Hyperbush Algorithm for Strategy-based Equilibrium Traffic Assignment Problems

Zhandong Xu\textsuperscript{1,2}, Jun Xie\textsuperscript{*1,2}, Xiaobo Liu\textsuperscript{1,2}, and Yu (Marco) Nie\textsuperscript{3}

\textsuperscript{1}School of Transportation and Logistics, Southwest Jiaotong University, Chengdu, China
\textsuperscript{2}National Engineering Laboratory of Integrated Transportation Big Data Application Technology, Southwest Jiaotong University, Chengdu, Sichuan 610031, China
\textsuperscript{3}Department of Civil and Environmental Engineering, Northwestern University, IL 60208, USA

February 1, 2022

Abstract

Strategy-based equilibrium traffic assignment (SETA) problems define travel choice broadly as a strategy rather than a simple path. Travelers navigating through a network based on a strategy end up following a hyperpath. SETA is well suited to represent a rich set of travel choices that take place en-route at nodes, such as transit passengers’ transfer decision, truckers’ bidding decision and taxi drivers’ re-position decision. This paper recognizes and highlights the commonalities among classical and emerging SETA problems and proposes to unify them within a same modeling framework, built on the concept of hypergraph. A generic hyperbush algorithm (HBA) is developed by decomposing a hypergraph into destination-based hyperbushes. By constructing hyperbushes and limiting traffic assignment to them, HBA promises to obtain more precise solutions to larger instances of SETA problems at a lower computational cost, both in terms of CPU time and memory consumption. To demonstrate its generality and efficiency, we tailor HBA to solve two SETA problems. The results confirm HBA consistently outperforms the benchmark algorithms in the literature, including two state-of-the-art hyperpath-based algorithms. To obtain high-quality equilibrium solutions for SETA instances of practical size, HBA runs up to five times faster than the best competitor with a fraction of its memory consumption.

Keywords: strategy-based equilibrium; traffic assignment; hyperpath; hyperbush

\textsuperscript{*}Corresponding author, E-mail: jun.xie@swjtu.edu.cn.
1 Introduction

The traffic assignment problem (TAP) aims to establish a spatial equilibrium between the demand for moving between pairs of locations in a network and the supply capacity available to serve such demand. It is a fundamental computational tool that has seen numerous applications, of which the most well-known is probably urban travel forecasting (Boyce and Williams 2015). In a generic setting, the equilibrium state in TAP is formed through the interaction between the travel choice on the demand side and the physical response on the supply side. Many variants of TAP can be defined based on how the travel choice and the physical response are specified. The standard user equilibrium TAP (Beckmann et al. 1956; Sheffi 1985, referred to as UETAP hereafter), for example, only allows travelers to choose paths based on travel cost, and model the travel cost on each link in the network as an increasing function of its own flow. In this paper, we focus on a class of strategy-based equilibrium traffic assignment (SETA) problems, in which the travel choice is broadly defined as a strategy rather than a simple path. To accommodate as many applications as possible, much flexibility is built on the supply side to allow: (i) additive link travel costs; (ii) delay/profits/penalty associated with nodes; and (iii) link weight/choice probability derived from strategy.

A classical SETA application in transportation concerns transit assignment (Spiess and Florian 1989). Due to the many options that passengers may face as they navigate through a multi-modal transit network, the travel choice is much better described as a strategy adaptive to varying conditions than a preset turn-by-turn path. This may be best explained using the common-line problem (Chriqui and Robillard 1975). In such a problem, passengers at a stop served by several bus lines must find an optimal strategy that dictates whether they should take the next arriving bus. Because the inter-arrival time between buses (i.e., headway) tends to be random, the optimal strategy is usually not to stick with a favorite line. Rather, to minimize the expected travel time, the passenger should choose a subset of the available lines, known as the attractive set, and board the first arriving bus from the set. Such a strategy implies the passenger has a nonzero probability of boarding any of the lines in the set. Adopting such a strategy in a network can be viewed as following a hyperpath (Nguyen and Pallottino 1988).

SETA problems are also closely linked to Markov Decision Problems (MDP). A recent example arises in the context of the Uber model for freight transportation (Miller and Nie 2020; Miller et al. 2020). In this application, individual truckers tour around a network of many cities, competing for loads advertised on an online freight exchange (OFEX) platform. The key issue here is to model a trucker’s decision at a city, including which loads to bid, at which price and in what order. Like passengers at a transit stop, truckers also face uncertainty: due to market conditions and the competition from fellow truckers, they must hedge against the chance to win a load at a given price. So their strategy, too, materializes as a hyperpath in the network. Another MDP routing game that could be potentially cast as a SETA problem deals with the decision of taxi drivers working for transportation network companies such as Uber and Lyft (Li et al. 2019; Calderone and Sastry 2017). Here, drivers cruise in a network to search for passengers, and their travel choice involves whether, where and how to re-position at a given moment. Similarly, this choice—driven by such uncertain factors as surge price, search time, and the probability of finding a ride—may also be viewed as a strategy.

In this paper, we recognize and highlight the commonalities among the classical and emerging problems mentioned above, and propose to unify them with a SETA modeling framework,
built on the concept of hypergraph. Unlike a regular graph, a hypergraph consists of nodes and hyperlinks. The latter generally represents a node-based decision that involves potential use of multiple downstream links. A hyperpath, then, is a collection of connected hyperlinks. A hyperpath may consist of multiple simple paths and each corresponds to a “use probability”. A SETA problem aims to assign the demand to the hyperpaths connecting each origin-destination (O-D) pair such that no one has the incentive to change travel choice unilaterally. At such a user equilibrium (UE) state, any used hyperpath has an identical and minimum cost (or maximum profit). While the characterization of UE in SETA is the same as in UETAP, we note that SETA—equipped with the structure embedded in hypergraph—can model a much richer set of travel choices, especially those that take place en-route at nodes.

Solving SETA problems is a considerable challenge for several reasons. First and foremost, as the number of hyperpaths in a network is generally much larger than that of simple paths, SETA problems are computationally more demanding to solve than UETAP of similar size. Second, finding an optimal hyperpath in a hypergraph, an indispensable building block of SETA, could be rendered a difficult problem on its own right by the complexities embedded in the travel choices. Last but not least, the supply-side flexibility introduced in SETA problems means the hyperpath cost functions often lack desirable analytical properties that are instrumental to the design of efficient algorithms. Consequently, developing a high-performance assignment algorithm is rarely a main focus in the SETA studies that scatter in the literature. The focus is usually on model formulation, which is then solved by a known algorithm designed to find an approximate UE solution quickly, such as the method of successive average (MSA). Moreover, these algorithms typically operate in the space of hyperpaths, limiting their appeal in large-scale practical applications.

The main contribution of this study is a general hyperbush algorithm (HBA) for SETA problems. HBA predicates on the idea of decomposing a hypergraph into hyperbushes, each rooted at a destination (or an origin). A hyperbush is an acyclic hypergraph that stores, in a highly compact manner, all hyperpaths destined for a destination. By maintaining hyperbushes and limiting traffic assignment to them, HBA promises to obtain more precise solutions to larger instances of SETA problems at a lower computational cost, both in terms of CPU time and memory consumption.

Conceptually, HBA for SETA problems is similar to the bush-based algorithm for UETAP (Nie, 2010). Specifically, they both bypass the need to store and manipulate paths by exploiting acyclicity of a destination-based (or origin-based) network flow pattern at UE. However, we shall see that some of the results key to the validity of the bush-based algorithms require radical changes to fit the context of hyperbush. Three issues are especially noteworthy. First, while a destination-based network flow pattern at UE is guaranteed to be acyclic for UETAP, such an assurance has never been established for any SETA problems. Second, new strategies are needed to iteratively construct an optimal hyperbush, which entails identifying hyperlinks that would both maintain acyclicity and help move the current solution towards UE. Finally, bringing a hyperbush into equilibrium is no longer a task as simple as shifting flows between shortest and longest hyperpaths (as in the case for the bush-based algorithms), since any pair of hyperpaths have much more complicated interactions. Much of the algorithm development will be devoted to analyzing and addressing these issues.

To demonstrate its generality, we tailor the proposed modeling framework to several SETA problems. These include the transit equilibrium assignment problem (TEAP, Spiess and Florian).
the capacitated traffic assignment problem (CTAP, Marcotte and Nguyen, 1998) and the on-line freight exchange assignment problem (OFEX, Miller and Nie, 2020; Miller et al., 2020). We also implement two variants of HBA and apply them to solve TEAP and CTAP. Our numerical experiments will confirm HBA consistently outperforms the benchmark algorithms in the literature, including two state-of-the-art hyperpath-based algorithms. For SETA instances of practical size, HBA runs up to five times faster than the best competitor in obtaining high quality UE solutions, at a considerably lower memory consumption.

The remainder of the paper is structured as follows. Section 2 briefly reviews the related studies. Section 3 gives the formal definitions and problem statement for a hyperpath-based SETA problem, and presents and discusses three applications. This is followed by a destination-based reformulation of SETA in Section 4. The general structure of the hyperbush algorithm is detailed in Section 5, followed by the implementation details for TEAP and CTAP in Section 6. Section 7 reports the results of numerical experiments. A summary of findings and further research directions are given in Section 8.

2 Related Studies

This section aims to cover the formulations and algorithms for the strategy-based equilibrium traffic assignment (SETA) problems. In Section 2.1, we review the application of the hypergraph/hyperpath concept in transportation network modeling. Section 2.2 focuses on transit equilibrium assignment problems (TEAP), which is by far the most important application of SETA problems in transportation. Finally, Section 2.3 provides an overview of the bush-based algorithms, which has become a state-of-the-art solution technique for the traffic assignment problem (TAP) in the past decade.

2.1 Hypergraphs and hyperpaths

First introduced by Berge (1973) to model and solve discrete mathematical problems, a hypergraph is a generalization of graphs. Defined on directed hypergraphs, a hyperpath may be viewed as a generalization of paths in directed graphs. Typical applications of the hypergraph/hyperpath concept can be found in the context of relational databases (Fagin, 1983), And–Or diagrams (Mello and Sanderson, 1986), and transportation.

To the best of our knowledge, the application of hyperpath in transportation first arises from the common line problem (Chriqui and Robillard, 1975). In this problem, transit passengers are assumed to follow a “strategy”, rather than a simple path, to navigate the network. Specifically, at a stop served by several transit lines, they may consider a subset of the lines as “attractive” and hence end up choosing each with a certain probability. Spiess and Florian (1989) extended the notion of strategy-based routing to a general transit network. It has since established that such a strategy can be represented as a hyperpath (Nguyen and Pallottino, 1988, 1989), leading to hyperpath-based formulations for frequency-based TEAPs that will be discussed in details in Section 2.2. Empirical studies based on transit smart card data (Schmöcker et al., 2013; Cheon et al., 2019) and a web-based survey (Kurauchi et al., 2012) uncovered strategy-based routing behavior in the real world.

Optimal hyperpath routing problems have been studied extensively in the literature. Nielsen et al. (2005) proposed an algorithm for finding K-shortest hyperpaths in a directed hypergraph.
Volpentesta (2008) developed an algorithm that can solve one-to-all and one-to-one shortest hyperpath problems in polynomial time. Lozano and Storchi (2002) studied the shortest viable hyperpath problem in a multimodal network, which explicitly considers the maximum number of transfers between different modes. Several recent studies (Gentile et al., 2005; Chen and Nie, 2015; Oliker and Bekhor, 2018) investigated the impact of on-line information on hyperpath routing in transit networks. Using GPS-based automatic vehicle location (AVL) data, Li et al. (2015) and Xu et al. (2020b) examined the empirical probabilistic distributions of headway in real-world transit services, and compared the results with the standard assumptions adopted in the literature.

Other than the variants of the common-line problem, the hyperpath concept has also been applied to formulate and solve various problems defined on stochastic time-dependent networks (Pretolani, 2000; Miller-Hooks, 2001; Gao, 2005; Boyles, 2009; Bell, 2009), the capacitated traffic network (Marcotte et al., 2004), the stochastic network with recourse (Unnikrishnan and Waller, 2009), the parking search network (Boyles et al., 2015), and more recently on-line freight exchange networks (Miller et al., 2020; Miller and Nie, 2020). The hyperbush algorithm developed for SETA in this paper can be adapted to tackle many of the above problems, due to their shared structure embodied in the hypergraph representation. We shall present a few examples in details in Section 3.

2.2 TEAP: formulations and algorithms

Mathematical formulations for TEAP can be classified as schedule-based (e.g., see Poon et al., 2004; Nuzzolo et al., 2012) and frequency-based (e.g., Nguyen and Pallottino, 1988; Wu et al., 1994; Codina and Rosell, 2017). We focus on the latter because they are more closely related to SETA problems.

Early TEAP models assume the headway of all transit lines are independently and exponentially distributed, and passengers always board the first arriving vehicle that serves a line included in their attractive set (Nguyen and Pallottino, 1988; Spiess, 1993; Wu and Florian, 1993; Wu et al., 1994). The common-line problem dictates that, for a given attractive set, passengers’ waiting time follows an inverse additive law of the nominal line frequency, and their probability of boarding each line is proportional to its nominal frequency. In these models, “congestion” usually takes the form of a crowding cost incurred on links, or of an extra boarding time at stops. However, such a representation is often regarded as inadequate for transit networks. Efforts to address this inadequacy had led to models that attempt to capture the so-called bunching effect and the effect of capacity restriction. The former, built on the bulk queue stop model proposed by Gendreau (1984), assumes the effective frequency of a transit line depends on the passenger flow assigned to it (Bouzaïene-Ayari et al., 2001; Cominetti and Correa, 2001; Cepeda et al., 2006; Codina, 2013; Codina and Rosell, 2017). The latter (Kurauchi et al., 2003) imposes the capacity constraint at stops by modeling the passenger’s probability of failing at first boarding due to the limited vehicle capacity. This idea of “fail-to-board” probability was later extended to consider dynamics (Schmöcker et al., 2008) and seat capacities (Schmöcker et al., 2011). Regardless of congestion representation, however, frequency-based TEAP models are commonly formulated as a variational inequality problem (VIP) in the space of hyperpaths.

While much attention has been paid to improving realism in TEAP formulations while maintaining tractability, less efforts were devoted to developing efficient solution algorithms for them, especially for large-scale applications. Spiess and Florian (1989) employed a Frank-Wolfe (FW)
algorithm to solve their model, which only considers the extra waiting time and is formulated as a nonlinear program. [Wu and Florian (1993) and Wu et al. (1994)] consider a variant of the model from [Spiess and Florian (1989)] that includes the disutility associated with the discomfort as part of the congestion cost. They developed two solution algorithms: a simplicial decomposition algorithm operated in the space of quasi link flows, and a projection algorithm operated in the space of hyperpath flows. Recently, [Xu et al. (2020a)] revisited this model, and proposed two Newton-type, hyperpath-based algorithms. Their numerical experiments show these new algorithms outperformed the existing ones by a large margin on large-scale problems. For those TEAP models that feature more realistic congestion effects (e.g., Cepeda et al., 2006; Codina and Rosell, 2017; Schmöcker et al., 2008), the primary solution algorithm is the method of successive averages (MSA). While easy to implement and computationally cheap to run, the inability of MSA to achieve a satisfactorily converged solution within a reasonable amount of computation time is well known and has been demonstrated in numerous studies (e.g., Sheffi, 1985; Mounce and Carey, 2015). Outside academia, FW and MSA are still the choices of the mainstream vendors of commercial transportation planning software, such as INRO Inc. (Spiess, 1993; INRO, 2018), for solving TEAP models.

In a nutshell, solving TEAP models on large-scale networks poses a computational challenge that has yet to be fully addressed. The Newton-type algorithms (Xu et al., 2020a)—recently developed for solving simpler variants of the problem—show promising convergence performance, but their requirement of storing and manipulating hyperpaths remains a formidable obstacle to practical applications. The new algorithm proposed in this paper bypasses this requirement using hyperbushes.

2.3 Bush-based algorithms for TAP

Standard TAP solvers for over a quarter century are typically operate in the space of link flows (e.g., Frank and Wolfe, 1956) or path flows (e.g., Jayakrishnan et al., 1994; Xie et al., 2018; Li et al., 2020). The concept of bush was originally coined in Dial’s seminal work on logit assignment (Dial, 1971). Dial showed that assigning origin-destination (O-D) flows to multiple paths based on the logit model can be carried out very efficiently in an acyclic network that he calls a bush. By exploiting the bush structure, Dial’s STOCH algorithm obviates path enumeration in logit assignment, a remarkable achievement critical to practice in 1970s. Applying the bush concept to solve the user equilibrium TAP, the origin-based algorithm of Bar-Gera (2002) was widely seen as a breakthrough owing to its ability to achieve highly precise solutions. Since then, the development of bush-based TAP algorithms has attracted continual interest from both the academia (e.g., Dial, 2006; Nie, 2010; Bar-Gera, 2010; Nie, 2012; Boyles, 2012; Xie et al., 2013; Gentile, 2014; Xie and Xie, 2016) and the software vendors such as PTV (VISUM) and Caliper (TransCAD).

While the popularity of bush-based algorithms has risen markedly in the past decade, their application outside the TAP literature remains scarce. A notable exception is Nguyen et al. (1998), who extended Dial’s idea to transit assignment. In a similar vein, they show that, with a bush structure, one can avoid path enumeration when performing a logit assignment in a transit network. The focus of Nguyen et al. (1998) is on developing a variant of STOCH that would work for the hyperpath-based assignment, which hinges on computing splitting probabilities for hyperlinks following appropriate topological orders. No attempt has been made, to the best of our knowledge, to develop an algorithm for TEAPs (or more generally SETA problems) using the bush concept. A critical gap is how to iteratively construct an equilibrium hyperbush for each
origin (or destination) that eventually encapsulates all hyperpaths used at equilibrium. This is not a straightforward extension from existing bush-based TAP algorithms, because the optimality conditions used to drive this process are very different in TEAP. We now set out to fill this gap.

3 SETA problems

3.1 Hypergraph and hyperpath

While we focus on the strategy-based equilibrium traffic assignment (SETA) problem defined over a hypergraph, it is useful to begin with the classical traffic assignment problem (TAP). Consider a network represented as a directed graph $G(N, A)$, where $N$ and $A$ denote the set of nodes and links, respectively. A link is identified by $(i, j)$ with $i$ and $j$ being the tail and head node respectively. We use $I(i)$ to denote the set of incoming links associated with node $i$, $I^{-}(i)$ the set of tail nodes corresponding to links in $I(i)$, $O(i)$ the set of outgoing links associated with $i$, and $O^{+}(i)$ the set of head nodes corresponding to links in $O(i)$. Travel demand is represented as the number of trips from each origin $r \in R \subseteq N$ to each destination $d \in D \subseteq N$, denoted as $q = \{q_{rd}\}$. Let $K_{rd}$ denote the set of all simple paths connecting O-D pair $(r, d)$. Each path $k \in K_{rd}$ consists of an ordered set of distinct nodes, i.e., $N_k = \{r = i_1, i_2, \cdots, i_n = d\}$, and a set of distinctive links $A_k = \{(i_1, i_2), \cdots, (i_{n-1}, i_n)\}$. The conventional TAP aims to allocate the demand $q$ onto the paths according to certain behavioral rules, typically producing a stable network flow pattern known as the user equilibrium solution.

We now define a directed hypergraph, and to contrast it with the regular graph, denote it as $G(N, S)$, where $S$ is a set of hyperlinks. A hyperlink $s \in S$ is defined as $(i, J_s)$, where $i = s^{-}$ denotes the tail node of $s$, and $J_s \subseteq O^{+}(i)$ is the head node set. Thus, whereas a link connects a tail node to a head node, a hyperlink connects a tail node to several head nodes [Klein and Manning 2004]. Unless otherwise specified, hereafter we always assume that for any hyperlink with downstream node set $J_s$, a feasible hyperlink corresponding to any (nonempty) subset of $J_s$ must exist. We further define the set of regular links connecting $i$ to each node $j \in J_s$ as $M_s$. Formally, $M_s = \{|s^{-}, j| \forall j \in J_s\} \subseteq A$, and $|M_s|$ is called the degree of the hyperlink $s$. We note that when $|M_s| = 1$ (or $|J_s| = 1$), $s \in S$, the hypergraph $G(N, S)$ is reduced to a standard graph $G(N, A)$. Note that more than one hyperlink could share the same tail node. Accordingly, we use $O_i$ to denote the set of all outgoing hyperlinks associated with node $i$. That is, for any $s \in O_i, s^{-} = i$. The reader is referred to Appendix A for a list of key notations used in this paper.

We are now ready to formally define a hyperpath as follows.

**Definition 1** (hyperpath). A sub-graph of $G(N, S)$ is said to be a hyperpath $G_k = (N_k, S_k, M_k, e_k)$ from node $r \in N_k$ to node $d \in N_k$, where $N_k \subseteq N$ is the set of nodes, $S_k \subseteq S$ is the set of hyperlinks, $M_k = \cup_{s \in S_k} M_s$ is the set of regular links, and $e_k = \{e_{ij, s}, \forall (i, j) \in M_s, s \in S_k, i = s^{-}\}$ is a weight vector, if

(i) $N_k = \cup_{(i,j) \in M_k} \{i, j\}$;

(ii) $|S_k \cap O(d)| = 0$;

(iii) $|S_k \cap O(i)| = 1$, $\forall i \in N_k \setminus \{d\}$;

(iv) there is a simple path $k$ between any node $i \in N_k \setminus \{d\}$ and $d$ such that $A_k \subseteq M_k$, and
(v) $e_k$ satisfies the following conditions

\[ \sum_{(i,j) \in M_k} e_{ij,s} = 1, \quad \forall s \in S_k, \ i = s^- \]  \hspace{1cm} (1a)

\[ e_{ij,s} > 0, \quad \forall s \in S_k, \ (i,j) \in M_s. \]  \hspace{1cm} (1b)

To further explain the above concepts, consider the example shown in Figure 1. Here, the tail of hyperlink $s_1$ (cf. Figure 1(a)) is node $i$, and its head set is $J_{s_1} = \{j_1, j_2, \ldots, j_n\}$. Thus, in this case, $|M_{s_1}| = n - 1$ and $s_1 \in O_i$. The hyperlink $s_2$ shown in Figure 1(b), on the other hand, can be simply viewed as a regular link since $J_{s_2} = \{j\}$. Figure 1(c) illustrates a hypergraph that consists of three hyperpaths $k_1, k_2$ and $k_3$ from origin node 1 to destination node 4. In this case, node 1 contains multiple outgoing hyperlinks, i.e. $O_1 = \{s_3, s_4, s_5\}$. For hyperpath $k_3$, we have $M_k = \{(1,2), (1,3), (2,4), (3,4)\}$. Once the weight vector $e_{k_3} = \{e_{12, s_3}, e_{13, s_3}, e_{24, s_6}, e_{34, s_7}\}$ is given, it is easy to compute the traverse probability on the hyperpath.

![Figure 1: Illustration of hyperlinks and hyperpaths. (a) Hyperlink $s_1$ such that $|M_{s_1}| > 1$; (b) hyperlink $s_2$ such that $|M_{s_2}| = 1$; (c) Three hyperpaths such that $S_{k_1} = \{s_4, s_6\}, S_{k_2} = \{s_5, s_7\}$ and $S_{k_3} = \{s_3, s_6, s_7\}$.](image)

We note that a hyperpath $k$ is reduced to a simple path when $|J_s| = 1, \forall s \in S_k$ and in such case we have $e_{ij,s} = 1, \forall s \in S$. In general, however, a hyperpath $k$ may consist of a set of paths staring from $r$ and ending at $d$, denoted as $H_k$ and each $k \in H_k$ is associated with a weight

\[ \eta_k = \prod_{(i,j) \in A_k, s \in S_k} e_{ij,s}, \quad \forall k \in H_k. \]  \hspace{1cm} (2)

Per the above definition, $\sum_{k \in H_k} \eta_k = 1$. Let $t_{ij} (> 0)$ be the cost associated with traversing a link $(i,j)$ and $o_{ij,s} (\geq 0)$ be the cost of passing through node $i = s^-$. Note that here the node cost is dependent on hyperlink $s$ but the link cost is not. The cost of using a hyperpath $k$ is calculated as a weighted average of the costs on its paths, i.e.

\[ C_{k}^{rd} = \sum_{k \in H_k} \eta_k \left( \sum_{(i,j) \in A_k} t_{ij} + \sum_{i \in N_k \setminus \{d\}, s \in S_k} o_{i,s} \right). \]  \hspace{1cm} (3)

Each O-D pair $(r,d)$ is connected by a set of hyperpaths $K_{rd}$. Let $h_{k}^{rd}$ be the flow from O-D pair $(r,d)$ assigned to hyperpath $k$. The flow conservation condition requires

\[ \sum_{k \in K_{rd}} h_{k}^{rd} = q_{rd}, \quad \forall r \in R, \ d \in D. \]  \hspace{1cm} (4)
A vector of hyperpath flow \( h = \{ h_{rd}^d \} \) is said to be feasible if it belongs to the set

\[
\Omega_h = \{ h | \Lambda h = q, h \geq 0 \},
\]

(5)

where \( \Lambda \) is the incidence matrix that links hyperpaths to O-D pairs.

As the demand \( q \) is loaded onto the hyperpaths, the weight vector \( e = \{ e_{ij, s} \forall s \in S_k, (i, j) \in M_s \} \) may or may not vary with the flow passing through the corresponding network components, depending on the application. To be consistent, we shall treat each as a general function of the hyperpath flow vector \( h \), written as \( t(h), e(h) \) and \( \varpi(h) \). In the next section, we shall state SETA problems and present a formulation without specifying these functions. Section 3.3 demonstrates how these functions can be specified to tailor the need of a wide variety of applications.

It is worth emphasizing that a hyperpath may or may not contain direct cycles (Ausiello et al., 1998). In fact, cycles may be even necessary to ensure optimality in certain applications. Chen and Nie (2015) provide such an example in the context of optimal transit routing with partial online information. Appendix B constructs a simple example, in which the minimum cost hyperpath contains an infinite number of cycles. However, the assumption of acyclicity is almost universal in the transportation literature involving hyperpath (Nguyen and Pallottino 1988; Marcotte et al. 2004; Hamdouch and Lawphongpanich 2008; Bell 2009; Miller et al. 2020). In part, this tradition is due to the fact that most hyperpath applications, especially those arising from transportation, are not prone to the impact of cycles. When we introduce various applications that motivate this study in the next section, we shall see how acyclicity is guaranteed through various mechanisms. Suffice it to say for now that this study will also focus on acyclic hyperpaths, and we define it formally as follows.

**Definition 2 (Acyclic hyperpath).** A hyperpath \( G_k = (N_k, S_k, M_k, e_k) \) from node \( r \in N \) to node \( d \in N \) is acyclic if

(i) it contains no directed simple path that connects a node \( i \in N_k \) to itself; and

(ii) \( \emptyset \)s \( s \in S_k \) such that \( r \in J_s \).

Hereafter, a hyperpath is always considered acyclic unless otherwise specified. With acyclicity come unique computational advantages. For example, evaluating the hyperpath cost using Eq. (3) either requires path enumeration—which is computationally prohibitive on large networks—or solving a linear system of equations (see Boyles and Rambha, 2016). Yet, by exploiting the acyclicity—which implies a topological order—path enumeration can be obviated. Specifically, following an ascending topology order, node traverse probability \( \vartheta^k_i \) and link traverse probability \( \varpi^k_{ij} \) can be evaluated as follows:

\[
\vartheta^k_i = 1, \quad (6a)
\]

\[
\varpi^k_{ij} = \vartheta^k_i e_{ij,s}, \quad \forall (i, j) \in M_k, \; i = s^-, \; s \in S_k, \quad (6b)
\]

\[
\vartheta^k_j = \sum_{(i,j) \in I(j) \cap M_k} \varpi^k_{ij}, \quad \forall j \in N_k \setminus \{r\}. \quad (6c)
\]

With the above definitions, we have

\[
C^d_{rd} = \sum_{(i,j) \in A_k} \pi^k_{ij} t_{ij} + \sum_{i \in N_k \setminus \{d\}, s \in S_k} \vartheta^k_i \omega_{is}. \quad (7)
\]
It is worth noting there is a unique sub-hyperpath from $i$ to $d$ on hyperpath $k$, and we denote $C_{id}^k$ as the cost of the sub-hyperpath. An alternative method to compute the hyperpath cost is to use the recursive formula as

$$C_{id}^k = \begin{cases} 0, & i = d, \\ \omega_{is} + \sum_{s \in S_k(i,j) \in M_s} e_{ij,s} \left( t_{ij} + C_{id}^k \right), & \forall i \in N_k \setminus \{d\}. \end{cases}$$  \hspace{1cm} (8)

Moreover, the flow on a link $(i, j)$ can be expressed as the function of all hyperpath flows, i.e.

$$v_{ij} = \sum_{r \in R} \sum_{d \in D} \sum_{k \in K_{rd}} h_{ik} \tau_{ij}^k, \hspace{0.5cm} \forall (i, j) \in A. \hspace{1cm} (9)$$

### 3.2 Problem statement and formulation

Simply speaking, the strategy-based equilibrium traffic assignment (SETA) problem concerned in this study is a traffic assignment problem defined over a hypergraph. When it comes to navigating the network, the conventional TAP only allows a traveler to choose a path at the origin. Equipped with the structure embedded in a hypergraph, however, SETA can model a much richer set of navigation decisions, especially those that usually take place en-route at nodes. For example, transit passengers often face strategic decisions at transfer stops, and they may not always end up using the same itinerary. Similarly, it is difficult to assign truckers to a predetermined path, because their next move always hinge on the availability of loads and their comparative ability to win them.

In SETA, each agent in the network is committed to a strategy or a hyperpath $k$, and doing so incurs a cost $C_{id}^k$. Following Wardrop (1952), we assume that each agent chooses a hyperpath with the minimum cost, which implies a solution to the SETA can be characterized by the following user equilibrium (UE) conditions

$$C_{id}^k - u_{rd} \geq 0, \hspace{0.5cm} \forall r \in R, d \in D, \hspace{0.2cm} k \in K_{rd}, \hspace{0.5cm} (10a)$$

$$h_{ik} (C_{id}^k - u_{rd}) = 0, \hspace{0.5cm} \forall r \in R, d \in D, \hspace{0.2cm} k \in K_{rd}, \hspace{0.5cm} (10b)$$

where $u_{rd}$ is the minimum cost incurred by an agent traveling between O-D pair $(r, d)$. The above conditions simply state that at UE a hyperpath between an O-D pair is used (i.e., $h_{ik} > 0$) if and only if its cost is the minimum.

It is well known that the UE conditions can be rewritten as a variational inequality problem (VIP) such as follows (see e.g. Florian and Spiess, 1983).

**Proposition 1.** A hyperpath flow vector $h^* \in \Omega_h$ satisfies the Wardrop’s user equilibrium condition \(10\) if and only if $h^*$ is a solution to the following variational inequality problem (VIP): find $h^* \in \Omega_h$ such that:

$$C(h^*)^T (h - h^*) \geq 0, \forall h \in \Omega_h,$$

where $\Omega_h$ is defined in Eq. (5) and $C(h^*)$ is the vector of hyperpath cost given $h^*$.

### 3.3 Applications

We present three applications of SETA problems, all taken from the literature. Each of these can be viewed as a special case of SETA, with its unique definition for the three cost components,
namely \( t, e \) and \( \phi \). Naturally, our focus is to specify these cost functions and to demonstrate how SETA provides a unifying framework to these applications.

### 3.3.1 Transit equilibrium assignment problem (TEAP)

In a congested transit network consisting of a set of transit lines and stops where a stop may be served by multiple lines, passengers usually behave strategically at stops to determine whether or not to board or alight a certain line, due to the stochastic nature of transit services (e.g., fluctuations in demand, headway and travel time). As we have seen in Section 2.2, various TEAP models have been developed to represent this strategic choice behavior of transit passengers (e.g., see Nguyen and Pallottino 1988; Wu et al. 1994; Cominetti and Correa 2001; Codina 2013). Here, we use the model proposed by Wu et al. (1994) to explain how TEAP fits in the SETA framework.

Figure 2 illustrates the details of a transit station in a hypergraph, which consists of a transfer node and a dummy transit node for each line making a stop at the station. The dummy transit nodes are introduced to represent boarding and alighting movements.

Consider a passenger waiting at transfer node \( i_1 \), and assume that both lines 1 and 2 can lead him to the destination. A widely adopted strategy for the passenger is to first choose a set of attractive lines and then always board the first incoming bus in the set. The attractive set can be naturally represented as a hyper-link. In this example, possible attractive sets may be \{Line 1\}, \{Line 2\}, or \{Line 1, Line 2\} depending on the conditions (nominal frequency and passenger load) on each line. Accordingly, each set corresponds to a hyperlink originated at \( i_1 \). That is, \( O_{i_1} = \{ s_1, s_2, s_3 \} \), where

\[
\begin{align*}
  s_1 &= (i_1, \{ j_1 \}), \\
  s_2 &= (i_1, \{ j_2 \}), \\
  s_3 &= (i_1, \{ j_1, j_2 \}).
\end{align*}
\]

The cost on each type of link (see Figure 2) depends on how the congestion effect is modeled. In Wu et al. (1994), for example, they are defined as follows

- For a transit link \((j_1, j_2)\) that connects two dummy transit nodes \( j_1 \) and \( j_2 \),

  \[
  t_{j_1j_2} = a_1 \hat{c}_{j_1j_2} + \beta_1 \left(\frac{v_{j_1j_2} - v_{i_1j_1}}{\kappa f_{i_1j_1}} + \gamma_1 v_{i_1j_1}\right)^n, \tag{12}
  \]

  where \( a_1, \beta_1, \gamma_1 \) and \( n \) are positive parameters, \( \hat{c}_{j_1j_2} \) is the average vehicle travel time on transit link \((j_1, j_2)\), \( \kappa \) is the vehicle capacity (passengers/veh), \( f_{i_1j_1} \) is the nominal frequency of the corresponding boarding link \((i_1, j_1)\), and \( v_{i_1j_1} \) and \( v_{j_1j_2} \) are the passenger flow on link \((i_1, j_1)\) and \((j_1, j_2)\), respectively. Here \( \gamma_1 \) measures the impact of waiting passengers on the experienced crowdedness of those who are already on-board.

- For a boarding link \((i_1, j_1)\) that connects a transfer node \( i \) with one of its dummy transit nodes \( j \),

  \[
  t_{i_1j_1} = \beta_2 \left(\frac{v_{i_1j_1} + \gamma_2 v_{j_1j_2}}{\kappa f_{i_1j_1}} - v_{i_1j_1}\right)^n, \tag{13}
  \]
where \( \beta_2 \) and \( \gamma_2 \) are positive parameters. The latter measures the impact of on-board passengers on the experienced crowdedness of those who are waiting to board.

- For a walking link \((i_1, i_2)\) that connects two transfer nodes \( i_1 \) and \( i_2 \) (that is, a passenger is allowed to walk from one stop to another),
  \[
  t_{i_1i_2} = \alpha_2 \bar{c}_{i_1i_2},
  \]
  where \( \alpha_2 \) is the value of walking time, and \( \bar{c}_{i_1i_2} \) is the time an average passenger takes to walk from \( i_1 \) to \( i_2 \).

- For an alighting link \((j_1, i_1)\) that connects a dummy transit node \( j_1 \) with a transfer node \( i_1 \),
  \[
  t_{j_1i_1} = \alpha_2 \bar{c}_{j_1i_1},
  \]
  where \( \bar{c}_{j_1i_1} \) is the time an average passenger takes to alight the vehicle and walk to the transfer node.

In TEAP, the cost of traversing a transfer node \( i \) corresponds to the waiting time experienced by a passenger who adopts strategy \( s \) (corresponding to an attractive set), denoted as \( \varpi_{i,s} \), and the weight of a boarding link \((i, j)\) included in a strategy \( s \) corresponds to the probability of boarding that link, denoted as \( e_{ij,s} \). If we assume (i) transit vehicles have unlimited capacity; (ii) passengers arrive randomly at stops without timing according to published schedules; and (iii) the line headway is exponentially distributed (Li et al., 2015), then both \( e_{ij,s} \) and \( \varpi_{i,s} \) can be derived in the following closed form:

\[
\begin{align*}
  e_{ij,s} &= \frac{f_{ij}}{\sum_{(i,j) \in M_s} f_{ij}}, \quad \forall s \in S, i = s^-, (i,j) \in M_s, \\
  \varpi_{i,s} &= \frac{1}{\sum_{(i,j) \in M_s} f_{ij}}, \quad \forall s \in S, i = s^-.
\end{align*}
\]

The nodes forbidding transfers do not have any outgoing hyperlink with a degree greater than 1 (see Figure [1](b)). For a link \((i,j)\) originated from such a node, we simply assume \( f_{ij} = +\infty, \{(i,j)\} = M_s \), and accordingly \( \varpi_{i,s} = 0 \) and \( e_{ij,s} = 1 \).

### 3.3.2 Capacitated traffic assignment problem (CTAP)

Marcotte and Nguyen (1998) and Marcotte et al. (2004) introduce a capacitated traffic assignment problem (CTAP) that explicitly imposes a rigid upper bound on the total flow traversing a link \((i,j)\), denoted by \( \lambda_{ij} \) \((> 0)\). Unlike the conventional CTAP models in which the capacity constraint simply adds an additional “queuing delay” when activated (equivalent to the Lagrange multiplier, see e.g., Larsson and Patriksson, 1995; Nie et al., 2004), they assume travellers behave strategically at a node whose downstream links may be saturated (i.e., the capacity constraint is active). Specifically, a traveler’s strategy \( s \) at a node \( i \) is represented by \( f_s \), an ordered subset of \( O^+(i) \), and denoted as \( s = (i, f_s) \). The traveler is assumed to always select the first available outgoing link whose associated link is unsaturated (i.e., \( v_{ij} < \lambda_{ij}, \forall (i,j) \in M_s \)). To ensure feasibility, any strategy \( s \) is assumed to contain in \( M_s \) a link \((i',j')\) that is least desirable (in terms of cost) but always has an unlimited capacity \( (\lambda_{i'j'} = +\infty) \). Marcotte et al. (2004) interpret such a link as the mode of “walking” in transportation.
In the context of CTAP, the cost on each link, \( t_{ij} \), is assumed to be a positive constant, and no additional cost is imposed at any node, so \( \omega_{i,s} = 0, \forall i, s^- = i \). The focus is instead placed on the specification of the weight vector \( \mathbf{e} \), which must be obtained through a flow loading process following a topological order on an acyclic network. Given a vector of hyperpath \( \mathbf{K} \) and the corresponding flow \( \mathbf{h} \), the loading problem aims to determine the actual flow assigned to each link on every hyperpath when all hyperpath flows pass through the network. Suppose two hyperpaths \( \mathbf{K}_1 \) and \( \mathbf{K}_2 \) pass through node \( i \). The corresponding hyperlink flows, denoted as \( y_{i,s_1} \) and \( y_{i,s_2} \), are iteratively assigned to the first link with available capacity proportional to \( y_{i,s_1} \) and \( y_{i,s_2} \). The loading process is terminated when all hyperlink flows are assigned to a link. Then, the weight \( e_{ij,s} \) can be computed based on the ratio between the total flows actually assigned to each link \( (i,j) \) in the loading process and the hyperlink flow \( y_{i,s} \). This weight affects the expected cost of a hyperpath, hence playing a similar role as the congestion effect in the standard TAP.

The reader who is interested in the loading process is referred to the CAPLOAD algorithm presented in Marcotte et al. (2004). To provide a bit more details here, consider the example shown in Figure 3, which displays two hyperlinks \( s_1 \) and \( s_2 \) going out of node 1, with the preference set being \( M_{s_1} = \{ (1,2), (1,3) \} \) and \( M_{s_2} = \{ (1,4), (1,3) \} \). Also, \( y_{1,s_1} = 10, y_{1,s_2} = 20 \), with a ratio \( 1 : 2 \), and the capacities on the three links are \( \lambda_{12} = 10, \lambda_{13} = 6 \) and \( \lambda_{14} = 16 \). In the first iteration, the first choice in each hyperpath keeps receiving flows until either link is saturated. In this case, link \( (1,4) \) gets saturated first, after it receives 16 units of flow from \( s_2 \). Accordingly, the flow received by link \( (1,2) \) from \( s_1 \) in this iteration is capped at 8 as dictated by the 1:2 ratio. This leaves a residual capacity of 2 on link \( (1,2) \). In the second iteration, link \( (1,2) \) is saturated first with 2 units of flow from \( s_1 \), which requires assigning no more than 4 units of flow from \( s_2 \) to link \( (1,3) \) (the current preferred link for this hyperpath). Evidently, all hyperpath flows are successfully assigned after two iterations. Upon termination, the weight on each link is \( e_{12,s_1} = (8+2)/10 = 1, e_{13,s_1} = 0, e_{13,s_2} = 4/20 = 0.2 \) and \( e_{14,s_2} = 16/20 = 0.8 \).

3.3.3 Equilibrium in an on-line freight exchange network

Miller and Nie (2020) model the routing strategy of individual truckers who compete for the loads advertised on an online freight exchange (OFEX) platform. Underlying such a strategy, represented as a hyperpath in a space-time expanded network, is the process of bidding for the next load at an intermediate city on a tour. To form a profit-maximizing strategy, a trucker must (1) consider the probability of winning a load at a given bid price and current market competition; (2) anticipate the future profit resulted from the current decision; and (3) prioritize the bidding order among possible load options.

As shown in Figure 4, there are four types of links in an OFEX network: empty links representing empty truck movement; loaded links representing loaded truck movement; waiting links representing waiting; and dummy links representing the bridge between empty and load movements. In this context, the cost on each link \( (i,j) \), \( t_{ij} \), is the profit associated with the movement, i.e., the revenue received less the cost incurred. Clearly, the profit can turn positive only on loaded links. Since all profits are earned on a loaded link, there is no need to model the node.
cost. Hence, $\omega = 0$.

At each city $i$, a trucker following a strategy $s = (i, j_s)$ only bids for a set of feasible loads, each corresponding to a link $(i, j) \in M_s$. For a given $M_s$ there is a set of feasible bidding orders, denoted as $\Phi_s$. For any order $\phi_s \in \Phi_s$, the trucker can maximize the expected profit by solving a dynamic program (DP). By solving the DP repeatedly for each $\phi_s \in \Phi_s$, the trucker can determine the optimal order $\phi^*_s$. This then leads to an optimal bid price for each load $(i, j)$, denoted $x^*_{ij}$, and the corresponding probability of winning each load, denoted as $F_{ij}(x^*_{ij}, \phi^*_s)$.

Accordingly, the weight vector on each hyperpath can be set according to the winning probabilities. Let the optimal order $\phi^*_s = \{(i, j_1), (i, j_2), \ldots, (i, j_{|M_s|})\}$, then we have

$$e_{ij_s} = F_{ij_s}(x^*_{ij_s}, \phi^*_s),$$

$$e_{ij_m,s} = F_{ij_m}(x^*_{ij_m}, \phi^*_s) \prod_{m' = 1}^{m-1} \left(1 - F_{ij_{m'}(x^*_{ij_{m'}}, \phi^*_s)}\right), \quad m = 2, \ldots, |M_s|. \quad (19)$$

Note that the winning probability function $F_{ij}(\cdot)$ depends on the number of bids received for each load at a city, which is related to the hyperpath flow vector $\mathbf{h}$. Similar to the CTAP model of Marcotte et al. (2004), this relationship is also not available in closed form, but rather must be obtained through a loading process (Miller and Nie, 2020). The above OFEX assignment problem is defined on a time-expanded network (see Figure 4), hence the optimal policy (i.e. hyperpath) would not involve cycles.

### 4 Destination-based reformulation

In this section, we reformulate the hyperpath-based problem (11) in the space of destination-based hyperlink flows. This lays the foundation for the development of the hyperbush algorithm.
for SETA problems in the next section. The reformulation decomposes the original SETA problem by destination in the spirit of Beckmann et al. (1956) and Bar-Gera (2002). In particular, the set of all hyperpaths leading to the same destination form a destination-based sub-hypergraph $G^d(N, S^d)$ where $S^d \subseteq S$, on which a single destination subproblem of the original SETA can be defined. We use destinations rather than origins as the "root" because most hyperpath problems are cast as an agent choosing the next node to visit according to downstream information realized upon the arrival at the current decision node. Such a causal relationship fits well a destination-based formulation.

Let us define the destination-based hyperlink flow, denoted by $y_{i,s}^d$, as the flow leaving from node $i$ that travels towards destination $d$ via hyperlink $s$. Per definition (6) we have

$$y_{i,s}^d = \sum_{r \in R(d)} \sum_{k \in K_{ud,s} \in S_k} \theta_{ik}^d \omega_{kr}, \quad \forall d \in D, \; i \in N \setminus \{d\}, \; s^- = i,$$

(20)

where $R(d)$ is the set of origins destined for destination $d$.

It is worth emphasizing that the hyperlink flow vector $y$ is directly obtained by aggregating hyperpath flows on $G^d$. Even though each hyperpath is acyclic, together they could still form cycles. We will discuss later how to ensure acyclicity on such a sub-hypergraph.

**Definition 3** (Feasible set of destination-based hyperlink flows). A destination-based hyperlink flow $y = \{ y_{i,s}^d \}$ is feasible, denoted as $y \in \Omega_y$, if it satisfies the following conditions

$$y_{i,s}^d \geq 0, \quad \forall i \in N \setminus \{d\}, \; s \in O_i, \; d \in D,$$

(21a)

$$\sum_{j \in I_i} \sum_{s' \in O_{ij}(j,i) \in M_{ij}} y_{j,s'}^d \cdot e_{ji,s} - \sum_{s \in O_i, i = s^-} y_{i,s}^d = q_i^d, \quad \forall i \in N, \; d \in D,$$

(21b)

where $q_i^d$ is defined as follows

$$q_i^d = \begin{cases} 
\sum_{r \in R(d)} q_{rd} & \text{if } i = d \\
-q_{rd} & \text{if } i = r \\
0 & \text{otherwise.} 
\end{cases}$$

(22)

We further define $u_{i}^d$ as the weighted minimum cost from node $i$ to destination $d$, represented in the form of a generalized Bellman equation as

$$u_{i}^d = \begin{cases} 
\min_{s \in O_i} \left\{ \omega_{i,s} + \sum_{(i,j) \in M_s} e_{ij,s} \cdot \left( u_j^d + t_{ij} \right) \right\} & \text{if } i \neq d \\
0 & \text{if } i = d. 
\end{cases}$$

(23)

Accordingly, $u_{i,s}^d$ denotes the weighted hyperlink cost from node $i$ to destination $d$ through $s$, which is given by

$$u_{i,s}^d = \omega_{i,s} + \sum_{(i,j) \in M_s} e_{ij,s} \cdot \left( u_j^d + t_{ij} \right), \quad \forall d \in D, \; i \in N \setminus \{d\}, \; s \in O_i.$$

(24)

With the above definition, we can restate the UE conditions (10) in terms of destination-based hyperlink flows as follows

$$y_{i,s}^d (u_{i,s}^d - u_i^d) = 0, \quad \forall d \in D, \; i \in N, \; s \in O_i,$$

(25)
where \( u_{d,i}^d \geq u_{d,i}^s \) as implied by Eqs. (23) and (24). Conditions (25) state that if hyperlink \( s \) is used to travel from node \( i \) to the destination \( d \) (i.e., \( y_{d,i}^s > 0 \)), then the hyperlink \( s \) must be on the minimum-cost hyperpath between \( i \) and \( d \).

**Lemma 1.** The destination-based UE conditions (25) are equivalent to the hyperpath-based UE conditions (10).

**Proof.** See Appendix C. □

The following result shows Condition (25) can be written as a VIP (see e.g. Allen, 1977).

**Proposition 2.** A hyperlink flow vector \( y^* \in \Omega_y \) satisfies the UE condition (25) if and only if \( y^* \) is a solution to the following VIP: find a hyperlink flow vector \( y^* \in \Omega_y \) such that

\[
u(y^*)^T(y - y^*) \geq 0, \forall y \in \Omega_y,
\]

(26)

where \( \nu = \{u_{d,i}^d\}(\cdot) \) is the vector of weighted hyperlink cost, which is a function of hyperlink flow \( y \).

We next discuss the solution existence and uniqueness results concerning VIP (26).

**Theorem 1 (Existence I).** If (i) \( e \) is flow-independent and (ii) \( t \) and \( \varphi \) are continuous functions of hyperlink flow \( y \), then VIP (26) has at least one solution.

**Proof.** If \( t \) and \( \varphi \) are continuous functions of \( y \), and \( e \) is fixed, then the weighted hyperlink cost vector \( \nu \) must also be continuous with respect to \( y \) as per definition (24). Moreover, the feasible set \( \Omega_y \) is non-empty, compact, and convex because it is a polyhedral. The existence of a solution then follows from Smith (1979). □

**Theorem 2 (Existence II).** If \( \nu \) is a lower semi-continuous function of the hyperlink flow \( y \), VIP (26) has at least one solution.

**Proof.** See Ky Fan’s Inequality (Fan, 1972). □

To establish solution uniqueness requires demonstrating \( \nu \) is a strictly monotone function of \( y \). This strict monotonicity does not usually hold for the same reason why path flows are not unique in the TAP. Under certain conditions, however, one can guarantee the uniqueness of total link flows, as shown below.

**Theorem 3 (Uniqueness).** If (i) \( t \) is a continuously and strictly monotone function of \( v \) and (ii) \( e \) is flow-independent and satisfies Eq. (1), then all solutions \( y^* \) to VIP (26) correspond to the same total link flow vector \( v^* \).

**Proof.** Because \( e \) is flow-independent, the link traverse probability \( \pi_{ij}^k \) and node traverse probability \( \varphi_{ij}^k \) are also flow-independent for each hyperpath \( k \). Let \( \pi(|A| \times |h|) \) be the link-hyperpath probability matrix, we can write \( v = \pi h \). Accordingly, the hyperpath-based VIP (11) can be rewritten as follows,

\[
t(v^*)^T(v - v^*) + \tau - \tau^* \geq 0, \forall (v, \tau) \in \Omega_v,
\]

(27)

\[\text{1A function } f: \mathbb{R}^n \rightarrow \mathbb{R} \text{ is lower semi-continuous at point } \bar{x}, \text{ if } \forall \epsilon > 0, \exists \eta > 0 \text{ such that when } ||x - \bar{x}|| < \eta, f(x) - f(\bar{x}) \leq \epsilon. \text{ Equivalently, } f(\bar{x}) \leq \liminf_{x \rightarrow \bar{x}} f(x).
\]

\[\text{2A function vector } f: \mathbb{R}^n \rightarrow \mathbb{R} \text{ is strictly monotone if for all } x_1, x_2 \in \mathbb{R}^n, x_1 \neq x_2, (x_2 - x_1)^T (f(x_2) - f(x_1)) > 0.
\]
where 
\[
\Omega_v \triangleq \{ (v, \tau) | v = \pi h, h \in \Omega_k \}, \quad \text{and} \quad \tau = \sum_{r \in K} \sum_{d \in D} \sum_{i \in N_k, \sigma(i,d), \delta \in S_k} \theta_i^k \Omega_i,s h_{lk}
\]
is the total node cost corresponding to a given hyperpath flow vector \( h \). The uniqueness of the solution to (27) can then be proven by invoking Theorem 2 in [Wu et al., 1994].

**Assumption 1.** At least one of the following two conditions holds: (i) the hypergraph \( G(N, S) \) is acyclic; and (ii) the cost vectors \( t \) and \( \omega \) and the weight vector \( e \) are such that at UE, for each \( y_{i,s}^d > 0, u_{i,s}^d > u_{i,s}^d, \forall (i,j) \) such that \((i,j) \in M_s \). Of the three applications discussed in Section 3.2, the capacitated assignment problem [Marchette and Nguyen, 1998], and the OFEX truck assignment problem [Miller and Nie, 2020] are both defined on acyclic networks, thus satisfying the first condition above. Specifically, the OFEX network shown in Figure 4 is a space-time expanded network. Such a network is acyclic by definition. The capacitated model was applied to solve the schedule-based transit assignment problem that is also defined on a space-time expanded network [Hamdouch and Lawphongpanich, 2008, 2010; Hamdouch et al., 2014].

Condition (ii) in Assumption 1 may be violated when both \( \omega_{i,s} \) and \( t_{ij} \) are zero. Thus, ensuring each link \((i,j)\) have a positive \( t_{ij} \) would avoid this problem. As to the TEAP model, the following lemma asserts that \( t \), \( e \) and \( \omega \) specified in Equations (12)-(17) would satisfy Condition (ii) in Assumption 1.

**Lemma 2.** For any hyperlink \( s \) that carries positive flow at UE, \( t \), \( e \) and \( \omega \) specified in (12)-(17) of TEAP ensure \( u_{i,s}^d \geq u_{i,s}^d + t_{ij} \), \( \forall (i,j) \in M_s \).

**Proof.** The proof is completed by contradiction. Suppose hyperlink \( s \) is used at UE (i.e., it carries a positive flow) and there exists a link \((i,j) \in M_s \) such that \( u_{i,s}^d = u_{i,s}^d < u_{i,s}^d + t_{ij} \). If \( M_s \) contains only link \((i,j)\), the above statement clearly contradicts with the definition of \( u_{i,s}^d \) (see Eq. (23)); otherwise, we shall show that there exists hyperlink \( s' \) with \( M_{s'} = M_s \setminus \{(i,j)\} \) such that \( u_{i,s'}^d < u_{i,s}^d \), which implies hyperlink \( s \) should not be used at UE, also a contradiction. By definition (16) and (17), we have

\[
u_{i,s'}^d \left( \sum_{(i,j') \in M_{s'}} \frac{f_{ij'}}{M_{s'}} \right) = 1 + \left( \sum_{(i,j') \in M_{s'}} f_{ij'} \left( u_{i,s}^d + t_{ij'} \right) \right) = 1 + \left( \sum_{(i,j') \in M_{s}} f_{ij'} \left( u_{i,s}^d + t_{ij'} \right) \right) - f_{ij} \left( u_{i,s}^d + t_{ij} \right)
\]

\[
u_{i,s}^d \left( \sum_{(i,j') \in M_{s}} f_{ij'} \right) - f_{ij} \left( u_{i,s}^d + t_{ij} \right) < u_{i,s}^d \left( \sum_{(i,j') \in M_{s}} f_{ij'} \right) - f_{ij} u_{i,s}^d \]  

(28)

where the first equality is due to Equation (24) and the first inequality holds because of the assumption \( u_{i,s}^d < u_{i,s}^d + t_{ij} \). This completes the proof.

Condition (ii) in Assumption 1 states that any used hyperlink at UE would bring agents “closer” (as measured by the weighted cost) to the destination, or equivalently they will not choose a hyperlink that may take them to a “farther” place (e.g., see Lemma 2). This is consistent
with the definition of an efficient hyperlink such that $u^h_i > u^h_j, \forall j \in J, s^- = i$ (Nguyen and Pallottino 1989). It is clear not all SETA problems can satisfy Assumption 1. When cycling is indeed a requirement for optimality (as demonstrated in Appendix B), if Assumption 1 is not satisfied for a SETA problem, one has two options: sticking to a hyperpath-based formulation, or imposing acyclicity as an external condition. The latter promises efficiency but may cause a solution so obtained to deviate from the true optimum. The choice hence hinges on the trade-off between efficiency and accuracy, which is affected by the nature and size of the problem.

We are now ready to give the following result, which lays the foundation for the hyperbush algorithm presented in the next section.

**Theorem 4 (Acyclicity at UE).** Under Assumption 1, each destination-based sub-hypergraph $G^d(N, S^d)$ formed by the hyperlinks that carry a positive flow at UE contains no direct cycle.

**Proof.** Condition (i) in Assumption 1 is obviously a sufficient condition for acyclicity. For Condition (ii), we prove by contradiction. Suppose a sub-hypergraph formed by the hyperlinks that carry a positive flow at UE contains a direct cycle that consists of nodes $\{i_0, i_1, \cdots, i_0\}$. According to Condition (ii), we have $u^d_{i_0} > u^d_{i_1} > \cdots > u^d_{i_0}$, a contradiction. □

5 Hyperbush algorithm

In this section, we develop a general algorithmic framework that exploits the acyclicity of equilibrium hyperlink flows (Theorem 4). In Section 5.1 we define hyperbush and describe a generic hyperbush algorithm (HBA) for SETA problems. Sections 5.2 and 5.3 discuss the two main building blocks of the algorithm: hyperbush expansion and hyperbush equilibrium, respectively.

5.1 Overview of the algorithm

Let us begin with a formal definition of hyperbush.

**Definition 4 (Hyperbush).** A destination-based sub-hypergraph $G^d(N, S^d)$ is called a hyperbush if it (i) is acyclic, (ii) has at least one hyperpath from every node $i \in N \setminus \{d\}$ to $d$.

Let $A^d = \bigcup_{s \in S^d} M_s \subseteq A$ be the set of all links included in $G^d$.

**Definition 5 (Topological distance).** Let the length of each link $(i, j) \in A^d$ be 1. The topological distance of node $i$ on a hyperbush $G^d$, denoted as $\rho_i$, is the maximum distance from $i$ to $d$ through a regular path.

**Definition 6 (Backward/forward pass).** A backward/forward pass is a sequential visit to each node in a hyperbush $G^d$ following an/a increasing/decreasing order of the node topological distance.

**Definition 7 (Hyperpath tree).** A destination-based sub-hypergraph is a hyperpath tree, denoted as $T^d(N, S^d)$, if (i) $O(d) \cap S^d = \emptyset$ and (ii) $|O(i) \cap S^d| = 1, \forall i \in N \setminus \{d\}$.

Thanks to acyclicity, the minimum and maximum weighted cost hyperpath tree on hyperbush $G^d(N, S^d)$ can be built efficiently by applying the Bellman equation in a single backward pass.
starting from $d$, as shown below.

\[
\begin{align*}
  u_i^d &= \min_{s \in S^d \cap \Omega_i} \left\{ u_{i,s}^d = \omega_{i,s} + \sum_{(i,j) \in M_i} e_{i,j,s} \cdot (u_j^d + t_{ij}) \right\}, \quad \forall i \in N \setminus \{d\}, \\
  U_i^d &= \max_{s \in S^d \cap \Omega_i} \left\{ U_{i,s}^d = \omega_{i,s} + \sum_{(i,j) \in M_i} e_{i,j,s} \cdot (U_j^d + t_{ij}) \right\}, \quad \forall i \in N \setminus \{d\}, \\
  u_d^d &= U_d^d = 0.
\end{align*}
\]

Here, $u_i^d$ and $U_i^d$ are respectively the minimum and maximum weighted cost from $i$ to $d$.

Theorem 4 asserts all hyperlinks carrying positive flows at UE would form no cycles. This does not mean, however, the sub-hypergraph formed by these hyperlink flows is necessarily a hyperbush, because Condition (ii) in Definition 4 may not be satisfied. Instead, we formally define a hyperbush corresponding to a UE hyperlink flow vector $y^*$ as follows:

**Definition 8** (UE hyperbush). A hyperbush is called a UE hyperbush, denoted as $G_d^*(N, S_d^*)$, if for any $s \in S^d \setminus S_d^*$ such that $u_d^d(y^*) - u_i^d(s) < 0$, $i = s^-$.

The main challenge is that the topology of the UE hyperbush is generally unknown a priori, and must be constructed iteratively. At the heart of a hyperbush algorithm is thus how to address this challenge. Similar to the bush-based algorithms for TAP (Bar-Gera, 2002; Dial, 2006; Nie, 2010), the basic idea is to start from an initial hyperpath tree, and then gradually refine it according to the UE conditions. Accordingly, the algorithm consists of two main steps: (i) expanding/trimming the hyperbush; and (ii) equilibrating flows among hyperpaths contained in the current hyperbush (i.e. allocating flows such that all used hyperpaths have the same and minimum cost). The second step is effectively equivalent to solving the original SETA problem over a restricted network, which is often called a restricted master problem (RMP). The UE conditions for the RMP corresponding to the current hyperbush $G_d^*(N, S_d^*)$ read

\[
\begin{align*}
  u_{i,s}^d - u_i^d &\geq 0, \quad \forall i \in N, \ s \in S^d, \ s^- = i, \\
  y_{i,s}^d (u_{i,s}^d - u_i^d) &= 0, \quad \forall i \in N, \ s \in S^d, \ s^- = i, \\
  y_{i,s}^d &= 0, \quad \forall i \in N, \ s \notin S^d, \ s^- = i.
\end{align*}
\]

Compared to Conditions (25), the above conditions emphasize $s$ must be on the current hyperbush to be considered (otherwise its flow is automatically set to zero).

By maintaining acyclicity in the solution process, the hyperbush algorithm promises computational efficiency. To illustrate this advantage, consider a simple network shown in Figure...
which has 4 nodes and each consecutive pair is connected by 3 parallel links. In this case, the possible number of hyperpaths from node 1 to 4 is \((2^4 - 1)^3\), an order of magnitude larger than the number of all simple paths \((3^3)\) and the number of hyperlinks \((2^4 - 1) \times 3\). By using a hyperbush representation, the algorithm can work in the space of hyperlink flows to avoid dealing with an unmanageable number of hyperpaths. This leads to potentially very large savings in terms of memory consumption and computation time. As a result, the algorithm could significantly increase the scale of the SETA models that can be solved and improve the quality of solutions that can be obtained, within a reasonable amount of computation time.

We are now ready to give a generic description of the hyperbush algorithm (HBA). Up to this point in this section, the focus is on a single destination problem rooted at the destination \(d\). Yet, we note that a general SETA problem with multiple destinations can be decomposed with respect to destinations. That is, one can construct and equilibrate each destination-based hyperbush sequentially, while holding the assignment results on other hyperbushes constant. The idea can be traced back to the seminal contribution of Beckmann et al. (1956), and has been widely used in the literature on TAP since Bar-Gera (2002). The main steps are comprised of the following.

Step 1. **Initialization:** Compute an initial hyperpath tree for each destination \(d \in D\). Assign all flows to the hyperlinks on this hyperbush according to Eq. (20). Update \(t, e\) and \(\omega\).

Step 2. **Main Loop:** For each hyperbush \(G^d(N, S^d)\)

   Step 2-1. **Hyperbush expansion:** Expand hyperbush by adding new hyperlinks that have the potential to reduce cost without introducing cycles.

   Step 2-2. **Hyperbush equilibration:** Adjust hyperlink flows on the hyperbush to satisfy UE conditions (30), and update \(t, e\) and \(\omega\).

Step 3. **Inner Loop:** For each hyperbush \(G^d(N, S^d)\), adjust hyperlink flows on the hyperbush to satisfy UE conditions (30), and update \(t, e\) and \(\omega\). Calculate the convergence measure

\[
\Phi_d = \max_{i \in N} \left\{U^d_i - u^d_i \right\}.
\]

If \(\Phi_d > \epsilon\) and the maximum allowed number of inner loops is not reached, repeat Step 3; otherwise go to Step 4.

Step 4. **Trim hyperbush:** Removing unused hyperlinks with zero flow from \(S^d, \forall d \in D\).

Step 5. **Convergence test:** If a satisfactory convergence or the maximum running time is not achieved, return to Step 2; otherwise stop.

A few remarks are in order here. First, the initial hyperpath tree can be obtained using a standard hyperpath routing algorithm (Nguyen and Pallottino, 1989). Second, in Step 4, we need to keep at least one hyperlink at a node (except \(d\)) in the hyperbush to ensure Condition (ii) in Definition 4 be satisfied. Finally, hyperbush expansion and equilibration are the key ingredients of the algorithm that we are now set out to describe in the next two subsections.
5.2 Hyperbush expansion

A destination-based hyperbush is expanded based on an equilibrium solution restricted by its topology. The underlying question is whether the current hyperbush leaves out hyperlinks that, if included, could further reduce the cost for some agents. Let us define the set of such hyperlinks as $B_1 = \{s|s \notin S^d, s = i, u_{i,s}^d < u_{i,s}^d\}$, where both $u_{i,s}^d$ and $u_{i,s}^d$ are computed based on the current solution. Thus, if we choose to add a hyperlink $s \in B_1$ to the hyperbush, travelers starting from (or passing through) $i$ could enjoy a smaller cost by switching to the new hyperlink. If $B_1$ turns out to be empty, then the optimality can be declared for the destination-based subproblem.

While the above idea seems straightforward, it has an important caveat: a cost-improving hyperlink might create cycles in the current hyperbush if the original network is not acyclic. Having cycles not only undermines the validity of all computational procedures that requires a topological order, but also violates optimality as per Theorem 4. Thus, expanding a hyperbush requires identifying links that (i) are not contained in the current hyperbush, (ii) have the potential to reduce the weighted cost at some nodes, and (iii) would not create cycles. The following result provides one strategy to preserve acyclicity through expansion.

**Proposition 3** (Acyclicity preservation I). Under Assumption 1 and assume the RMP defined on the hyperbush $G^d(N,S^d)$ is optimally solved, the hypergraph with links $A^d \cup \{(i,j)|(i,j) \notin A^d, u_{i,j}^d > u_{i,j}^d\}$ remains a hyperbush.

**Proof.** Suppose that a direct cycle with node sequence of $\{i_0,i_1,\ldots,i_0\}$ forms after an addition of a link within the hyperlink $s$. Along the cycle, we have $u_{i_0}^d < u_{i_0}^d$ by the above two conditions. This is clearly impossible. $\square$

The above strategy works only if the RMP is optimally solved, which would ensure for every link $(i,j)$ included in $A^d$, $u_{i,j}^d > u_{i,j}^d$. However, achieving a “perfect” UE solution is technically impossible. While a highly precise UE solution may reduce the risk of violating the above requirement, it cannot eliminate such a risk all together. Moreover, pursuing a highly precise RMP solution is expensive and may not be a worthwhile effort, especially when the current hyperbush itself is far from the optimal topology.

An alternative strategy is to replace $u_{i,j}^d$ with the maximum weighted cost $\bar{U}_{i,j}^d$, as suggested by Nie (2010). However, there is no guarantee that $\bar{U}_{i,j}^d > U_{i,j}^d, \forall (i,j) \in A^d$, for precisely the same reason. As a remedy, we propose a maximum dummy cost $\bar{U}_{i,j}^d$ obtained through recursion as follows:

\[
\bar{U}_{i,j}^d = \max_{s \in S^d: x_i} \left\{ \max \left( \omega_{i,s} + \sum_{(i,j) \in M_s} \epsilon_{ij,s} \cdot \left( \bar{U}_{i,j}^d + t_{ij} \right), \max_{(i,j) \in M_s} \left( t_{ij} + \bar{U}_{i,j}^d \right) \right) \right\}, \forall i \in N \setminus \{d\},
\]

(32a)

\[
\bar{U}_{i,j}^d = 0.
\]

(32b)

The proposed dummy cost is such set that the weighted maximum dummy cost of any hyperlink $s$ (the first term in the second max operator) must be no less than the max dummy cost of using any link $(i,j) \in M_s$ to the destination (the second term in the second max operator). Accordingly, we define

\[
L = \{(i,j)|(i,j) \in A \setminus A^d, \bar{U}_{i,j}^d > \bar{U}_{i,j}^d\}.
\]

(33)
**Proposition 4** (Acyclicity preservation II). Under Assumption \( \square \) and assume \( t_{ij} > 0 \) for all \((i, j) \in A, a hyperbush \( G^d(N, S^d) \) remains acyclic after adding one or more links \((i, j) \in L \).

*Proof.* For any link \((i, j) \in A^d, we have \( \hat{U}_i^d > \hat{U}_j^d \). This is because, per (32), \( \hat{U}_i^d \geq \hat{U}_j^d + t_{ij} \) and \( t_{ij} > 0 \). The existence of any cycle would contradict with the fact that \( \hat{U}_i^d > \hat{U}_j^d \) for all links along that cycle. \( \square \)

While adding any links from \( L \) would ensure acyclicity, it does not guarantee reducing the minimum weighted cost at any node. This is because \( \hat{U}_i^d \), unlike \( u_i^d \) or \( U_i^d \), is not directly linked to the optimal conditions. To address this issue, we propose to first select, at a node \( i \neq d \), the hyperlink

\[
s^* = \min \{u_{i,s}^d | s \in B_2\},
\]

where \( B_2 = \{s | s \notin S^d, s^- = i, M_s \subseteq (A^d \cap O(i)) \cup (L \cap O(i)), u_{i,s}^d < u_i^d\} \). If \( B_2 \neq \emptyset \), then we update \( A^d = A^d \cup M_s^r \). Note that this way we guarantee the newly updated hyperbush has the potential to reduce the weighted cost, i.e., it provides a descent direction.

Algorithm \( \square \) summarizes the procedure for hyperbush expansion, which consists of two main steps carried out following an increasing topological order. Step 1 updates the minimum weighted cost and the maximum dummy cost (cf. lines 5-13). This is followed by adding new hyperlinks in Step 2. For each node \( i \), all links belong to \((A^d \cap O(i)) \cup (L \cap O(i))\) are added into a temporary link set \( B_i^d \) as candidates (cf. lines 17-21). We then search through all subsets of \( B_i^d \) to find \( s^* \) that solves the optimization problem (34). It is worth noting that, when \( B_i^d \) contains more than a few links, enumerating all subsets of \( B_i^d \) could be expensive computationally. A potential bypass is to apply a greedy algorithm, which sequentially adds links into \( M^r \) according to a predetermined order. In some applications, such a greedy algorithm can guarantee optimality. An example is TEAP, as we shall see later. In other cases, the greedy algorithm may be used as a performance-enhancing heuristic. Note that the greedy algorithm has a complexity of \( O(|B_i^d|) \) compared to \( O(2^{|B_i^d|}) \) in a brute-force enumeration. When implemented with the greedy algorithm, the complexity of hyperbush expansion is roughly \( O(\sum_{s \in S^d}|M_s| + \sum_{i \in N}(|O(i)| + |B_i^d|)) \) in Algorithm \( \square \)

### 5.3 Hyperbush equilibration

We now turn to the problem of solving the RMP, i.e., equilibrating flows on the destination-based hyperbush that defines the RMP. Since the RMP is defined on an acyclic graph, we propose to adopt the algorithm suggested by [Dafermos (1972)](Dafermos1972) and operationalized in Dial’s Algorithm B ([Dial] 2006). The original idea is to shift flows between the shortest and the longest paths between each O-D pair until the costs on them are equalized. [Dial] 2006 suggests this flow equilibration can be performed between any node pair on a bush that is connected by more than one path. In what follows, we show how this idea can be extended to paired hyperpaths defined on a hyperbush. To facilitate the discussion, let us first introduce the following definition.

**Definition 9** (First pseudo destination). For any given node \( i \neq d \) in a hyperbush, a node \( j \) is the first pseudo destination (FPD) of \( i \) if (i) \( \rho_j > \rho_j \); (ii) \( \theta_{i,k} = \theta_{j,k} = 1 \) when \( k \) is either the minimum or maximum weighted cost hyperpath; and (iii) \( \exists j' \neq j \) such that \( j' \) satisfies (i) and (ii), and \( \rho_i - \rho_j < \rho_i - \rho_{j'} \).
Algorithm 1 Hyperbush expansion

1: **Input:** Hyperbush $G^d(N, S^d)$.
2: **Initialization:** Set $u_i = +\infty$ and $\bar{U}_d = -\infty$, $\forall i \in N \backslash \{d\}$, $u_d = \bar{U}_d = 0$, and $n = 0$.
3: for each $i \in N$ in an increasing topological order do
4:     **Step 1:** Update $u_i$ and $U_i'$ by Eq. (29a) and (32a) respectively.
5:     for each $s \in S^d \cap O_i$ do
6:         Let temp variables $a = b = \omega_{i,s}$, $z = -\infty$.
7:         for each $(i, j) \in M_s$ do
8:             Update $a = a + e_{ij} (u_d + t_{ij})$, $b = b + e_{ij} (\bar{U}_d + t_{ij})$.
9:             Update $z = \bar{U}_d + t_{ij}$ if $\bar{U}_d + t_{ij} > z$.
10:         end for
11:     Update $u_i = a$ if $a < u_i$.
12:     Update $\bar{U}_d = \max(z, b)$ if $\max(z, b) > \bar{U}_d$.
13: end for
14: **Step 2:** Add a new hyperlink to improve $G^d$.
15: if $i \neq d$ then
16:     Initialize a temporary link set $B_i' = \emptyset$, $s^* = \emptyset$, $u_{i,s^*} = +\infty$.
17:     for each $(i, j) \in O(i)$ do
18:         if $((i, j) \notin A^d$ and $\bar{U}_d > \bar{U}_d$) or $(i, j) \in A^d$ then
19:             Set $B_i' = B_i' \cup \{(i, j)\}$.
20:     end if
21: end for
22: for each $s$ such that $M_s \subseteq B_i'$ do
23:     Calculate $u_{i,s}$ by Eq. (24).
24:     if $u_{i,s} < u_{d,s}$ and $u_i > u_d$, $\forall j \in J_s$ then
25:         Set $u_{i,s^*} = u_{i,s}$, $s^* = s$.
26:     end if
27: end for
28: if $u_{i,s^*} < u_i$ and $s^* \notin S^d$ then
29:     Update $S^d = S^d \cup s^*$ and $A^d = A^d \cup M_{s^*}$, $n = n + 1$.
30: end if
31: end for
32: if $n > 0$ then
33:     Update topological order on $G^d$.
34: end if
35: **Output:** Updated $G^d(N, S^d)$.
Accordingly, a shortest (longest) hyper-segment starting at node $i$ on a hyperbush rooted at $d$ is defined as the portion of the minimum (maximum) weighted cost hyperpath tree between $i$ and its FPD. A pair of hyper-segments at node $i$ consisting of the shortest and the longest hyper-segments is called the extreme pair of hyper-segments (EPHS) for $i$, denoted as $E_i(\mathbf{k}, \mathbf{l})$, where $\mathbf{k}$ and $\mathbf{l}$ can each be viewed as a hyperpath between $i$ and its FPD. The basic flow equilibration operation hence involves shifting flows within each EPHS.

To better illustrate the concept of EPHS, consider the example shown in Figure 6, where node 9 is the FPD of node 1. In the figure, we use $\mathbf{s}_m$ and $\mathbf{s}_m$ to represent, respectively, the hyperlinks at node $m$ on the minimum and maximum weighted cost hyperpath trees. Figures 6(a) and 6(b) show the portion of the minimum and maximum weighted cost hyperpath tree between nodes 1 and 9, respectively. Combining them together forms an EPHS $E_1(\mathbf{k}, \mathbf{l})$ in Figure 6(c), where the thick red hyperlinks represent the maximum weighted cost hyper-segment $\mathbf{k}$ and the thin blue hyperlinks represent the minimum weighted cost hyper-segment $\mathbf{l}$. Per Eq. 6, we have $\phi^k_1 = \phi^k_9 = \phi^k_9 = \phi^k_9 = 1$. We note that a critical difference between FPD and last common node
on a regular bush [Bar-Gera, 2002] is that the paired alternative segments between \( i \) and its last common node share no nodes other than the two ends. The two hyper-segments included in the EPHS, however, can share more than the two ends. As shown in Figure 6(c), node 3, in addition to 1 and 9, is shared by the EPHS. Because of this complexity, finding FPD itself requires attention, as discussed in the following.

We begin by noting the FPD for any node \( i \neq d \) must exist in a destination-based hyperbush, because all hyperpaths must converge at the destination \( d \) in such a hypergraph. Algorithm 2 describes how to find a node \( i \)'s FPD and EPHS. The idea is to track the traverse probability of the nodes “downstream” (i.e., with a lower topological order) of node \( i \), over either the minimum or the maximum weighted cost hyperpath tree. Since the traverse probability is initialized to 1 at node \( i \), the “nearest” (in terms of topological distance) downstream node where the traverse probability reaches 1 on both trees is the FPD of \( i \). In the worst case, the algorithm will find a FPD after all nodes and links on both trees are visited, leading to a complexity of \( O(|N| + |N| + \sum_{s \in S^d} |M_s| + \sum_{s \in S^d} |M_s|) \).

**Algorithm 2** Generation of extreme pair of hyper-segments

1: **Inputs:** \( G^d(N,S^d) \), the maximum weighted cost hyperpath tree \( \overrightarrow{T}^d(N,S^d) \), the minimum weighted cost hyperpath tree \( \overleftarrow{T}^d(N,S^d) \), and a starting node \( i \neq d \).
2: **Initialization:** \( j_1^* = j_2^* = i \), \( k = k' = \emptyset \).
3: repeat
   4:   **Step 1:** find the nearest downstream converge node for \( j_1^* \) on \( \overrightarrow{T}^d \).
   5:   **while** \( \rho_{j_1^*} > \rho_{j_2^*} \) or \( j_1^* = i \) **do**
   6:      Call Algorithm 3 to find the first \( j^* \) along the hyper-segment \( k' \) on \( \overrightarrow{T}^d \) such that \( \vartheta_{j^*}^{k'} = 1 \).
   7:      Set \( j_1^* = j^* \), and join \( k = k' \cup k \).
   8:   **end while**
9:   **Step 2:** find the nearest downstream converge node for \( j_2^* \) on \( \overrightarrow{T}^d \).
10:  **while** \( \rho_{j_2^*} > \rho_{j_1^*} \) or \( j_2^* = i \) **do**
11:     Call Algorithm 3 to find the first \( j^* \) along the hyper-segment \( k' \) on \( \overrightarrow{T}^d \) such that \( \vartheta_{j^*}^{k'} = 1 \).
12:    Set \( j_2^* = j^* \), and join \( k = k' \cup k \).
13:  **end while**
14: **until** \( j_1^* = j_2^* \)
15: **Output:** EPHS \( E_i(k, k) \).

Once an \( E_i(k, k) \) is identified, we simply shift flows from \( k \) to \( k \) to equalize their cost. It is worth emphasizing, due to potential asymmetric and non-monotone interactions, this strategy does not always monotonically decrease the cost on \( k \) and increase the cost on \( k \). Also, equalizing their cost should be interpreted as enforcing the pair to satisfy the UE conditions, which does not always result in the identical cost on both hyperpaths. The key is to determine how much flow should be shifted in order to satisfy the UE conditions. Typically, this means a step size must be found through a line search procedure (e.g., the bisection method and the Armijo’s rule). How quickly a good (if not optimal) step size can be found largely depends on the properties of the cost and weight functions. When the derivative of the weighted cost with respect to hyperpath flow is available in closed form, that information can be used to scale a predetermined step size, thus bypassing the line search procedure all together (Dial, 2006; Nie, 2010).
Algorithm 3 Find the nearest downstream converge node

1: **Input:** A hyperpath tree $T^d$, starting node $i$.
2: **Initialization:** Set $k' = \emptyset$, $Q = \{i\}$, and a flag $a = 0$.
3: **repeat**
4:   Take the first node $n$ from $Q$, and set $Q = Q \setminus \{n\}$.
5:   Add the hyperlink $sn$ on $T^d$ into $k'$.
6: **for** each link $(n, p) \in M_{sn}$ **do**
7:     Update $\varphi_{kp}$ and $\pi_{kp}$ according to Eq. (6).
8:     Set $Q = Q \cup p$ if $p \notin Q$.
9:     **if** $\varphi_{kp} = 1$ **then**
10:        Set $j^* = p$, $a = 1$.
11: **end if**
12: **end for**
13: **until** $a = 1$
14: **Output:** $j^*$ and $k'$.

Let $\Delta$ denote the flow to be shifted from $k$ to $k$. We have
\[
\Delta = \min \{ \mu (C_k - C_k), \chi \},
\] (35)
where $\mu$ is the step size, $\chi$ is the maximum flows allowed to be shifted from $k$, and $C_k$ and $C_k$ are the maximum and minimum hyper-segment cost (recursively evaluated using Eq. (3)). Note that the shifted flow cannot exceed $\chi$ at any node because otherwise it will produce a negative hyperlink flow. If weight function $e$ is flow-independent, the maximum allowed flow
\[
\chi = \min_{l \in N_k \setminus \{j\}} \left( \frac{y_{l}^{d}}{\sigma_{l}} \right),
\] (36)
and the hyperlink flows can be updated by
\[
\tilde{y}_{l}^{d} = y_{l}^{d} - \Delta \cdot \sigma_{l}^{R}, \quad \forall l \in N_k \setminus \{j\}, \tag{37a}
\]
\[
\tilde{y}_{l}^{d} = y_{l}^{d} + \Delta \cdot \sigma_{l}^{K}, \quad \forall l \in N_k \setminus \{j\}. \tag{37b}
\]
If $e$ is flow-dependent, the simple flow shifting strategy introduced above no longer works. This is because shifting flow within an EPHS may change $e$, which in turn triggers redistributing flows that might ultimately violate the flow conservation constraints (21). Given the delicacy of the issue, we address it in detail when discussing the implementation of CTAP in the next section.

6 Implementations

This section tailors the general framework of the hyperbush algorithm presented in Section 5.1 to different applications. We focus on two applications discussed in Section 3.3: the transit equilibrium assignment problem (TEAP) and the capacitated traffic assignment problem (CTAP). The online freight exchange (OFEX) assignment problem is excluded here, because the OFEX problem bears similarities with CTAP: they are both defined on an acyclic network, and they both rely on a complex loading process to evaluate weighted costs.
6.1 TEAP

The cost and weight functions of TEAP, as defined in Section 3.3.1, have special structures that can be exploited for efficient implementation of the hyperbush algorithm. Specifically, two operations can be accelerated substantially: the solution of Problem (34) in hyperbush expansion, and the line search in hyperbush equilibration.

To begin, the algorithm must first identify an initial hyperbush for each destination. This task can be carried out by applying a label-setting optimal hyperpath algorithm over a free-flow (i.e., assuming all link flows be zero) network (see e.g., Spiess and Florian 1989; Li et al. 2015; Xu et al. 2020a). Once a hyperbush is obtained, all O-D flows associated with the destination can be loaded on the hyperbush, leading to an all-or-nothing assignment solution.

We next discuss the solution of Problem (34), i.e., finding the optimal hyperlink $s^*$ at node $i$ from a set of outgoing links $B_i$ (cf. lines 17-21 in Algorithm 1). In the context of TEAP, such an $i$ is usually a transit stop where a transfer can take place. If node $i$ forbids transfer (i.e., $B_i$ only includes non-boarding links), $|M_s| = 1$, and $s^*$ can be simply obtained by taking the link $(i, j)$ in $B_i$ such that $t_{ij} + u^d_{ij}$ is the minimum. Otherwise, Algorithm 4 finds $s^*$ using a Greedy method that enjoys a complexity of $O(|B_i|)$ (compared to $O(2^{|B_i|})$ in a brute-force enumeration). For the TEAP defined in Section 3.3.1, the algorithm actually ensures optimality (see e.g. Li et al. 2015).

Algorithm 4 Greedy algorithm for finding $s^*$ at node $i$

1: Input: Candidate link set $B_i$ and $t_{ij} + u^d_{ij}$, $\forall (i, j) \in B_i$.
2: Initialization:
3: Set $\omega_{i,s^*} = 0$, and $u^d_{i,s^*} = +\infty$.
4: Sort $B_i = \{1, 2, \ldots, n\}$ (n = $|B_i|$) such that $u^d_{i1} + t_{i1} \leq u^d_{i2} + t_{i2} \leq \cdots \leq u^d_{in} + t_{in}$.
5: Set $l = 1$, $M_s = M_s \cup (i, j_l)$.
6: Update $e_{ij,s}$, $\omega_{i,s}$, and $u^d_{i,s}$ by Eq. (16), (17) and (24) respectively.
7: Set $l = l + 1$, $s = s^*$, $M_s = M_s \cup (i, j_l)$. Update $e_{ij,s}$, $\omega_{i,s}$ and $u^d_{i,s}$.
8: while $u^d_{i,s} < u^d_{i,s^*}$ and $l \leq n$ do
9:   Set $l = l + 1$.
10:   if $l \leq n$ then
11:      Set $M_s = M_s \cup (i, j_l)$. Update $e_{ij,s}$, $\omega_{i,s}$ and $u^d_{i,s}$.
12:     end if
13: end while
14: Output: $s^*$.

In TEAP, the derivative of the hyperpath cost with respect to hyperpath flow can be obtained analytically. This property enables the application of Newton-type methods in the line search. Specifically, the step size $\mu$ in Eq. (35) can be simply approximated as the reciprocal of $\partial(C_k - C_k) / \partial h_k$, following the same logic used to develop the gradient projection algorithm (Jayakrishnan et al. 1994) and Algorithm B (Dial 2006). Namely,

$$
\frac{\partial(C_k - C_k)}{\partial h_k} = \sum_{(i,j) \in M_k} (\pi_{ij}^k - \pi_{ij}^k) \left( \sum_{j' \in M_k} \frac{\partial t_{ij}}{\partial v_{ij}} (\pi_{ij}^{k'} - \pi_{ij}^{k'}) \right) \approx \frac{1}{\mu}.
$$

(38)
where \( \varphi_{ij} \) denotes the set of links that have interactions with link \((i, j)\) (cf. the link cost functions \([12]\) and \([13]\)).

Finally, to measure the convergence of the hyperbush algorithm as it applies in TEAP, we use the relative gap, which is computed by

\[
RG = 1 - \frac{\sum_{(r,d) \in W} u_r^d \tau^d}{\sum_{(i,j) \in A} t_{ij} v_{ij} + \tau}, \tag{39}
\]

where \( \tau \) is total waiting cost, \( \sum_{(i,j) \in A} t_{ij} v_{ij} \) is the total link travel cost, and \( u_r^d \) is the minimum travel cost between O–D pair \((r, d)\), all evaluated based on the current flow pattern.

### 6.2 CTAP

As explained in Section 3.3.2, the flow-dependent weight vector \( e \) in CTAP is evaluated by a loading process (called CAPLOAD) following a topological order on an acyclic hypergraph \([\text{Marcotte et al.}, 2004]\). This section explains how the network loading procedure is implemented in the hyperbush algorithm for CTAP and addresses several other implementation issues.

Since CTAP is defined over an acyclic network, initializing a hyperbush is straightforward: we simply assign a feasible outgoing hyperlink (a hyperlink is considered feasible if it contains at least one link with unlimited capacity, see \([\text{Marcotte et al.}, 2004]\)) to each node \( i \neq d \). To obtain the hyperlink flow vector \( y \) and the weight vector \( e \) for the initial hyperbush, we first load \( q_{i}^d \) onto the only outgoing hyperlink for each node \( i \in R(d) \), and then distribute these O–D flows by calling the network loading procedure.

To solve Problem (34) in hyperbush expansion, we also employ a greedy algorithm (cf. Proposition 1 in \([\text{Marcotte et al.}, 2004]\)). At node \( i \neq d \), the candidate link set \( B_i^t \) can be simply taken as \( O(i) \) thanks to acyclicity. The greedy algorithm for CTAP consists of three main steps: (i) sort all links in \( O(i) \) according to the increasing order of \( t_{ij} + u_{ij}^d \) such that \( u_{i_1}^d + t_{i_1} \leq u_{i_2}^d + t_{i_2} \leq \cdots \leq u_{i_n}^d + t_{i_n} \) (\( n = |O(i)| \)) and set \( M_r = \{(i, j_1), \ldots \} \); (ii) sequentially add \((i, j_i)\) into \( M_r \), and stop after link \((i, j_i)\) with unlimited capacity (i.e., \( \lambda_{ij} = +\infty \) is added; and (iii) update the weight \( \{e_{ij,s^*}\} \) by applying the loading procedure to node \( i \) (see Micro-Loading in \([\text{Marcotte et al.}, 2004]\)).

The implementation of flow shifting in hyperbush equilibrium is rendered much more complicated in CTAP by two issues: (i) the hyperpath cost function is neither continuous nor differentiable, and (ii) the weight vector is flow-dependent. Because of (i), a line search must be performed to find a step size. A bisection algorithm is proposed for this purpose, see Algorithm 5. In each iteration, a flow shift \( \Delta \) is first determined based on the current step size \((a + b)/z\), cf. line 6), and applied to the hyperlinks covered by the EPHS (cf. lines 8-10). To update the cost on the two hyper-segments based on the shifted flow, however, the weight vector \( e \) must also be updated as per the issue (ii) above. The question is how to maintain the feasibility of \( y \) (cf. Definition 3) when hyperlink flows and the weights are simultaneously changed. To this end, we check the violation of the flow conservation condition at node \( l \) on the hyperbush \( G^d \), defined as

\[
\phi^d_l = \left( \sum_{m \in l} \sum_{s' \in O_m, (m,l) \in M^*_l} y_{ms'}^d e_{ms'} - q_l^d \right) - \sum_{s \in O_l \cap S^d} y_{ls}^d \forall l \in N, d \in D. \tag{40}
\]

Then, the flow on each outgoing hyperlink of \( l \) is proportionally adjusted by

\[
g_{ls}^d = y_{ls}^d + \frac{y_{ls}^d \phi_{ls}^d}{\sum_{s \in O_l \cap S^d} y_{ls}^d} \phi_{ls}^d, \quad \forall s \in O_l \cap S^d, l \in N \tag{41}
\]
to balance out the deficit or surplus caused by the flow update (cf. line 11). The flow-dependency of $e$ implies that the maximum allowed flow $\chi$ may also be affected by flow shifting. In some cases, especially when the step size is large, this interaction could produce negative hyperlink flow, violating another feasibility constraint. Because determining a priori the “correct” $\chi$ while anticipating this complex interaction is very difficult, we opt to implement a heuristic bypass. Specifically, when a negative flow is detected, we simply abort the current flow update, return to square one and restart with a smaller step size (cf. lines 12-14). Note that the maximum number of iterations $\bar{\varsigma}$, the level of precision $\bar{\epsilon}$ and the scaling coefficient $\bar{\eta}$ are set by default to 30, $10^{-10}$ and 2.0 respectively.

**Algorithm 5** Bisection algorithm for flow shifting between an extreme pair of hyper-segment

1: **Input:** $G^d$, $E_i((\overline{\bar{k}}, \overline{\bar{k}}))$, $y$, $e$, maximum allowed flow $\chi$, parameters $\bar{\varsigma}$, $\bar{\epsilon}$ and $\bar{\eta}$.
2: **Initialization:**
3: Set $a = 0$, $b = 1$, $z = 2$, and loop counter $\iota = 1$.
4: Initialize $y' = y$, $e' = e$, and NA = true.
5: while $\iota < \bar{\varsigma}$ and $\chi \cdot (b - a) \geq \bar{\epsilon}$ and NA is true do
6: Set $\Delta = \chi \cdot (a + b) / z$, and $\iota = \iota + 1$.
7: for $l \in N$ in an increasing topological order do
8: if $l \in N_k \cup N_k^d$ then
9: Update $\tilde{y}^d_{l,s}$ and $\tilde{y}_l^d$ by Eq. (37).
10: end if
11: if $\tilde{y}^d_{l,s} < 0$, $\exists s' = l$, $s \in S_d$ then
12: Set $\tilde{y} = y'$, $\tilde{e} = e'$, and $z = \bar{\eta} * z$.
13: NA = false.
14: else
15: Update $\tilde{e}_{lm,s}$, $\forall s' = l$, $s \in S_d$, $d \in D$ by the loading procedure.
16: end if
17: end for
18: if NA is true then
19: Update $C_{\overline{\bar{k}}}$ and $C_{\overline{\bar{k}}}$ using Eq. (8).
20: if $C_{\overline{\bar{k}}} > C_{\overline{\bar{k}}}$ then
21: Set $a = (a + b) / z$.
22: else
23: Set $b = (a + b) / z$.
24: end if
25: end if
26: end while
27: Set $y = \tilde{y}$, $e = \tilde{e}$.
28: **Output:** Updated $y$ and $e$.

Finally, the following relative gap function is used to measure the convergence of the hyper-
bush algorithm for CTAP (see Marcotte et al., 2004),

\[ RG = 1 - \frac{\sum_{(r,d) \in W} u_r^d q_r}{\sum_{(i,j) \in A} t_{ij} v_{ij}} \]  \hspace{0.5cm} (42)

where \( \sum_{(i,j) \in A} t_{ij} v_{ij} \) is the total link travel cost, and \( u_r^d \) is the minimum travel cost between O-D pair \((r,d)\), all evaluated based on the current flow pattern.

7 Numerical experiments

In this section, we solve both CTAP and TEAP using HBA, and compare the computational performance of the new algorithm with that of the respective benchmarks. Both HBAs are coded based on the TNM package, a C++ class library specialized in modeling transportation networks (Nie, 2006). For CTAP, the results of the benchmark algorithms are directly taken from the literature, and the focus is on the level of solution precision achieved. For TEAP, all benchmark algorithms are also coded in TNM to ensure the implementations share codes wherever possible. This provides a basis for a fair comparison of the computational performance.

All numerical results reported in this section were produced on a Windows XP X64 Workstation with Intel Xeon CPU E3-1225 V3 3.3 GHz CPU and 32G RAM. In what follows, Sections 7.1 and 7.2 report test results for CTAP and TEAP respectively.

7.1 CTAP

For CTAP, we compare HBA with three projection-based algorithms employed by Marcotte et al. (2004), including the projection algorithm, the Konnov algorithm and the extragradient algorithm. All algorithms are tested on a small toy network (see Figure 7) and an “acyclic” version of the celebrated Sioux-Falls network (see Figure 8). As shown in Figure 7, each link in the network has a cost, and its limited capacity—if there is one—is reported in a bracket. The toy network consists of 6 nodes, 9 links and 2 O-D pairs, where O-D pair \((1,6)\) and \((2,6)\) both have a demand of 10 units. The Sioux-Falls network has 24 nodes, 42 links and 4 O-D pairs, where O-D pair \((1,24)\), \((1,22)\), \((7,24)\) and \((7,22)\) each has a demand of 35, 25, 20 and 20 units, respectively.

In both cases, HBA produced assignment results that are identical to those reported in Marcotte et al. (2004). For the toy network, the equilibrium travel costs obtained by HBA for O-D pair \((1,6)\) and \((2,6)\) are 60.0 and 55.0 respectively, which agree with Table 13 in Marcotte et al. (2004). Similarly, HBA replicates the equilibrium O-D costs reported in Table 21 in Marcotte et al. (2004), namely 120.0 for \((1,24)\), 140.0 for \((1,22)\), 111.8 for \((7,24)\) and 100.0 for \((7,22)\).

Table 1 compares the relative gap (Eq. (42)) achieved for the toy network by the four algorithms in the first 5 iterations. Whereas the best relative gap obtained by any benchmark
algorithm is only $0.005$, HBA attains a relative gap of about $10^{-13}$, closer to the limit allowed by computation in double-precision floating-point format. Moreover, HBA reaches near $10^{-12}$ after the first main iteration. In our experiments, up to 20 inner iterations can be performed in HBA to equilibrate flows after each hyperbush expansion. In the three projection algorithms, each iteration contains five column generation steps (to enrich the used hyperpath set) and 100 projection operations (to equilibrate flows between hyperpaths), according to Marcotte et al. (2004).

The results on the Sioux-Falls network are reported in Table 2 which tells a story rather similar to Table 1. HBA is able to reduce the relative gap below $10^{-12}$ after 15 iterations, while its competitors lag far behind: the best benchmark performance is achieved by Konnov, at 0.000002 at 20 iterations.

It seems that the reason why HBA converges so much faster for CTAP is its exploitation of acyclicity. Evidently, shifting flows between EPHS, which is possible only in acyclic networks, provides a sharper descent direction toward the equilibrium than the benchmarks, which are all based on the idea of projection in the space of hyperpath flows.

While the above comparison shows HBA can obtain much more precise CTAP solutions with the same number of iterations, it cannot confirm the algorithm holds a computational advantage, because of the difficulty to gauge the amount of work involved in one iteration across different algorithms. We decide not to wrestle with the question of computation efficiency here for two reasons. First, CTAP examples that are large enough to produce a meaningful difference in computation time are not readily available. Second, we do not have access to the code of the three benchmark algorithms. Instead, we will focus on computational performance in the experiments with TEAP, for which we have both the networks and the code.

Table 1: Convergence performance of the four algorithms for solving CTAP: the toy network

<table>
<thead>
<tr>
<th>Iteration</th>
<th>HBA</th>
<th>Projection</th>
<th>Konnov</th>
<th>ExtraGradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$3.5 \times 10^{-1}$</td>
<td>$3.5 \times 10^{-1}$</td>
<td>$2.8 \times 10^{-1}$</td>
<td>$2.2 \times 10^{-1}$</td>
</tr>
<tr>
<td>1</td>
<td>$1.8 \times 10^{-12}$</td>
<td>$7.6 \times 10^{-2}$</td>
<td>$2.0 \times 10^{-1}$</td>
<td>$1.8 \times 10^{-1}$</td>
</tr>
<tr>
<td>2</td>
<td>$1.6 \times 10^{-12}$</td>
<td>$4.6 \times 10^{-2}$</td>
<td>$8.6 \times 10^{-2}$</td>
<td>$1.5 \times 10^{-1}$</td>
</tr>
<tr>
<td>3</td>
<td>$4.2 \times 10^{-13}$</td>
<td>$2.8 \times 10^{-2}$</td>
<td>$5.2 \times 10^{-2}$</td>
<td>$1.2 \times 10^{-1}$</td>
</tr>
<tr>
<td>5</td>
<td>$1.8 \times 10^{-13}$</td>
<td>$5.1 \times 10^{-3}$</td>
<td>$1.6 \times 10^{-2}$</td>
<td>$6.3 \times 10^{-2}$</td>
</tr>
</tbody>
</table>
Table 2: Convergence performance of the four algorithms for solving CTAP: the Sioux-Falls network.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>HBA</th>
<th>Projection</th>
<th>Konnov</th>
<th>Extragradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2.2 \times 10^{-1}$</td>
<td>$1.4 \times 10^{-2}$</td>
<td>$1.5 \times 10^{-2}$</td>
<td>$1.4 \times 10^{-2}$</td>
</tr>
<tr>
<td>5</td>
<td>$2.0 \times 10^{-3}$</td>
<td>$1.4 \times 10^{-2}$</td>
<td>$1.3 \times 10^{-3}$</td>
<td>$1.5 \times 10^{-2}$</td>
</tr>
<tr>
<td>10</td>
<td>$9.7 \times 10^{-5}$</td>
<td>$6.9 \times 10^{-3}$</td>
<td>$9.9 \times 10^{-4}$</td>
<td>$1.3 \times 10^{-2}$</td>
</tr>
<tr>
<td>15</td>
<td>$9.3 \times 10^{-13}$</td>
<td>$4.1 \times 10^{-4}$</td>
<td>$4.2 \times 10^{-5}$</td>
<td>$6.3 \times 10^{-3}$</td>
</tr>
<tr>
<td>20</td>
<td>$1.9 \times 10^{-13}$</td>
<td>$4.0 \times 10^{-5}$</td>
<td>$2.0 \times 10^{-6}$</td>
<td>$8.8 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

7.2 TEAP

In this section, we compare HBA with three benchmark TEAP algorithms. These include the Generalized Frank-Wolfe (GFW) algorithm (Spiess, 1993), and two state-of-the-art hyperpath-based algorithms: the Greedy with adaptive inner looping (Greedy-AIL) algorithm and the gradient projection with adaptive inner looping (GP-AIL) algorithm (see Xu et al., 2020a). Developed and tested in Xu et al. (2020a), the latter two were found to outperform other known TEAP solution algorithms by a wide margin. Note that the functional forms of $t$, $e$ and $\omega$ take formulas in Section 3.3.2 (c.f Equation (12)-(17)). Two batches of experiments are conducted. The first focuses on four benchmark networks from the literature (Section 7.2.1), and the second performs a sensitivity analysis using randomly generated grid networks (Section 7.2.2).

7.2.1 Benchmark networks

The four algorithms are tested over two small networks from the literature, and two large real-world transit networks. Table 3 reports the details of the four networks, and Figure 9 shows their topology. The parameters used to specify the cost functions on various types of links in each network are adopted from Table 1 in Xu et al. (2020a).

Table 3: Details of tested transit networks for TEAP.

<table>
<thead>
<tr>
<th>Scale</th>
<th>Networks</th>
<th>Stops</th>
<th>Lines</th>
<th>Nodes</th>
<th>Links</th>
<th>O-D pairs</th>
<th>Trips</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>Wu-Small</td>
<td>8</td>
<td>6</td>
<td>26</td>
<td>44</td>
<td>4</td>
<td>599</td>
</tr>
<tr>
<td></td>
<td>Sioux-Falls</td>
<td>24</td>
<td>10</td>
<td>84</td>
<td>150</td>
<td>16</td>
<td>7,200</td>
</tr>
<tr>
<td>Large</td>
<td>Winnipeg</td>
<td>690</td>
<td>132</td>
<td>5,186</td>
<td>13,276</td>
<td>5,332</td>
<td>18,166</td>
</tr>
<tr>
<td></td>
<td>Shenzhen</td>
<td>1,736</td>
<td>245</td>
<td>7,513</td>
<td>22,786</td>
<td>60,248</td>
<td>141,050</td>
</tr>
</tbody>
</table>
Figure 9: Topology of the tested transit networks for TEAP.

Originally tested in Wu et al. (1994), the Wu-Small network has a total of three routes and six transit lines. The Sioux-Falls transit network is adopted from Szeto and Jiang (2014), which consists of ten transit lines (illustrated in different color and arrow type in Figure 9(b)). The Shenzhen urban transit network is provided by Shenzhen Bus Authority, and the Winnipeg transit network comes with an Emme3 demo project (INRO, 2018). The O-D matrix in Shenzhen network is obtained from smart-card data collected during three weekdays in 2016.

Figure 10 compares the convergence performance of the four algorithms in solving the two small networks. Here the convergence criterion is set as $10^{-12}$. That is, the execution of an algorithm is terminated if it reduces the relative gap below that threshold. In both cases, GFW failed to reach the threshold. Instead, it is stuck stubbornly before even achieving $10^{-4}$. This sluggish behavior of Frank-Wolfe type algorithms, of course, is well known. The other three, HBA, Greedy-AIL and GP-AIL, all quickly converge below the threshold. To solve TEAP on Wu-Small, the time required for convergence is negligible for all three algorithms. For Sioux-Falls, none of the three algorithms requires more than 0.5 seconds to converge. HBA is the fastest in all cases, although the computational advantage seems trivial in these small networks.

The superiority of HBA becomes much more prominent in the larger examples. Here, an
algorithm is terminated if either it reaches a relative gap of $10^{-12}$ or its execution time exceeds six hours. Figure 11 shows that neither GFW nor Greedy-AIL was able to reach the convergence target for either network. The best relative gap achieved by the former is around $10^{-4}$, and about $10^{-9}$ by the latter. As for GP-AIL, it consumed about two and four hours, respectively, to successfully reach the convergence criterion for the Winnipeg and the Shenzhen network. In contrast, HBA requires only 0.5 and 1.5 hours to converge for the two networks. In other words, HBA is able to reduce the computation time required by the fastest hyperpath algorithm by nearly two thirds. This is a remarkable speedup, especially considering both GP-AIL and Greedy-AIL are themselves state-of-the-art TEAP algorithms.

![Figure 10: Convergence performance for two small networks in TEAP.](image1)

![Figure 11: Convergence performance for two large networks in TEAP.](image2)

Finally, Table 4 shows that the memory (as reported by the operating system) used by HBA amounts to but a fraction of what is needed by both hyperpath-based algorithms.

<table>
<thead>
<tr>
<th>Networks</th>
<th>Greedy-AIL</th>
<th>GP-AIL</th>
<th>HBA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Winnipeg</td>
<td>1,826 MB</td>
<td>1,623 MB</td>
<td>369 MB</td>
</tr>
<tr>
<td>Shenzhen</td>
<td>14,482 MB</td>
<td>13,723 MB</td>
<td>2,819 MB</td>
</tr>
</tbody>
</table>
7.2.2 Sensitivity analysis

In this section, we conduct a sensitivity analysis using three sets of randomly generated grid transit networks (see Figure 6 in [Xu et al. 2020a]). Each transit stop has random travel demand to every other stop. Taking 5 by 5 grid network (25-stops) for example, the number of O-D pairs is $25 \times 24 = 600$. Set 1 includes five grid networks whose size increases from 5 by 5 to 25 by 25, and the travel demand is set such that the congestion level in each network is comparable. The five networks in Set 2 are 15 by 15 grid, with the number of parallel transit lines varying from 2 to 6 between any two adjacent stops. Set 3 consists of five 20 by 20 grid networks, with O-D demands uniformly scaled by a different factor (0.50, 0.75, 1.00, 1.25 and 1.50). For convenience, the fifteen grid networks used in these experiments are named from GridNet #1 to #15, as shown in Table 5. Given the size of these networks, the maximum running time is set to one hour.

<table>
<thead>
<tr>
<th>Name</th>
<th>Stops</th>
<th>Parallel lines</th>
<th>Lines</th>
<th>Nodes</th>
<th>Links</th>
<th>OD pairs</th>
<th>Trips</th>
</tr>
</thead>
<tbody>
<tr>
<td>GridNet #1</td>
<td>25</td>
<td>2</td>
<td>40</td>
<td>225</td>
<td>480</td>
<td>600</td>
<td>11,728</td>
</tr>
<tr>
<td>GridNet #2</td>
<td>100</td>
<td>2</td>
<td>80</td>
<td>900</td>
<td>2,160</td>
<td>9,900</td>
<td>39,606</td>
</tr>
<tr>
<td>GridNet #3</td>
<td>225</td>
<td>2</td>
<td>120</td>
<td>2,025</td>
<td>5,040</td>
<td>50,400</td>
<td>137,116</td>
</tr>
<tr>
<td>GridNet #4</td>
<td>400</td>
<td>2</td>
<td>160</td>
<td>3,600</td>
<td>9,120</td>
<td>159,600</td>
<td>329,165</td>
</tr>
<tr>
<td>GridNet #5</td>
<td>625</td>
<td>2</td>
<td>200</td>
<td>5,625</td>
<td>14,400</td>
<td>390,000</td>
<td>591,372</td>
</tr>
<tr>
<td>GridNet #6</td>
<td>225</td>
<td>2</td>
<td>120</td>
<td>2,025</td>
<td>5,040</td>
<td>50,400</td>
<td>137,116</td>
</tr>
<tr>
<td>GridNet #7</td>
<td>225</td>
<td>3</td>
<td>180</td>
<td>2,925</td>
<td>7,560</td>
<td>50,400</td>
<td>137,116</td>
</tr>
<tr>
<td>GridNet #8</td>
<td>225</td>
<td>4</td>
<td>240</td>
<td>3,825</td>
<td>10,080</td>
<td>50,400</td>
<td>137,116</td>
</tr>
<tr>
<td>GridNet #9</td>
<td>225</td>
<td>5</td>
<td>300</td>
<td>4,725</td>
<td>12,600</td>
<td>50,400</td>
<td>137,116</td>
</tr>
<tr>
<td>GridNet #10</td>
<td>225</td>
<td>6</td>
<td>360</td>
<td>5,625</td>
<td>15,120</td>
<td>50,400</td>
<td>137,116</td>
</tr>
<tr>
<td>GridNet #11</td>
<td>400</td>
<td>2</td>
<td>160</td>
<td>3,600</td>
<td>9,120</td>
<td>159,600</td>
<td>164,583</td>
</tr>
<tr>
<td>GridNet #12</td>
<td>400</td>
<td>2</td>
<td>160</td>
<td>3,600</td>
<td>9,120</td>
<td>159,600</td>
<td>246,874</td>
</tr>
<tr>
<td>GridNet #13</td>
<td>400</td>
<td>2</td>
<td>160</td>
<td>3,600</td>
<td>9,120</td>
<td>159,600</td>
<td>329,165</td>
</tr>
<tr>
<td>GridNet #14</td>
<td>400</td>
<td>2</td>
<td>160</td>
<td>3,600</td>
<td>9,120</td>
<td>159,600</td>
<td>411,456</td>
</tr>
<tr>
<td>GridNet #15</td>
<td>400</td>
<td>2</td>
<td>160</td>
<td>3,600</td>
<td>9,120</td>
<td>159,600</td>
<td>493,748</td>
</tr>
</tbody>
</table>

Table 6 reports the CPU times required by different algorithms to achieve various levels of precision. HBA is the front runner in all but the lowest convergence target ($10^{-2}$), followed by GP-AIL, Greedy-AIL and GFW. For larger networks and tighter convergence targets, it outperforms the two hyperpath algorithms by a factor of 2 to 5 in terms of CPU time. Moreover, the larger the network and the tighter the convergence target, the greater the lead enjoyed by HBA. For the largest network (GridNet #5), neither of the two hyperpath algorithms were able to reach $10^{-6}$ within one hour. As the network size increases, all algorithms require more time to reach the same level of precision, and the rate of increase is evidently nonlinear. For example, as the number of stops increases from 225 to 625 (a factor of about 2.8), the CPU time required to achieve a relative gap of $10^{-4}$ rises by a factor of 44 for Greedy-AIL, 47 for GP-AIL, and 24 for HBA. To reach a relative gap of $10^{-12}$, HBA requires about 40 times as much CPU time for GridNet #5 as for GridNet #3.

Table 7 reports the sensitivity of the algorithms’ performance to the number of parallel lines. It shows, again, HBA consistently outperforms all benchmark algorithms by a significant margin, except for the lowest convergence criterion. Moreover, HBA is less sensitive to the number of
Table 6: Sensitivity results for TEAP based on grid networks: network size.

<table>
<thead>
<tr>
<th>Network</th>
<th>Algorithm</th>
<th>Relative gap, CPU time</th>
<th>10^{-2}</th>
<th>10^{-4}</th>
<th>10^{-6}</th>
<th>10^{-8}</th>
<th>10^{-10}</th>
<th>10^{-12}</th>
</tr>
</thead>
<tbody>
<tr>
<td>GridNet #1</td>
<td>GFW</td>
<td></td>
<td>0.04s</td>
<td>0.07s</td>
<td>0.52s</td>
<td>8.80s</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Greedy-AIL</td>
<td>+</td>
<td>0.07s</td>
<td>0.28s</td>
<td>0.38s</td>
<td>0.49s</td>
<td>0.70s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GP-AIL</td>
<td>+</td>
<td>0.07s</td>
<td>0.23s</td>
<td>0.32s</td>
<td>+</td>
<td>0.41s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HBA</td>
<td>+</td>
<td>0.07s</td>
<td>0.15s</td>
<td>0.20s</td>
<td>+</td>
<td>0.24s</td>
<td></td>
</tr>
<tr>
<td>GridNet #2</td>
<td>GFW</td>
<td></td>
<td>0.78s</td>
<td>3.39s</td>
<td>12.02s</td>
<td>1.06m</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Greedy-AIL</td>
<td>+</td>
<td>1.62s</td>
<td>7.4s</td>
<td>10.26s</td>
<td>15.95s</td>
<td>21.79s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GP-AIL</td>
<td>+</td>
<td>1.63s</td>
<td>6.92s</td>
<td>9.56s</td>
<td>12.20s</td>
<td>14.93s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HBA</td>
<td>+</td>
<td>1.50s</td>
<td>3.89s</td>
<td>5.44s</td>
<td>6.17s</td>
<td>6.91s</td>
<td></td>
</tr>
<tr>
<td>GridNet #3</td>
<td>GFW</td>
<td></td>
<td>6.13s</td>
<td>1.73m</td>
<td>20.05m</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Greedy-AIL</td>
<td>+</td>
<td>11.91s</td>
<td>50.25s</td>
<td>1.60m</td>
<td>2.49m</td>
<td>4.69m</td>
<td>12.31m</td>
</tr>
<tr>
<td></td>
<td>GP-AIL</td>
<td>+</td>
<td>11.68s</td>
<td>48.06s</td>
<td>1.51m</td>
<td>2.81m</td>
<td>3.84m</td>
<td>4.40m</td>
</tr>
<tr>
<td></td>
<td>HBA</td>
<td>+</td>
<td>12.31s</td>
<td>22.02s</td>
<td>43.17s</td>
<td>1.15m</td>
<td>1.33m</td>
<td>1.50m</td>
</tr>
<tr>
<td>GridNet #4</td>
<td>GFW</td>
<td></td>
<td>34.86s</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Greedy-AIL</td>
<td>+</td>
<td>1.00m</td>
<td>6.59m</td>
<td>17.73m</td>
<td>30.19m</td>
<td>47.51m</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>GP-AIL</td>
<td>+</td>
<td>57.63s</td>
<td>5.04m</td>
<td>17.12m</td>
<td>28.42m</td>
<td>43.50m</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>HBA</td>
<td>+</td>
<td>56.51s</td>
<td>2.41m</td>
<td>5.36m</td>
<td>7.57m</td>
<td>8.68m</td>
<td>10.21m</td>
</tr>
<tr>
<td>GridNet #5</td>
<td>GFW</td>
<td></td>
<td>26.07m</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Greedy-AIL</td>
<td>+</td>
<td>5.11m</td>
<td>36.87m</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>GP-AIL</td>
<td>+</td>
<td>5.22m</td>
<td>38.04m</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>HBA</td>
<td>+</td>
<td>2.92m</td>
<td>8.71m</td>
<td>26.56m</td>
<td>43.85m</td>
<td>51.02m</td>
<td>59.21m</td>
</tr>
</tbody>
</table>

Note: s, seconds; m, minutes; +, when the required relative gap is skipped by the next higher resolution achievement; -, when fail to achieve the required relative gap.
parallel lines than the other algorithms. As the number of parallel lines increases from 2 to 6, the computation times required to achieve $10^{-12}$ by HBA increase by a factor of 4 (a nearly linear rate). For GP-AIL, the CPU time increases eight times, and Greedy-AIL fails to reach the highest level of precision within an hour.

Table 7: Sensitivity results for TEAP based on grid networks: number of parallel lines.

<table>
<thead>
<tr>
<th>Network</th>
<th>Algorithm</th>
<th>Relative gap, CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td>GridNet #6</td>
<td>GFW</td>
<td>6.13s</td>
</tr>
<tr>
<td></td>
<td>Greedy-AIL</td>
<td>11.91s</td>
</tr>
<tr>
<td></td>
<td>GP-AIL</td>
<td>11.68s</td>
</tr>
<tr>
<td></td>
<td>HBA</td>
<td>12.31s</td>
</tr>
<tr>
<td>GridNet #7</td>
<td>GFW</td>
<td>9.69s</td>
</tr>
<tr>
<td></td>
<td>Greedy-AIL</td>
<td>19.62s</td>
</tr>
<tr>
<td></td>
<td>GP-AIL</td>
<td>17.63s</td>
</tr>
<tr>
<td></td>
<td>HBA</td>
<td>19.69s</td>
</tr>
<tr>
<td>GridNet #8</td>
<td>GFW</td>
<td>15.74s</td>
</tr>
<tr>
<td></td>
<td>Greedy-AIL</td>
<td>28.47s</td>
</tr>
<tr>
<td></td>
<td>GP-AIL</td>
<td>27.99s</td>
</tr>
<tr>
<td></td>
<td>HBA</td>
<td>30.08s</td>
</tr>
<tr>
<td>GridNet #9</td>
<td>GFW</td>
<td>1.04m</td>
</tr>
<tr>
<td></td>
<td>Greedy-AIL</td>
<td>51.57s</td>
</tr>
<tr>
<td></td>
<td>GP-AIL</td>
<td>49.51s</td>
</tr>
<tr>
<td></td>
<td>HBA</td>
<td>40.31s</td>
</tr>
<tr>
<td>GridNet #10</td>
<td>GFW</td>
<td>1.28m</td>
</tr>
<tr>
<td></td>
<td>Greedy-AIL</td>
<td>1.06m</td>
</tr>
<tr>
<td></td>
<td>GP-AIL</td>
<td>1.14m</td>
</tr>
<tr>
<td></td>
<td>HBA</td>
<td>59.34s</td>
</tr>
</tbody>
</table>

Table 8 confirms again the computational superiority of HBA. What is noteworthy here is how much more sensitive the hyperpath algorithms are to congestion level than HBA. Take the convergence criterion $10^{-10}$ for example. When the demand scalar increases from 0.5 (GridNet #11) to 1.0 (GridNet #13), the CPU time required by GP-AIL grows from 16.6 minutes to 43.5 (a factor of 2.6), whereas that required by HBA grows from 4.47 to 8.68 (a factor of 1.9). For the most congested case (scalar = 1.5), no algorithm except HBA was able to reach $10^{-8}$ within an hour. As a comparison, HBA can reach $10^{-12}$ in less than 17 minutes.
Table 8: Sensitivity results for TEAP based on grid networks: congestion level.

<table>
<thead>
<tr>
<th>Network</th>
<th>Algorithm</th>
<th>Relative gap, CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>10^{-2}</td>
</tr>
<tr>
<td>GridNet #11</td>
<td>GFW</td>
<td>31.11s</td>
</tr>
<tr>
<td></td>
<td>Greedy-AI</td>
<td>46.64s</td>
</tr>
<tr>
<td></td>
<td>GP-AI</td>
<td>45.20s</td>
</tr>
<tr>
<td></td>
<td>HBA</td>
<td>43.71s</td>
</tr>
<tr>
<td>GridNet #12</td>
<td>GFW</td>
<td>33.66s</td>
</tr>
<tr>
<td></td>
<td>Greedy-AI</td>
<td>57.21s</td>
</tr>
<tr>
<td></td>
<td>GP-AI</td>
<td>55.80s</td>
</tr>
<tr>
<td></td>
<td>HBA</td>
<td>52.44s</td>
</tr>
<tr>
<td>GridNet #13</td>
<td>GFW</td>
<td>34.86s</td>
</tr>
<tr>
<td></td>
<td>Greedy-AI</td>
<td>1.00m</td>
</tr>
<tr>
<td></td>
<td>GP-AI</td>
<td>57.63s</td>
</tr>
<tr>
<td></td>
<td>HBA</td>
<td>56.51s</td>
</tr>
<tr>
<td>GridNet #14</td>
<td>GFW</td>
<td>1.08m</td>
</tr>
<tr>
<td></td>
<td>Greedy-AI</td>
<td>58.84s</td>
</tr>
<tr>
<td></td>
<td>GP-AI</td>
<td>59.16s</td>
</tr>
<tr>
<td></td>
<td>HBA</td>
<td>48.22s</td>
</tr>
<tr>
<td>GridNet #15</td>
<td>GFW</td>
<td>1.76m</td>
</tr>
<tr>
<td></td>
<td>Greedy-AI</td>
<td>1.12m</td>
</tr>
<tr>
<td></td>
<td>GP-AI</td>
<td>1.03m</td>
</tr>
<tr>
<td></td>
<td>HBA</td>
<td>54.38s</td>
</tr>
</tbody>
</table>

8 Conclusions

We proposed a unified modeling framework for a class of strategy-based equilibrium traffic assignment (SETA) problems, which is well suited to representing a much richer set of travel choices than what is available in conventional standard traffic assignment problems. We also developed a generic hyperbush algorithm (HBA) based on the idea of decomposing a hypergraph into destination-based hyperbushes. By constructing hyperbushes and limiting traffic assignment to them, HBA promises to obtain more precise solutions at a lower computational cost, both in terms of CPU time and memory consumption. The results from the numerical experiments show that HBA holds a substantial advantage against the state-of-the-art, hyperpath-based algorithms in terms of computational efficiency. For larger, denser and more congested TEAP instances, HBA gains a speedup of up to 5 times over the best competitor. This encouraging performance warrants remarks in lieu of the recent development in the traffic assignment literature.

For the standard TAP, recent studies suggest that properly implemented path-based algorithms could rival, if not always outperform, the most efficient bush-based algorithms (e.g., Algorithm B, TAPAS and iTAPAS). For SETA problems studied herein, however, the hyperpath-based algorithms, despite adopting the strategies that had successfully boosted the performance of the path-based algorithms for TAP, seem still no match for the hyperbush algorithm.

We believe the success of HBA can be attributed to two reasons. First, finding optimal hyper-
paths on a hypergraph is more expensive than finding a simple path on a regular graph, largely
due to the extra time required to compute the optimal strategy at each node. By focusing on
minimum- and maximum-cost hyperpath in a hyperbush, therefore, HBA gains a greater com-
putational advantage over hyperpath-based algorithms than bush-based algorithms over their
path-based counterparts. In the Shenzhen network, it takes about 12 seconds to find shortest
hyperpaths for all the O-D pairs using the label setting algorithm presented in Li et al. (2015). In
contrast, finding all minimum- and maximum-cost hyperpaths on destination-based hyper-
bushes requires less than 3 seconds, a speedup of four times. Second, HBA is much less memory
intensive than hyperpath-based algorithms, because it does not bear the burden to store individ-
ual hyperpaths. This saving not only means HBA can solve much larger problems with the same
computational resources, but also brings direct computational benefits by reducing the amount
of data that must be held in RAM.

We did not address the issue of convergence for the generic HBA. The conventional proofs
for bush-based algorithms (e.g. Bar-Gera, 2002; Dial, 2006) may encounter difficulties in dealing
with SETA problems, because of the complex interactions between hyperpath flows, and the fact
that these problems can no longer be formulated as minimizing a closed-form potential function.
Indeed, neither Marcotte and Nguyen (1998) nor Marcotte et al. (2004) offered convergence results
for their CTAP algorithm. Even though the numerical results reported herein suggest HBA
consistently converges to high-quality solutions for both TEAP and CTAP (as measured by the
relative gap), it should still be considered a heuristic until a rigorous convergence proof can be
provided. In any case, establishing the convergence result for HBA may depend on the structure
and properties of the cost functions, which vary significantly from one problem to another. The
key may lie in careful exploitation of these properties. We leave a thorough investigation of the
convergence issue to a future study.

Another possible direction for future research is to tailor the methodological tools developed
in this paper, both the model and the solution algorithm, to other SETA problems. Examples in-
clude but are not limited to: variants of TEAP that consider bus bunching and/or seat restriction
effects, OFEX assignment problems, and MDP-based assignment problems.

Acknowledgments

This research is jointly funded by National Nature Science Foundation of China (Grant No.
71971178), Sichuan Science and Technology Program (Grant No. 2020010, 2021YFH0041), Chengdu
Science and Technology Program (2021-YF05-00177-SN). The effort of Yu (Marco) Nie was par-
tially supported by the US National Science Foundation under the award number CMMI 1922665.

References

Allen, G., 1977. Variational inequalities, complementarity problems, and duality theorems. Jour-
nal of Mathematical Analysis and Applications 58, 1 – 10. doi:https://doi.org/10.1016/
0022-247X(77)90222-0

Ausiello, G., Italiano, G.F., Nanni, U., 1998. Hypergraph traversal revisited: Cost measures and
dynamic algorithms, in: Brim, L., Gruska, J., Zlatuška, J. (Eds.), Mathematical Foundations of


42


45
## Appendix A  Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sets:</strong></td>
<td></td>
</tr>
<tr>
<td>$N$</td>
<td>set of nodes, indexed by $i$, $j$, $l$, $m$, or $n$;</td>
</tr>
<tr>
<td>$A$</td>
<td>set of links, indexed by $(i,j)$;</td>
</tr>
<tr>
<td>$R$</td>
<td>set of origins, indexed by $r$;</td>
</tr>
<tr>
<td>$D$</td>
<td>set of destinations, indexed by $d$;</td>
</tr>
<tr>
<td>$R(d)$</td>
<td>set of origins destined for destination $d$;</td>
</tr>
<tr>
<td>$I(i)$</td>
<td>set of incoming links associated with node $i$.</td>
</tr>
<tr>
<td>$O(i)$</td>
<td>set of outgoing links associated with node $i$;</td>
</tr>
<tr>
<td>$I^-(i)$</td>
<td>set of tail nodes corresponding to links in $I(i)$;</td>
</tr>
<tr>
<td>$O^+(i)$</td>
<td>set of head nodes corresponding to links in $O(i)$.</td>
</tr>
<tr>
<td>$S$</td>
<td>set of hyperlinks, indexed by $s$;</td>
</tr>
<tr>
<td>$O(i)$</td>
<td>set of outgoing hyperlinks associated with node $i$;</td>
</tr>
<tr>
<td>$J_s$</td>
<td>set of head nodes of $s$ ($J_s \subseteq O^+(i)$, where $s^- = i$ denotes the tail node);</td>
</tr>
<tr>
<td>$M_s$</td>
<td>set of regular links corresponding to $s$ ($M_s = {(s^-, j) \mid \forall j \in J_s \subseteq A}$;</td>
</tr>
<tr>
<td>$K_{rd}$</td>
<td>set of hyperpath for O-D pair $(r,d)$, indexed by $k$;</td>
</tr>
<tr>
<td>$N_k$</td>
<td>set of nodes corresponding to $k$;</td>
</tr>
<tr>
<td>$S_k$</td>
<td>set of hyperlinks corresponding to $k$;</td>
</tr>
<tr>
<td>$M_k$</td>
<td>set of regular link corresponding to $k$;</td>
</tr>
<tr>
<td>$S^d$</td>
<td>set of hyperlinks on the hyperbush $G^d$ corresponding to destination $d$;</td>
</tr>
<tr>
<td>$A^d$</td>
<td>set of all links included in $G^d$ ($A^d = \cup_{s \in S^d} M_s \subseteq A$);</td>
</tr>
<tr>
<td><strong>Parameters:</strong></td>
<td></td>
</tr>
<tr>
<td>$q_{rd}$</td>
<td>travel demand for OD pair $(r,d)$;</td>
</tr>
<tr>
<td>$f_{ij}$</td>
<td>mean frequency corresponding to link $(i,j)$ in TEAP;</td>
</tr>
<tr>
<td>$\lambda_{ij}$</td>
<td>limited capacity corresponding to link $(i,j)$ in CTAP;</td>
</tr>
<tr>
<td><strong>Variables:</strong></td>
<td></td>
</tr>
<tr>
<td>$h^d_{rk}$</td>
<td>flow assigned to hyperpath $k$ for O-D pair $(r,d)$;</td>
</tr>
<tr>
<td>$C^d_k$</td>
<td>cost of using hyperpath $k$;</td>
</tr>
<tr>
<td>$y^d_{is}$</td>
<td>flow leaving from node $i$ to $d$ via $s$;</td>
</tr>
<tr>
<td>$u^d_i$</td>
<td>weighted minimum cost from $i$ to $d$;</td>
</tr>
<tr>
<td>$U^d_{is}$</td>
<td>weighted hyperlink cost from $i$ to $d$ via $s$;</td>
</tr>
<tr>
<td>$U^d_i$</td>
<td>weighted maximum cost from $i$ to $d$;</td>
</tr>
<tr>
<td>$\tilde{U}^d_i$</td>
<td>weighted maximum dummy cost from $i$ to $d$;</td>
</tr>
<tr>
<td>$v_{ij}$</td>
<td>flow on link $(i,j)$;</td>
</tr>
<tr>
<td>$l_{ij}$</td>
<td>cost on link $(i,j)$;</td>
</tr>
<tr>
<td>$e_{ij,s}$</td>
<td>conditional probability choosing link $(i,j)$ via $s$;</td>
</tr>
<tr>
<td>$\omega^s_i$</td>
<td>cost passing through node $i$ via $s$;</td>
</tr>
<tr>
<td>$\pi^k_{ij}$</td>
<td>probability traverse through link $(i,j)$ on hyperpath $k$;</td>
</tr>
<tr>
<td>$\varphi^k_i$</td>
<td>probability traverse through node $i$ on hyperpath $k$.</td>
</tr>
</tbody>
</table>
Appendix B  An example for a cyclic hyperpath

To illustrate an optimal hyperpath that contains cycles, consider a small hypergraph shown in Figure 12. The hypergraph consists of six hyperlinks, in which hyperlink $s_1$, $s_2$ and $s_3$ make up the optimal hyperpath $k^*$ from the origin node 1 to destination node 4. The weight vector $e = \{e_{12,s_1}, e_{13,s_1}, e_{24,s_2}, e_{34,s_3}, e_{13,s_3}, e_{12,s_3}\} = \{0.5, 0.5, 1, 1, 1, 1\}$, the link cost vector $t = \{t_{12}, t_{13}, t_{31}, t_{24}, t_{34}\} = \{1, 1, 3, 12, 22\}$ and the node cost vector $\omega = \{\omega_{1,s_1}, \omega_{1,s_5}, \omega_{1,s_6}, \omega_{2,s_2}, \omega_{3,s_3}, \omega_{3,s_4}\} = \{0, 100, 100, 0, 0, 0\}$. The minimum hyperpath cost from each node $i$ to destination node 4 converges to a value given in the bracket. To show the iterative process, the weighted minimum cost for each node $i$, $4$ is initialized as $+\infty$, i.e. $u^4_i = +\infty$, and $u^4_4 = 0$. The optimal hyperlink is initialized as an empty set. We note that the optimal hyperlink for node 2 must be $s_2$ because it is the only choice, leading to $u^4_4 = \omega_{2,s_2} + e_{24,s_2}(u^4_4 + t_{24}) = 12$. In each iteration, the algorithm finds the optimal hyperlink and updates corresponding weighted minimum cost for node 1 and 3 once at a time. For example, at the first iteration, $u^4_1 = \omega_{3,s_3} + e_{34,s_4}(u^4_4 + t_{34}) = 22$ and $u^4_1 = \omega_{1,s_1} + e_{12,s_1}(u^4_4 + t_{12}) + e_{13,s_1}(u^4_3 + t_{13}) = 18$. The results are summarized in Table 9. We can observe at the second iteration, the optimal hyperlink for node 3 has changed from $s_4$ to $s_3$ because $21 < 22$. From there on, the algorithm enters a cycle that iteratively reduce the $u^4_1$ and $u^4_3$ until those values converge to the equilibrium state. Clearly, a cycle is identified between node 1 and 3 that consists of $s_1$ and $s_3$.

Here, agents that follow the optimal hyperpath will first have a 50% probability to go to node 2, and a 50% probability to go to node 3. In the first case, once agents arrive at node 2, they will continue to node 4; in the latter case, they will return to node 1 after they visit node 3. Once agents returns to node 1, they again face the same choice. Thus, they have a 25% chance to chose node 3 (hence return to node 1) the second time, and a 12.5% chance to cycle the third time. It is easy to show the cumulative probability of choosing all the cycles adds to 1.0 ($0.5 + 0.25 + \ldots$). Thus, we can simplify the process as follows: 50% of agents choose node 3, return to node 1, and then they all choose node 2 to arrive at the destination. The total system cost would be exactly the same as the above case of infinite cycles.

Table 9: Results obtained in the iterative process for finding the optimal hyperpath between node 1 and node 4.

<table>
<thead>
<tr>
<th>iteration</th>
<th>$u^4_1$</th>
<th>$s^*_1$</th>
<th>$u^4_3$</th>
<th>$s^*_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$+\infty$</td>
<td>$\emptyset$</td>
<td>$+\infty$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>1</td>
<td>18</td>
<td>$s_1$</td>
<td>22</td>
<td>$s_4$</td>
</tr>
<tr>
<td>2</td>
<td>17.5</td>
<td>$s_1$</td>
<td>21</td>
<td>$s_3$</td>
</tr>
<tr>
<td>3</td>
<td>17.25</td>
<td>$s_1$</td>
<td>20.5</td>
<td>$s_3$</td>
</tr>
<tr>
<td>4</td>
<td>17.125</td>
<td>$s_1$</td>
<td>20.25</td>
<td>$s_3$</td>
</tr>
<tr>
<td>5</td>
<td>17.03125</td>
<td>$s_1$</td>
<td>20.125</td>
<td>$s_3$</td>
</tr>
<tr>
<td>6</td>
<td>17.01563</td>
<td>$s_1$</td>
<td>20.0625</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\infty$</td>
<td>17</td>
<td>$s_1$</td>
<td>20</td>
<td>$s_3$</td>
</tr>
</tbody>
</table>
Appendix C  Proof for Lemma 1

We shall prove that \( y^d_{i,s}(u^d_{i,s} - u^d_i) = 0, \forall i, s, d \) implies that \( h_k(C^d_{k} - u^d_i) = 0, \forall k, r, d \), and vice versa. Note that the above condition is obviously satisfied for zero-flow hyperlinks or hyperpaths, and therefore we will focus on hyperlinks and hyperpaths that carry positive flow:

(1) We first show that \( y^d_{i,s}(u^d_{i,s} - u^d_i) = 0, \forall i, s, d \) implies \( h_k(C^d_{k} - u^d_i) = 0, \forall k, r, d \).

Consider a destination-based sub-hypergraph \( G(N, S^d) \) such that for each \( s \in S^d, y^d_{i,s} > 0 \) and \( y^d_{i,s}(u^d_{i,s} - u^d_i) = 0 \). Thus, \( u^d_{i,s} = u^d_i, \forall i \in N, s \in S^d \). Let \( k = \{ r = i_1, i_2, \ldots, i_m, s_1, s_2, \ldots, s_{n-1} \} \) represent any hyperpath connecting \( r \) and \( d \) in the sub-hypergraph. Hence, we have \( N_k = \{ r, i_2, \ldots, i_{n-1}, d \} \) and \( S_k = \{ s_1, s_2, \ldots, s_{n-1} \} \). \( C^d_{k} \) can be recursively calculated by a backward pass of the nodes in \( N_k \) from \( d \) to \( r \) as follows (see Eq. (8)):

\[
C^d_{i_{n-1}d} = \omega_{i_{n-1}s_{n-1}} + \sum_{(i_{n-1},j) \in M_{i_{n-1}}} e_{i_{n-1}j,s_{n-1}} (t_{i_{n-1}j} + 0),
\]

\[
C^d_{i_{n-2}d} = \omega_{i_{n-2}s_{n-2}} + \sum_{(i_{n-2},j) \in M_{i_{n-2}}} e_{i_{n-2}j,s_{n-2}} (t_{i_{n-2}j} + C^d_{j}),
\]

\[
\ldots
\]

\[
C^d_{i_2d} = \omega_{i_2s_2} + \sum_{(i_2,j) \in M_{i_2}} e_{i_2j,s_2} (t_{i_2j} + C^d_{j}),
\]

\[
C^d_{i_1d} = \omega_{i_1s_1} + \sum_{(i_1,j) \in M_{i_1}} e_{i_1j,s_1} (t_{i_1j} + C^d_{j}).
\]

Further note that for any \( s \in S_k \), we can compute its weighted hyperlink cost on \( G^d \) according
We then show (by construction), i.e., \( h_k \) in the sub-hypergraph, then it must belong to a hyperpath \( G \). Note that a sub-hypergraph that are destined for the same destination \( d \). In turn, this implies \( u^d_i = u^d_i \), \( \forall i \in N, s \in S \), we have

Comparing the above two sets of equations and noticing \( u^d_i = u^d_i \), \( \forall i \in N, s \in S \), we have

In turn, this implies \( h_k(C^d_k - u^d_i) = 0 \). Since any hyperpath satisfies the above condition, we have \( h_k(C^d_k - u^d_i) = 0, \forall k, r, d \).

(2) We then show \( h_k(C^d_k - u^d_i) = 0, \forall k, r, d \) implies \( y^d_i(u^d_i - u^d_i) = 0, \forall i, s, d \).

Note that a sub-hypergraph \( G^d(N, S^d) \) is obtained by aggregating all the used hyperpaths that are destined for the same destination \( d \). Let \( s \in S^d \) be a hyperlink starting from node \( i \) in the sub-hypergraph, then it must belong to a hyperpath \( k \) from node \( i \) to \( d \), and we have \( u^d_i = C^d_k \) per definition. Since any hyperpath \( k \) from node \( i \) to \( d \) in \( G^d \) carries positive flows (by construction), i.e., \( h_k > 0 \), \( C^d_k = u^d_i \) as per \( h_k(C^d_k - u^d_i) = 0, \forall k, i, d \). This then leads to \( u^d_i = u^d_i \). Since any hyperlink in the sub-hypergraph satisfies the above condition, we have \( y^d_i(u^d_i - u^d_i) = 0, \forall i, s, d \).

The proof is completed.