambda_2)\gamma_a^{i-1} + \lambda_2 \frac{u_a}{r_a}, \forall a \in \mathcal{A}$, continue to 3(a).

Step 3 - splitting rate update

(a) Update inflows via network loading, $u_{ap} = \delta_{ap} f_p \prod_{a' \in \mathcal{A}_{pa}} \alpha_{a'}$, with $\alpha_a = \frac{\beta_a^{i-1}}{\gamma_a^i}$, $\forall a \in \mathcal{A}, \forall p \in \mathcal{P}$

(b) *Update splitting rates*, $\varphi_{ab} = \frac{1}{u_a} \sum_{p \in \mathcal{P}} \delta_{bp} u_{ap}$, with $u_a = \sum_{p \in \mathcal{P}} u_{ap}, \forall a, b \in \mathcal{A}$.

(c) *Update smoothed splitting rates*, $\varphi_{ab}^i := (1-\lambda_1)\varphi_{ab}^{i-1} + \lambda_1 \varphi_{ab}, \forall a, b \in \mathcal{A}$, continue to 4(a).

Step 4 - receiving flow update – lock splitting rates and sending flows

(f) Update link outflows via node model, $(\mathbf{u}_n, \mathbf{v}_n,) = \Gamma_n(\mathbf{s}_n, \mathbf{r}_n, \boldsymbol{\varphi}_n^i), \forall b \in \mathcal{A}_n^+, \forall n \in \mathcal{N}$.

(g) *Update receiving flows*, $r_a = \left\{q_a^{\max}, v_a + \frac{1}{T}\ell \Phi_{II}^{-1}(v_a)\right\}, \forall a \in \mathcal{A}$.

(h) Verify local convergence, $H = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \left|\tilde{r}_a - r_a\right|$ and set $r_a = \tilde{r}_a, \forall a \in \mathcal{A}$.

(i) If not converged, i.e. $H > \varepsilon$, return to 4(a)

(j) *Update smoothed betas*, $\beta_a^i := (1-\lambda_3)\beta_a^{i-1} + \lambda_3 \frac{v_a}{r_a}, \forall a \in \mathcal{A}$. , continue to overall convergence check

Verify overall convergence

(a) Verify overall convergence, $H = \frac{1}{|\tilde{\mathcal{A}}|} \sum_{a \in \tilde{\mathcal{A}}} \left|\frac{\beta_a^i}{\gamma_a^i} - \frac{\beta_a^{i-1}}{\gamma_a^{i-1}}\right|$, $\tilde{\mathcal{A}} = \left\{a \in \mathcal{A} \middle| \min\left\{\frac{\beta_a^i}{\gamma_a^i}, \frac{\beta_a^{i-1}}{\gamma_a^{i-1}}\right\} < 1\right\}$.

(b) Increment iteration index $i := i + 1$.

(c) If not converged return to 1(a), else **done**, i.e. $H < \varepsilon$.

Appendix D

Final solution scheme including the two extensions

Initialisation

(a) Set iteration number $i = 1$.

(b) Initial link flow and storage factors, $\beta_a^0 = \gamma_a^0 = 1, \forall a \in \mathcal{A}$.

(c) Initial in/outflows via network loading, $v_a = u_a = \sum_{a \in A} u_{ap}$, given $u_{ap} = \delta_{ap} f_p$.

(d) Initial sending and receiving flows, $s_a = u_a, r_a = \min\left\{q_a^{\max}, v_a + \frac{\ell \Phi_{II}^{-1}(v_a)}{T}\right\}, \forall a \in A$.

(e) Continue to 1(b).

Step 1 - splitting rate update

(a) Update inflows via network loading, $u_{ap} = \delta_{ap} f_p \prod_{a' \in \mathcal{A}_{pa}} \alpha_{a'}$, with $\alpha_{a'} = \frac{\beta_{a'}^{i-1}}{\gamma_{a'}^{i-1}}$, $\forall a \in \mathcal{A}, \forall p \in \mathcal{P}$

(b) *Determine splitting rates*, $\varphi_{ab} = \frac{1}{u_a} \sum_{p \in \mathcal{P}} \delta_{bp} u_{ap}$, with $u_a = \sum_{p \in \mathcal{P}} u_{ap}, \forall a, b \in \mathcal{A}$.

(c) *Update smoothed splitting rates*, $\varphi_{ab}^i := (1 - \lambda_1)\varphi_{ab}^{i-1} + \lambda_1 \varphi_{ab}, \forall a, b \in \mathcal{A}$ $i > 1$, continue to 2(b).

Step 2 - sending flow update – lock splitting rates and receiving flows

(a) Update link inflows via node model, $(\mathbf{u}_n, \mathbf{v}_n) = \Gamma_n(\mathbf{s}_n, \mathbf{r}_n, \boldsymbol{\varphi}_n^i), \forall b \in \mathcal{A}_n^+, \forall n \in \mathcal{N}$.

(b) *Update sending flows*, $\tilde{s}_a = \min\left\{q_a^{\max}, u_a\right\}, \forall a \in \mathcal{A}$.

(c) Verify local convergence, $H = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \left| \tilde{s}_a - s_a \right|$ and set $s_a = \tilde{s}_a$.

(d) If not converged, i.e. $H > \varepsilon$ return to 2(a).

(e) *Update smoothed gammas*, $\gamma_a^i := (1 - \lambda_2)\gamma_a^{i-1} + \lambda_2 \frac{u_a}{r_a}, \forall a \in \mathcal{A}$, continue to 3(a).

Step 3 - splitting rate update and multiplication factor estimate

(a) Update inflows via network loading, $u_{ap} = \delta_{ap} f_p \prod_{a' \in \mathcal{A}_{pa}} \alpha_{a'}$, with $\alpha_a = \frac{\beta_a^{i-1}}{\gamma_a^i}$, $\forall a \in \mathcal{A}, \forall p \in \mathcal{P}$

(b) *Update splitting rates*, $\varphi_{ab} = \frac{1}{u_a} \sum_{p \in \mathcal{P}} \delta_{bp} u_{ap}$, with $u_a = \sum_{p \in \mathcal{P}} u_{ap}, \forall a, b \in \mathcal{A}$.

(c) *Update smoothed splitting rates*, $\varphi_{ab}^i := (1 - \lambda_1)\varphi_{ab}^{i-1} + \lambda_1 \varphi_{ab}, \forall a, b \in \mathcal{A}$.

(d) Estimate multiplication factor, $\mu_b = \sum_{a \in \mathcal{A}_n^-} \frac{\tilde{\varphi}_{ab}}{\varphi_{ab}}$, with $\tilde{\varphi}_{ab} = \frac{v_a \varphi_{ab}^i}{u_b}; v_a = u_a \alpha_a$, $\forall b \in \mathcal{A}$, continue to 4(a).

Step 4 - receiving flow update – lock splitting rates and sending flows

(a) Update link outflows via node model, $(\mathbf{u}_n, \mathbf{v}_n,) = \Gamma_n(\mathbf{s}_n, \mathbf{r}_n, \boldsymbol{\varphi}_n^i), \forall b \in \mathcal{A}_n^+, \forall n \in \mathcal{N}$.

(b) *Update receiving flows*, $\tilde{r}_a = u_a - \frac{1}{\mu_a} \Delta r_a$, with $\Delta r_a = u_a - \min\left\{q_a^{\max}, v_a + \frac{\ell \Phi_{II}^{-1}(v_a)}{T}\right\}, \forall a \in \mathcal{A}$.

(c) Verify local convergence, $H = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \left| \tilde{r}_a - r_a \right|$ and set $r_a = \tilde{r}_a, \forall a \in \mathcal{A}$.

(d) If not converged, i.e. $H > \varepsilon$, return to 4(a)

(e) *Update smoothed betas*, $\beta_a^i := (1 - \lambda_3)\beta_a^{i-1} + \lambda_3 \frac{v_a}{r_a}, \forall a \in \mathcal{A}$. , continue to overall convergence check

Verify overall convergence

(d) Verify overall convergence, $H = \frac{1}{|\tilde{\mathcal{A}}|} \sum_{a \in \tilde{\mathcal{A}}} \left| \frac{\beta_a^i}{\gamma_a^i} - \frac{\beta_a^{i-1}}{\gamma_a^{i-1}} \right|$, $\tilde{\mathcal{A}} = \left\{ a \in \mathcal{A} \middle| \min\left\{ \frac{\beta_a^i}{\gamma_a^i}, \frac{\beta_a^{i-1}}{\gamma_a^{i-1}} \right\} < 1 \right\}$.

(e) Increment iteration index $i := i + 1$.

(f) If not converged return to 1(a), else **done**, i.e. $H < \varepsilon$.

References

Bakker, D., P. Mijjer, F. Hofman, 1994. QBLOK: een toedelingsmethodiek voor het modelleren van de afhankelijkheid tussen knelpunten en de voorspelling van blokkades. In Proceedings of CVS Congres, Rotterdam (pp. 313-332)

Beckmann, M., McGuire, C. B., Winsten, B. W., 1956. *Studies in the economics of transportation*. New Haven CT, USA: Yale University Press.

Bifulco, G., Crisalli, U., 1998. Stochastic user equilibrium and link capacity constraints: formulation and theoretical evidencies. In *Proceedings of the European Transport Conference.* (pp. 85–96).

Bliemer, M. C. J., Raadsen, M. P. H., Smits, E.-S., Zhou, B., Bell, M. G. H., 2014. Quasi-dynamic traffic assignment with residual point queues incorporating a first order node model. *Transportation Research Part B: Methodological*, *68*, 363–384. https://doi.org/10.1016/j.trb.2014.07.001

Bliemer, M.C.J., Raadsen M.P.H., 2018a. Static traffic assignment with residual queues and spillback. Submitted to ISTTT23 to be held in Lausanne Switzerland.

Bliemer, M. C. J., Raadsen, M. P. H., 2018b. Continuous-time general link transmission model with simplified fanning, part I: Theory and link model formulation. *Transportation Research Part B: Methodological*, *0*, 1–29. https://doi.org/10.1016/j.trb.2018.01.001

Brederode, L., Pel, A., Wismans, L., de Romph, E., Hoogendoorn, S., 2018. Static Traffic Assignment with Queuing: model properties and applications. *Transportmetrica A: Transport Science*, *0*(0), 1–36. https://doi.org/10.1080/23249935.2018.1453561

Bundschuh, M., Vortisch, P., van Vuuren, T., Mott McDonald., 2006. Modelling queues in static traffic assignment. *European Transport Conference Proceedings*.

Courant, R., Friedrichs, K., Lewy, H., 1928. Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*. https://doi.org/10.1007/BF01448839.

Daganzo, C. F. (1995). The cell transmission model, part II: Network traffic. *Transportation Research Part B: Methodological*, *29*(2), 79–93. https://doi.org/10.1016/0191-2615(94)00022-R

Davidson, P., Thomas, A., Teye-Ali, C., 2011. Clocktime assignment: a new mesoscopic junction delay highway assignment approach to continuously assign traffic over the whole day. In *European Transport Conference Proceedings*. Retrieved from http://trid.trb.org/view.aspx?id=1237885.

Frank, M., Wolfe, P., 1956. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, *3*, 95–109. https://doi.org/10.1016/S0377-2217(87)80152-2

Gentile, G., 2010. The General Link Transmission Model for Dynamic Network Loading and a comparison with the DUE algorithm. In *New Developments in Transport Planning: Advances in Dynamic Traffic Assignment (Chapter 8)* (pp. 1615–1620).

Hall, M.D., Vliet van D., Willumsen, L.G., 1980. Saturn - a Simulation-Assignment Model for the Evaluation of Traffic Management Schemes. *Traffic Engineering & Control*, *21*(4), 168–170. Retrieved from https://trid.trb.org/view/159897

Hearn, D.W., 1980. Bounding flows in traffic assignment models. In: Research Report 80–4. Gainesville, FL: Department of Industrial and Systems Engineering, University of Florida.

Himpe, W., Corthout, R., Tampère, M. J. C., 2016. An efficient iterative link transmission model. *Transportation Research Part B: Methodological*, *92*, 170–190. https://doi.org/10.1016/j.trb.2015.12.013

Hungerink G.J., 1989. Q-Net: Assignment on Over-Congested Networks by Link Inflow Constraint. In *Proc., U.S.-Italy Joint Seminar on Urban Traffic Networks: Dynamic Control and Flow Equilibrium.* Capri.

Lam, W., Zhang, Y., 2000. Capacity-constrained traffic assignment in networks with residual queues. *Journal of Transportation Engineering*, (April), 121–128. Retrieved from http://ascelibrary.org/doi/abs/10.1061/(ASCE)0733-947X(2000)126:2(121)

Larsson, T., Patriksson, M., 1995. An augmented lagrangean dual algorithm for link capacity side constrained traffic assignment problems. *Transportation Research Part B: Methodological*. https://doi.org/10.1016/0191-2615(95)00016-7.

Newell, G. F. (1993). A simplified theory of kinematic waves in highway traffic, part II: Queueing at freeway bottlenecks. *Transportation Research Part B: Methodological*, *27*(4), 289–303. https://doi.org/10.1016/0191-2615(93)90039-D

Nie, Y., Zhang, H., Lee, D., 2004. Models and algorithms for the traffic assignment problem with link capacity constraints. Transportation Research Part B 38, 285–312.

Raadsen, M. P. H., Bliemer, M. C. J., Bell, M. G. H., 2016. An efficient and exact event-based algorithm for solving simplified first order dynamic network loading problems in continuous time. *Transportation Research Part B: Methodological*, *92*, 191–210. https://doi.org/10.1016/j.trb.2015.08.004

Shahpar, A. H., Aashtiani, H. Z., Babazadeh, A., 2008. Dynamic penalty function method for the side constrained traffic assignment problem. *Applied Mathematics and Computation*, *206*(1), 332–345. https://doi.org/10.1016/j.amc.2008.09.014

Smith, M. J., 2013. A link-based elastic demand equilibrium model with capacity constraints and queueing delays. *Transportation Research Part C: Emerging Technologies*, *29*, 131–147. https://doi.org/10.1016/j.trc.2012.04.011

Smith, M., Huang, W., Viti, F., 2013. Equilibrium in Capacitated Network Models with Queueing Delays, Queue-storage, Blocking Back and Control. *Procedia - Social and Behavioral Sciences*, *80*(Isttt), 860–879. https://doi.org/10.1016/j.sbspro.2013.05.047.

Smulders, S., 1987 Modelling and filtering of freeway traffic flow. Report OS-R8706, Centre of Mathematics and Computer Science, the Netherlands.

Smulders, S., 1990. Control of freeway traffic flow by variable speed signs. Transp. Res. Part B 24 (2), 111–132.

Tajtehranifard, H., Bhaskar, A., Nassir, N., Haque, M. M., Chung, E., 2018. A path marginal cost approximation algorithm for system optimal quasi-dynamic traffic assignment. *Transportation Research Part C: Emerging Technologies*. https://doi.org/10.1016/j.trc.2018.01.002

assignment. Transp. Res. Part C: Emerg. Technol. 88, 91–106. https://doi.org/10.1016/J.TRC.2018.01.002.

Tampère, C.M.J., Corthout, R., Cattrysse, D., Immers, L.H., 2011. A generic class of first order node models for dynamic macroscopic simulation of traffic flows. Transp. Res. Part B Methodol. 45, 289–309. doi:10.1016/j.trb.2010.06.004

van Vliet, D., 1982. SATURN: a modern assignment model. *Traffic Engineering and Control*, *23*, 575–581.