Computers & Operations Research 54 (2015) 90–107

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

A framework for and empirical study of algorithms for traffic assignment

Olga Perederieieva [a,*], Matthias Ehrgott [b], Andrea Raith [a], Judith Y.T. Wang [c]

[a] *Department of Engineering Science, University of Auckland, Auckland, New Zealand*
[b] *Department of Management Science, Lancaster University, Lancaster, UK*
[c] *School of Civil Engineering & Institute for Transport Studies, University of Leeds, Leeds, UK*

ARTICLE INFO

Available online 16 September 2014

Keywords:
User equilibrium
Traffic assignment
Algorithms
Numerical study

ABSTRACT

Traffic congestion is an issue in most cities worldwide. Transportation engineers and urban planners develop various traffic management projects in order to solve this issue. One way to evaluate such projects is traffic assignment (TA). The goal of TA is to predict the behaviour of road users for a given period of time (morning and evening peaks, for example). Once such a model is created, it can be used to analyse the usage of a road network and to predict the impact of implementing a potential project. The most commonly used TA model is known as user equilibrium, which is based on the assumption that all drivers minimise their travel time or generalised cost. In this study, we consider the static deterministic user equilibrium TA model.

The constant growth of road networks and the need of highly precise solutions (required for select link analysis, network design, etc.) motivate researchers to propose numerous methods to solve this problem. Our study aims to provide a recommendation on what methods are more suitable depending on available computational resources, time and requirements on the solution. In order to achieve this goal, we implement a flexible software framework that maximises the usage of common code and, hence, ensures comparison of algorithms on common ground. In order to identify similarities and differences of the methods, we analyse groups of algorithms that are based on common principles. In addition, we implement and compare several different methods for solving sub-problems and discuss issues related to accumulated numerical errors that might occur when highly accurate solutions are required.

1. Introduction and motivation

Because of the fast development of cities and road networks, transportation engineers face various difficulties of maintaining current roads and planning improvements and changes of the infrastructure in order to satisfy the existing and future travel demand. Since any changes to the urban infrastructure require large investments of time and money, making wise strategic decisions is very important [1]. One way to evaluate potential projects and analyse the usage of existing road networks is to apply mathematical modelling. As presented in Ortúzar and Willumsen [1], transportation planning tools include many different mathematical models designed to solve various tasks. In our study, we concentrate on the traffic assignment (TA) problem that is part of transportation planning.

The TA model describes travel behaviour of road users. In particular, this model is designed to predict what route choice every individual will make during a given period of time. The conventional approach to model the behaviour of travellers is to make some assumptions on how people choose routes and to find a traffic flow pattern satisfying these assumptions. The most well-known assumptions are the ones following Wardrop's first principle (also called user equilibrium condition): "*The journey times on all the routes actually used are equal, and less than those which would be experienced by a single vehicle on any unused route*" [2].

This principle models the behaviour of travellers by assuming that all drivers are selfish and that they choose the fastest routes going from their origin to their destination. As a result, an *equilibrium state* is achieved, when no one has an incentive to switch to another route. This principle allows different mathematical formulations of the problem based on different additional assumptions. The classical model that is commonly used in practice is a static deterministic traffic assignment (TA) that is the subject of this paper. Other traffic assignment problems such as the TA problem with elastic demand (see [3]) or dynamic TA (see [4]) are not considered in this paper.

This classical model was developed in the 1950s and since then various algorithms have been proposed to solve it. The wide research interest in this problem has several reasons. First, TA is a

[*] Corresponding author.
E-mail address: o.perederieieva@auckland.ac.nz (O. Perederieieva).

http://dx.doi.org/10.1016/j.cor.2014.08.024
0305-0548/© 2014 Elsevier Ltd. All rights reserved.

challenging problem that arises in different practical applications. Second, the existing transportation models continue to grow and become more detailed. For example, in 2006 the ART model (Auckland Regional Transport Model) included 202 zones, whereas in 2008 it contained 512 zones [5]. A *zone* refers to an area of a transportation network that can vary from a city block to a neighbourhood [6]. Third, high accuracy of the solution is required for select link analysis[1] and for consistent comparison between design scenarios, as presented in Slavin et al. [7] and Gentile [8]. Therefore, we can conclude that there is a growing need for efficient algorithms able to solve TA problems of realistic size with high accuracy.

Regardless the number of proposed algorithms in the literature, there is no comprehensive survey study comparing them. To the best of our knowledge, only the paper of Inoue and Maruyama [9] analyses many different methods under the same computational environment. The authors implemented 11 algorithms for traffic assignment. However, all implementation details are omitted in the paper. It is not clear if the authors used the same framework for all algorithms and how carefully they followed the descriptions of the algorithms available in the literature.

The aim of our research is to analyse and compare the most promising approaches for solving TA in the context of a *framework* that can be shared by different algorithms. We compare methods without considering any special implementation details that may have been used in commercial or research software. Instead, we focus on the use of common code wherever possible. This will allow testing general ideas of the methods without giving an advantage to any of them.

Another motivation of this study is to identify the advantages and disadvantages of different *groups* of algorithms (the classification is presented in Section 3). Early on only link-based approaches were available because of memory limitations. In recent years, however, other types of algorithms have been implemented in many commercial software packages used by practitioners. One of the main advantages of new algorithms is that they can provide the path choice information which is necessary to evaluate effects of schemes such as congestion pricing without re-running the algorithm [10]. Therefore, it becomes important to analyse them and to compare them with classical link-based approaches. Other factors that motivate our research are highlighted in Section 3 along with a literature overview.

Preliminary results of our study can be found in Perederieieva et al. [11].

The rest of the paper is organised as follows. Section 2 states the deterministic static traffic assignment problem. Section 3 is devoted to a literature review and our choice of algorithms for comparison. In Section 4, various implemented methods for solving traffic assignment are described. Section 5 discusses the computational study and comparison of algorithms. Finally, Section 6 presents conclusions and future work.

2. Problem formulation

This section introduces a mathematical formulation of the TA problem and notation that is used throughout the paper.

A transportation network is defined as a directed graph $G(N,A)$ where N is a set of nodes and A is a set of links. The users of the transportation network travel from their origins to their destinations. Let D_p denote travel demand between origin–destination (O–D) pair $p \in Z$, where Z is the set of all O–D pairs. A *demand* represents how many vehicles are travelling from an origin to a destination.

[1] Select link analysis provides information of where traffic is coming from and going to for vehicles at selected links (and combination of links) throughout the modelled network [1].

The key feature of TA models consists in taking into account congestion effects that occur in road networks. In order to consider congestion, link cost functions are introduced into the model. They represent travel times through links of a network depending on the traffic flow on those links. Let $c_a(\mathbf{f})$ denote a link cost function of link a that depends on link flows $\mathbf{f} = (f_1, f_2, \dots, f_{|A|})$. Link flow is the number of vehicles per time unit on each link.

Let $\mathbf{F} = (F_1, \dots, F_{|K|})$ denote a vector of path flows, where K is the set of all simple paths of graph $G(N,A)$. Path flows are related to link flows by the following expression:

$$f_a = \sum_{p \in Z} \sum_{k \in K_p} \delta_a^k F_k, \tag{1}$$

where δ_a^k equals one if link a belongs to path k, and zero otherwise; $K_p \subseteq K$ is the set of paths between O–D pair p. Let path cost function $C_k(\mathbf{F})$ denote the travel time on path k.

The conventional model of the traffic assignment problem is based on two assumptions that allow to formulate and solve it as a mathematical programme.

1. *Additivity* of path cost functions: travel time on each path is the sum of travel times of links belonging to this path, i.e. $C_k(\mathbf{F}) = \sum_{a \in A} \delta_a^k c_a(\mathbf{f})$.
2. *Separability* of link cost functions: travel time on each link depends only on flow on this link, i.e. $c_a(\mathbf{f}) = c_a(f_a)$.

If these assumptions are satisfied, solving the following optimisation problem (2) results in the link flows satisfying the user equilibrium condition [6]:

$$\begin{aligned} &\min \sum_{a \in A} \int_0^{f_a} c_a(x)\,dx \\ &\sum_{k \in K_p} F_k = D_p, \quad \forall p \in Z, \\ &F_k \geq 0, \quad \forall k \in K_p, \forall p \in Z, \\ &f_a = \sum_{p \in Z} \sum_{k \in K_p} \delta_a^k F_k, \quad \forall a \in A. \end{aligned} \tag{2}$$

If all path cost functions $C_k(\mathbf{F})$ are *positive and continuous* then existence of a solution of TA is ensured. If, furthermore, the path cost functions are *strictly monotone*, the solution is guaranteed to be unique [12]. In the following, it is assumed that these requirements are satisfied.

The formulation (2) is sometimes referred to as link-route or path flow formulation [13,14]. We will use the latter term. The path-based algorithms that are discussed in Section 4.2 use this mathematical programme. Other optimisation formulations of the TA problem based on a different set of decision variables can be found in Patriksson [13] and Bertsekas [14].

3. Literature overview

One of the possible ways to classify traffic assignment algorithms is according to how the solution is represented: link-based (solution variables are link flows), path-based (solution variables are path flows) and bush-based (solution variables are link flows coming from a particular origin), see [15].

Historically, the first algorithms developed for solving the traffic assignment problem were link-based. The most well-known such algorithm is the Frank–Wolfe (FW), a general algorithm for convex optimisation problems [16]. Due to its simplicity and low memory requirements, it is used even now and is implemented in different commercial software packages. However, this algorithm is known to tail badly in the vicinity of the optimum and usually cannot be used to achieve highly precise solutions as demonstrated by numerous numerical studies [9,10,17].

Many improvements of FW were proposed in the literature. As explained in Zhou and Martimo [15], some of them try to improve the FW search direction (for example, see [18–20]) or step size (see [21,22]). Among these variations of FW, we are interested in conjugate (CFW) and bi-conjugate Frank–Wolfe (BFW) methods [20] as explained later in this section. In Zhou and Martimo [15], the authors also categorise restricted simplicial decomposition (RSD) [23] and non-linear simplicial decomposition (NSD) [24] as link-based methods. These algorithms apply a more complicated structure than FW, however, they apply FW to solve sub-problems. In RSD and NSD the link flow solution is represented by a convex combination of extreme points. Algorithms of this type usually have two main routines: generating new extreme points and optimising the restricted master problem with respect to previously generated extreme points.

Path-based methods decompose the TA problem into sub-problems corresponding to O–D pairs. They bring the current solution to the equilibrium by sequentially shifting path flows. The first path-based algorithm called the path equilibration (PE) was proposed in Dafermos and Sparrow [25]. Its main idea is to shift flow from the longest path (path with maximum cost) to the shortest path (path with minimum cost). However, in the years after the PE algorithm was published, path-based methods were considered impractical because they required to store all paths. When column generation approaches for the TA problem were proposed by Gibert [26] and Leventhal et al. [27], the development of path-based methods started again since it allowed to generate paths when needed instead storing all of them. Larsson and Patriksson [28] proposed disaggregated simplicial decomposition (DSD) which is similar to RSD in the way that it also uses a convex combination of extreme points to represent the solution. However, the extreme points are in the space of path flows instead of link flows. In Jayakrishnan [29], the gradient projection (GP) method was proposed and further studied in Chen and Jayakrishnan [30]. It is similar to PE, but O–D flow is moved from several non-shortest paths to the shortest path. Another path-based algorithm was proposed in Florian et al. [10]. It is called the projected gradient (PG) and is based on the idea of moving flow from the set of paths with cost greater than the average path cost to the set of paths with cost less than the average path cost. Another similar approach called the improved social pressure (ISP) algorithm was developed in Kumar and Peeta [31] and compared to its previous version called the social pressure algorithm (SP). This method also shifts flow from the set of costlier paths to the set of cheaper paths, but a more complicated strategy of flow distribution than in GP and PG is applied.

Bush-based algorithms (sometimes called origin-based) represent a more recent development. Their main idea is to decompose the problem into a sequence of sub-problems that operate on acyclic sub-networks of the original transportation network [32]. In general, for this group of algorithms, the flow shifts are restricted to sub-problems and are usually similar to the ideas presented earlier for path- and link-based approaches. The first algorithm of this type applied to the traffic assignment problem was proposed by Bar-Gera [33]. It is called the origin-based algorithm (OBA). Nie [34] presented some corrections to OBA, and Nie [32] compared different bush-based algorithms (corrected OBA (COBA) and modified OBA (MOBA)) based on OBA. Other recently developed bush-based methods include algorithm B (B) proposed by Dial [35], some modifications of it (iB), see Inoue and Maruyama [9] and Zhang et al. [36], linear user cost equilibrium (LUCE) developed by Gentile [8] and traffic assignment by paired alternative segments (TAPAS) introduced by Bar-Gera [37]. It must be noted that TAPAS cannot be strictly classified as a bush-based algorithm: it uses origin flows in order to represent the solution, however the general structure of the algorithm is different from other bush-based approaches.

In order to select algorithms for implementation, we analyse different empirical studies from the literature. During such analysis it is important to pay attention to how the algorithms were compared (re-implemented by the authors of the study or using existing software), what instances were used and how precise the obtained solutions were.

In the majority of the studies, relative gap is used as a convergence measure. It is calculated as follows:

$$RGAP = 1 - \frac{\sum_{p \in Z} D_p \cdot C^p_{min}}{\sum_{a \in A} f_a \cdot c_a}, \tag{3}$$

where $C^p_{min} = \min_{k \in K_p} C_k$ is the shortest path of O–D pair p. We divide the existing numerical studies into two groups of low and high precision that correspond to the accuracy of the solution. The algorithms from the low precision group are stopped when the relative gap is in the interval $[10^{-7}, 10^{-4}]$, and for the high precision group the interval is $[10^{-14}, 10^{-10}]$. Table 1 presents the first group and Table 2 the second one. In each table, one algorithm from each study is highlighted in bold, which means that it showed the best performance on the majority of the tested instances. All other algorithms in the same study are listed in the order of decreasing performance. If in a particular study the existing executable of the algorithm or commercial software was used, it is highlighted in dark grey. If comparison of the algorithms was made indirectly, i.e. based on the running times reported in other papers, it is highlighted in light grey. We also try to align the algorithms that were compared in different studies. However, since sometimes contradictory conclusions are drawn in different papers, this alignment is approximate. The last line of each table summarises the algorithms that seem to be the most promising. The article of

Table 1
Summary of existing empirical studies. Low precision: $RGAP \in [10^{-7}, 10^{-4}]$.

Mitradjieva and Lindberg [20]	Zhou and Martimo [15]	Slavin et al. [39]	Dial [35]	Florian et al. [10]	Gentile [8]	Zhang et al. [36]	Kumar and Peeta [31]
					LUCE		**ISP**
				PG	PG		SP
		B	**B**		B	**iB**, B	
		OBA	OBA	OBA	OBA		
		FW	FW	FW	FW	FW	
	GP	GP				OBA	
BFW	BFW						
CFW	CFW						
FW	FW						
OBA	OBA						
DSD							
Most promising: BFW, GP, PG, ISP, B, LUCE							

Table 2
Summary of existing empirical studies. High precision: $RGAP \in [10^{-14}, 10^{-10}]$.

Bar-Gera [33]	Nie [32]	Bar-Gera [37]	Inoue and Maruyama [9]
		TAPAS	**TAPAS**
	B		B, iB
	MOBA		
	COBA		
OBA	OBA	OBA	OBA, DSD, MOBA, LUCE, ASD
FW		FW	FW
Most promising: B, TAPAS			

Chen et al. [38] is not included in Tables 1 and 2 because they study convergence of algorithms GP and DSD with respect to relative error based on objective function values. This makes it difficult to categorise this study with respect to relative gap. GP is reported to have a better performance. As discussed later, we also implement GP in our study.

As presented in Tables 1 and 2, in many studies only a few algorithms are compared. Often existing executables of algorithms and low precision are used. None of these studies analysed how different methods for solving sub-problems influence the performance of TA algorithms. These facts motivate us to study the TA methods in more detail and to analyse their advantages and disadvantages.

Following the above analysis of Tables 1 and 2, we choose to implement the following algorithms:

1. *Link-based:* FW (FW is consistently reported to have the worst performance. However, it is the basis for CFW and BFW, which is why it is also considered in our study), CFW, BFW.
2. *Path-based:* PE (this algorithm was not compared in the studies presented above. We choose to implement it due to its simplicity and ideas similar to other path-based approaches), GP, PG and ISP.
3. *Bush-based:* B, LUCE, TAPAS.

4. Algorithms

One of the reasons for the development of various traffic assignment algorithms is a specific problem structure that can be exploited by solution methods in many different ways [13].

The constraints of the TA problem represent a polyhedral set and the objective function is convex. As a result, feasible direction methods can be applied as a solution technique [40]. The algorithms described later in this section belong to this type of nonlinear optimisation methods. The main idea behind this type of methods is: starting from a feasible solution, a feasible direction of descent is calculated and the solution is moved along this direction [41]. Usually a new solution, obtained in such a way, has a decreased value of the objective function.

Another property of traffic assignment that can be exploited is the O–D pair separability: the flow conservation constraints of one O–D pair do not affect those of any other pair. This fact gives raise to various algorithms based on the idea of decomposing the problem into smaller sub-problems that can be solved one after another or in parallel [13].

The TA problem allows various formulations based on different solution variables [13]. As a result, different solution spaces are used by the methods such as link, path, and origin flows.

This section discusses several decomposition approaches and algorithms. In particular, we focus on the algorithms that were chosen for our numerical study, see Section 3.

4.1. Link-based algorithms

This section presents Frank–Wolfe and two of its modifications: conjugate and bi-conjugate Frank–Wolfe methods. These algorithms can be described by the framework presented in Fig. 1. The methods differ only in the way the direction of descent is defined.

Each algorithm starts with an initial feasible link flow pattern. It is usually generated by *all-or-nothing* (AON) assignment. Each link flow is initialised with zero and corresponding link travel times are calculated. Then shortest paths for each O–D pair are found and all corresponding demand is assigned to each shortest path. All these path flows are then projected on all links using Eq. (1) which gives the initial feasible link flows.

FW further exploits the AON procedure. In order to generate a feasible direction of descent, it performs AON assignment at each iteration and obtains corresponding link flows $y_a^{AON,i},\ \forall a \in A$. Since the AON link flow solution is feasible, given a current link flow solution $f_a^i,\ \forall a \in A$, vector $d_a^i = y_a^{AON,i} - f_a^i,\ \forall a \in A$ is a feasible direction. Moreover, this direction is also a direction of descent [6]. The FW direction of descent is, in fact, obtained by solving a linearised problem (the objective function in (2) is linearised based on the first order Taylor series expansion) [14]. For the TA problem this linearised problem becomes a shortest path problem and, hence, AON is applied [14].

The CFW and BFW algorithms use information about previously generated directions of descent in order to find a new one. In particular, the new search direction is constructed via mutually conjugate directions with respect to the Hessian of the objective. CFW takes into account the usual FW direction and the direction from the previous iteration, whereas BFW in addition considers one more direction from iteration $i-2$ where i is the current iteration. Both algorithms operate on auxiliary variables $s_a^i,\ \forall a \in A$ called points of sight that store information about previously generated directions of descent (for details see Mitradjieva and Lindberg [20]).

After finding a direction of descent, each algorithm proceeds to step size calculation which determines how far the current solution must be moved along the direction of descent. Section 4.4.2 discusses how to solve this sub-problem.

4.2. Path-based algorithms

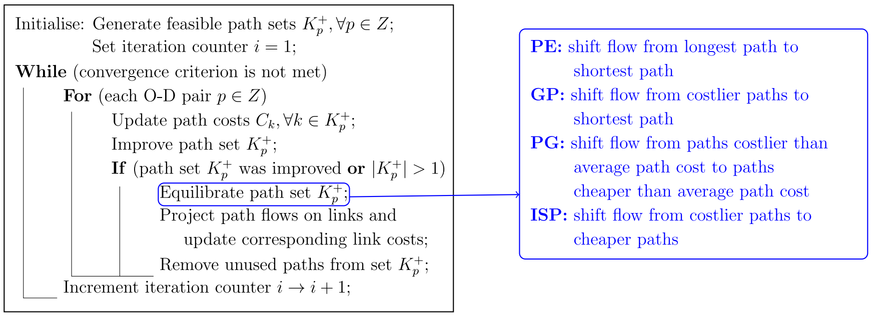
Path-based methods exploit the O–D pair separability and a path flow formulation (2) of the TA problem. This group of methods operates in the space of path flows. At each iteration, the flows are moved only within one O–D pair and path flows of the other O–D pairs are fixed. Therefore, paths and the corresponding path flows must be stored. Let K_p^+ denote the set of paths between O–D pair p that are currently in use, i.e. they carry positive flow. A general framework of this group of algorithms is presented in Fig. 2.

In order to prevent storing all possible paths for each O–D pair, a *column generation* approach is usually applied, which is based on the idea of generating new paths when needed [13]. For a given O–D pair p find the shortest path and add it to K_p^+ if the found path is shorter than the current shortest path contained in this set. This step corresponds to "*Improve path set* K_p^+" of the framework. In order to keep only promising paths in K_p^+, the paths that do not carry flow are removed from K_p^+.

```
Initialise: Generate feasible link flow solution $\mathbf{f^0}$;
            Set iteration counter $i = 1$;
While (convergence criterion is not met)
    Update link costs $c_a(f_a^i), \forall a \in A$;
    Find direction of descent $\mathbf{d}^i$;
    Find step size $\lambda$;
    Update solution: $f_a^{i+1} \rightarrow f_a^i + \lambda \cdot d_a^i$;
    Increment iteration counter $i \rightarrow i + 1$;
```

FW: $d_a^i = y_a^{AON,i} - f_a^i$
CFW: $d_a^i = \alpha \cdot s_a^{i-1} + (1-\alpha) \cdot y_a^{AON,i} - f_a^i$
BFW: $d_a^i = \alpha_1 \cdot s_a^{i-2} + \alpha_2 \cdot s_a^{i-1} + \alpha_3 \cdot y_a^{AON,i} - f_a^i$

Fig. 1. Framework for link-based algorithms.



```
Initialise: Generate feasible path sets $K_p^+, \forall p \in Z$;
            Set iteration counter $i = 1$;
While (convergence criterion is not met)
    For (each O-D pair $p \in Z$)
        Update path costs $C_k, \forall k \in K_p^+$;
        Improve path set $K_p^+$;
        If (path set $K_p^+$ was improved or $|K_p^+| > 1$)
            Equilibrate path set $K_p^+$;
            Project path flows on links and
                update corresponding link costs;
            Remove unused paths from set $K_p^+$;
    Increment iteration counter $i \to i + 1$;
```

```
PE: shift flow from longest path to
    shortest path
GP: shift flow from costlier paths to
    shortest path
PG: shift flow from paths costlier than
    average path cost to paths
    cheaper than average path cost
ISP: shift flow from costlier paths to
    cheaper paths
```

Fig. 2. Framework for path-based algorithms.

As in the case of link-based methods, initialisation is performed by AON assignment. In addition to a link flow solution, each set K_p^+, $\forall p \in Z$ must also be initialised. This can be done by adding the shortest path corresponding to O–D pair p to K_p^+. Every such shortest path p is initialised with flow equal to demand D_p.

Let a commodity refer to trips between a single O–D pair. Due to the decomposition by O–D pairs, the solution to the original TA problem is found by solving several single commodity sub-problems sequentially until the desired precision of the solution is reached. This single commodity sub-problem is identical to (2), but restricted to O–D pair p and fixed set of paths K_p^+. Path-based algorithms differ in how such a sub-problem is solved. As presented in Patriksson [13], for the single commodity sub-problem the feasible direction of descent is defined as the path cost differences between cheaper and costlier paths. Equivalently, moving the current solution in the feasible descent direction means that path set K_p^+ is equilibrated, i.e. some or all the path costs of set K_p^+ are equalised.

Many algorithms for convex optimisation problems scale the direction of descent by the Hessian matrix or its approximation in order to achieve higher convergence rate [41]. In the case of a single commodity sub-problem the direction of descent can also be scaled before applying a line search yielding a constrained quasi-Newton or Newton method depending on how the direction is scaled. “Newton method” here means that second-order derivatives are used to guide the search direction to solve decomposed sub-problems. However, the structure of these algorithms is a Gauss–Seidel decomposition[2] process, the convergence rate of which is linear [32].

If the direction of descent is scaled by the Hessian matrix, then the resulting method corresponds to a constrained version of the Newton method [41]. If an approximation of the Hessian is used instead, then the method can be viewed as a scaled gradient projection or constrained quasi-Newton approach [41].

The case when the direction of descent contains only two non-zero elements (it also means that the flow is moved between only two paths) is of a particular interest. In this case, the Hessian matrix is a scalar and can be easily calculated. In the following, this situation will be referred to as Newton step (for details, see Section 4.4.3). When there are more than two non-zero elements in the direction of descent, i.e. more than two paths are considered simultaneously, then different approximations of the Hessian matrix can be used.

The following subsections discuss different path-based algorithms and their variations depending on whether a scaled or non-scaled direction of descent is applied. All algorithmic steps are described for one O–D pair denoted by p. We also simplify the notation of path cost functions by using C_k instead of $C_k(\mathbf{F})$.

[2] Gauss–Seidel decomposition refers to the process of solving a problem by decomposing it into several sub-problem and solving them sequentially several times until a required precision is achieved [13].

4.2.1. Path equilibration

The PE algorithm equalises the costs of the current longest path $l \in K_p^+$ with positive flow and the current shortest path $s \in K_p^+$. The direction of descent $\mathbf{d}$ is defined as follows [12]:

$$\begin{aligned} d_s &= C_l - C_s, \\ d_l &= C_s - C_l, \\ d_j &= 0, \quad \forall j \in K_p^+,\ j \neq s,\ j \neq l. \end{aligned} \tag{4}$$

The current path flow solution is moved along this direction using the step size calculated as presented in Section 4.4.2.

Since only two path costs are equalised, the Newton step can be applied as discussed in Section 4.4.3.

Originally PE was implemented to solve the traffic assignment problem with quadratic link cost functions and was applied only to small instances [27]. In Leventhal [27], it is not specified what method was used to equalise path costs at each iteration. Florian and Hearn [12] present a feasible direction method with a line search for PE. We implemented both scaled and non-scaled descent directions for this algorithm.

4.2.2. Gradient projection

The GP algorithm considers several paths at each iteration. In particular, it moves flow to the current shortest path $s \in K_p^+$ from all other paths of set K_p^+. As presented in Jayakrishnan et al. [29], firstly, a flow shift is calculated:

$$\Delta F_k = \frac{C_k - C_s}{\sum_{a \in A_{s,k}} \frac{\partial c_a}{\partial f_a}}, \qquad \forall k \in K_p^+,\ k \neq s, \tag{5}$$

where $A_{s,k}$ is the set of links that belong either to path k or to path s but not to both of them. Secondly, a new solution is projected onto the feasible set:

$$\begin{aligned} F_k &= F_k - \min\{\alpha \Delta F_k, F_k\}, \qquad \forall k \in K_p^+,\ k \neq s, \\ F_s &= D_p - \sum_{k \in K_p^+, k \neq s} F_k, \end{aligned} \tag{6}$$

where α is a predefined constant that must be small enough in order to guarantee convergence of the algorithm [29]. Jayakrishnan et al. [29] recommend to set it to 1. However, after several runs of the algorithm we set α to 0.25, because this value allows all tested instances to converge. The algorithm performance depends a lot on parameter α. In particular, for some instances when the algorithm converges with $\alpha = 1$, it achieves the required level of precision much faster compared to the case when $\alpha = 0.25$. However, we did not adjust α specifically for every instance, but rather fixed it to a value that allows all tested instances to converge. Chen et al. [42,43] also suggest a self-adaptive step size strategy for GP and other algorithms that we did not implement in our study. Another modification of GP based on conjugate directions is presented in Lee et al. [44].

This version of GP with projection operator can be classified as a scaled gradient projection with diagonal approximation of the Hessian [14]. In the following, this method will be referred to as GP Newton.

Alternatively, instead of projecting the current solution onto the feasible set, a feasible direction of descent combined with a line search can be used

$$d_k = \frac{C_s - C_k}{\sum_{a \in A_{s,k}} \frac{\partial c_a}{\partial f_a}}, \quad \forall k \in K_p^+, \; k \neq s,$$
$$d_s = -\sum_{k \in K_p^+, k \neq s} d_k. \tag{7}$$

Once the direction of descent is calculated the current solution is updated: $F_k = F_k + \lambda d_k$, where λ is a step size found by applying a line search (see Section 4.4.2). This approach was presented in Cheng et al. [45]. In the rest of the paper, this method will be called GP2.

Another variation of GP includes a non-scaled feasible direction combined with a line search. The descent direction is

$$d_k = C_s - C_k, \quad \forall k \in K_p^+, \; k \neq s,$$
$$d_s = -\sum_{k \in K_p^+, k \neq s} d_k. \tag{8}$$

In the following, this method will be denoted GP1.

4.2.3. *Projected gradient*

The main idea of the PG algorithm is to move flow from the paths that have cost greater than the current average path cost to the paths that have cost less than the average value. It is equivalent to defining the following direction of descent **d**:

$$d_k = \overline{C_p} - C_k, \quad \forall k \in K_p^+, \tag{9}$$

where $\overline{C_p} = \sum_{k \in K_p^+} C_k / |K_p^+|$ is the average cost of the paths of O–D pair p [10]. In order to find the appropriate amount of flow to move, a line search is applied along direction **d**. As a result, step size λ is calculated and the path flows are updated accordingly: $F_k = F_k + \lambda d_k, \; \forall k \in K_p^+$.

For this particular algorithm, it is not obvious how the Hessian matrix can be approximated. A first idea that comes to mind is to apply the same approach as for GP Newton or GP2, namely by considering only two paths at a time. However, it is not clear how to choose these two paths (in GP Newton flow is always shifted from non-shortest paths to the shortest path). Therefore, scaled direction of descent is not implemented in our study for PG.

4.2.4. *Improved social pressure*

The ISP algorithm is based on the idea of "*social pressure*" which is defined as the difference between the cost of a path and the cost of the shortest path [31]. All the paths are divided into two groups $\underline{P}_p \subset K_p^+, \; \forall p \in Z$ (paths with cost less than or equal to some value π) and $\overline{P_p} \subset K_p^+, \; \forall p \in Z$ (paths with cost greater than π) such that $\underline{P}_p \cup \overline{P_p} = K_p^+$. The value of π is determined at each iteration as $\pi = C_s + \delta \cdot (C_l - C_s)$, where C_s and C_l are the costs of the current shortest and longest paths, δ is a predefined constant that was set to 0.15 as suggested in Kumar and Peeta [31]. Flow is shifted from the paths belonging to set $\overline{P_p}$ to the paths belonging to set $\underline{P}_p$. This is equivalent to defining a direction of descent **d** and moving the solution along this direction. Direction **d** is

$$d_k = C_s - C_k, \quad \forall k \in \overline{P_p},$$
$$d_l = \frac{-\sum_{k \in \overline{P_p}} d_k}{s_l(F_l) \cdot \sum_{m \in \underline{P}_p} \frac{1}{s_m(F_m)}}, \quad \forall l \in \underline{P}_p, \tag{10}$$

where $s_m(F_m)$ is the first derivative of the cost function of path m with respect to path flow, $s_m(F_m) = \sum_{a \in m} \partial c_a(f_a)/\partial f_a$. The step size λ is calculated as presented in Section 4.4.2 and path flows are updated according to $F_k = F_k + \lambda d_k, \; \forall k \in K_p^+$.

ISP scales some elements of the direction of descent by a scaling factor based on second derivative information. Such an approach can be viewed as an approximation of the Hessian.

Applying a non-scaled direction of descent to PG makes this algorithm equivalent to GP1 (the non-scaled part of the direction of descent in Eq. (10) is the same as in Eq. (8) of GP1) and, hence, is not implemented in our study.

4.3. *Bush-based algorithms*

Bush-based algorithms exploit the O–D pair separability of the TA problem. But instead of decomposing the problem into sub-problems corresponding to each O–D pair, algorithms of this type decompose the problem into sub-problems corresponding to each origin. At each iteration, flows are moved within a special data structure called the *bush*. As presented in Nie [32], a bush is a directed sub-network of the original network $G(N, A)$, rooted at a given origin o. Every bush is

1. *Connected*, i.e. using only links in the bush, it is possible to reach every node that is reachable in the original network.
2. *Acyclic*, i.e. the bush does not contain directed cycles.

We denote a bush by $B_o(A_o, N) \subset G(A, N)$, where $A_o \subset A$. Let $O \subseteq N$ denote the set of origins. Then, in each iteration, only one bush $B_o(A_o, N)$ is considered. The solution variables are represented by origin flows f_a^o, which denotes the flow on link a of the bush $B_o(A_o, N)$ [35]. The current link-based solution **f** is the sum of $|O|$ link flows of $B_o(A_o, N), \; \forall o \in O$, i.e. $f_a = \sum_{o \in O} f_a^o, \; \forall a \in A$.

The decomposition with respect to origins is equivalent to solving the TA problem by sequentially solving several single commodity problems as in the case of decomposition by O–D pairs. However, the difference consists in the sub-problem formulation: path-based approaches use a path flow formulation defined for each O–D pair, bush-based methods use a link flow formulation defined for each origin. Link flow formulations do not use path flow variables in the constraint set. Several such formulations based on different decision variables exist, see [14,32,35].

Decomposition of the traffic assignment problem by origin is possible due to the *acyclicity of user equilibrium*: for the single origin formulation (when the problem is restricted to flows coming from a given origin) of the traffic assignment problem, links that have positive flow at user equilibrium never form a directed cycle [32]. This property gives several advantages from a practical point of view since different operations such as shortest path calculations can be performed more efficiently on acyclic structures (such as bushes) than on general networks [32]. This is possible due to existence of *topological orders* in directed acyclic graphs. A topological order is a labelling of each node of an acyclic graph with a number between 1 and $|N|$ such that every link connects a node of lower topological order to a node of higher topological order [46].

All algorithms that decompose the TA problem by origins share a general framework presented in Fig. 3. Each bush $B_o(A_o, N)$ is usually initialised with a shortest path tree rooted at origin o and the link flows are initialised by AON assignment.

As in the case of path-based methods, bushes are constructed iteratively by adding promising links and removing unused ones. The links that carry zero flow are dropped if the connectivity of the bush is retained. The addition of new links must be performed with care in such a way that directed cycles are not created.

```
Initialise: Generate feasible bushes $B_o^0, \forall o \in O$;
            Set iteration counter $i = 1$;
While (convergence criterion is not met)
    For (each origin $o \in O$)
        If (links were removed from bush $B_o^{i-1}$)
            Create topological order for bush $B_o^i$;
            Build min- and max-trees for bush $B_o^i$;
        Improve bush $B_o^i$;
        If (bush $B_o^i$ was improved)
            Create topological order for bush $B_o^i$;
            Build min- and max-trees for bush $B_o^i$;
        Equilibrate bush $B_o^i$;
        Project origin flows on links and
            update corresponding link costs;
        Remove unused links from bush $B_o^i$;
    Increment iteration counter $i \to i + 1$;
```

B: shift flow from longest path to shortest path within current bush

LUCE: shift flow within current bush by solving quadratic sub-problem

Fig. 3. Framework for bush-based algorithms.

The reader is referred to Nie [32] for a detailed explanation of this algorithmic step.

4.3.1. *Algorithm B*

Algorithm B was introduced by Dial [35]. Its main idea is similar to PE (see Section 4.2.1), i.e. the flow is shifted from the longest used path to the shortest path, but within a given bush. Instead of considering paths from a given origin explicitly, for every node j of the current bush (considered in descending topological order) the algorithm builds two segments of paths: one segment belongs to the shortest path ending at node j and the other one belongs to the longest path with flow also ending at node j (if such a segment exists). Both segments do not share links and start at the same node. Once the segments are built, flow is moved from the segment with higher cost to the segment with lower cost. As in the case of PE, in order to find the amount of flow to move several approaches can be implemented, i.e. feasible direction with a line search and Newton step.

4.3.2. *Linear user cost equilibrium algorithm*

This algorithm was proposed by Gentile [8]. It decomposes the problem by destinations. We reformulated it as an origin-based approach in order to use the same framework as for other bush-based algorithms. The main idea of this method is to seek at every node the equilibrium flow coming from the same origin among the in-coming links of this node. LUCE uses flow portions $\phi_{ij}, \ \forall (i,j) \in A_o$ as solution variables. Let η_j denote the total amount of flow coming to node j from a given origin, then ϕ_{ij} is a portion of flow on every link coming into node j.

The descent direction of LUCE is obtained by solving a quadratic programming sub-problem presented in [8]. Unlike algorithm B, this sub-problem is based on flow portions ϕ_{ij}. This raises certain difficulties related to the fact that the second-order derivative of the objective function with respect to ϕ_{ij} is not available in the closed form [47]. The practical implications of this are discussed in Xie et al. [47] and in Section 5.2.3. Once the direction of descent is calculated, a line search is used in order to determine the appropriate flow shift.

4.3.3. *TAPAS*

This algorithm was developed by Bar-Gera [37]. As mentioned earlier, this method cannot be strictly classified as a bush-based approach since its structure is different from other bush-based techniques. However, we classify it as bush-based because it uses origin flows as solution variables. The TAPAS framework is presented in Fig. 4.

```
Initialise: Generate feasible bushes $B_o^0, \forall o \in O$;
            Set iteration counter $i = 1$;
While (convergence criterion is not met)
    For (each origin $o \in O$)
        Remove cyclic flows from $B_o^i$
        Find tree of least cost paths
        For (every link $a$ used by origin $o$ which is not part of the least cost tree)
            Either create a new PAS or add origin $o$ to existing PAS
        For (every active PAS)
            Shift flow within PAS to equilibrate costs
    For (every active PAS)
        Check if PAS must be deleted
        Perform flow shifts to equilibrate costs
    Increment iteration counter $i \to i + 1$;
```

Fig. 4. Framework for TAPAS.

The algorithm uses paired alternative segments (PAS). *A paired alternative segment* contains two path segments that do not share links and have the same first and last nodes. A PAS is not related to any particular O–D pair or origin, but it can be the part of paths belonging to different O–D pairs. In fact, each PAS has a set of origins associated with it.

TAPAS consists of two routines. First, it loops through all origins (during this stage new PASs are created and flow shifts are performed). Second, it loops through all PASs (during this stage flows are shifted within the PASs and inactive PASs are removed).

Flow is shifted within each PAS from the higher cost segment to the lower cost one. This flow shift is similar to the one in PE and algorithm B and the same approaches can be applied in order to do this (feasible direction with a line search or Newton step). In addition, each PAS has a set of relevant origins, which means that when a flow shift within a given PAS is performed, origin flows of relevant origins must be updated (for details see [37]).

It is not necessary to keep bushes acyclic in the way it is usually done in bush-based approaches. Cyclic flows are directly removed from each bush if they occur (in case of algorithm B or LUCE links that can create directed cycles are never added to the bush). This is done by applying a topological sorting algorithm to links of a given bush that carry positive flow. Then cyclic flow is removed along each identified cycle.

TAPAS seems to be similar to algorithm B because it also moves flows between path segments. However, unlike algorithm B, one flow move in TAPAS usually involves changes of origin flows of several origins.

It must be noted that we did not implement the iterations of maintaining the condition of path flow proportionality because we did not need unique path flows which are guaranteed to be found if this condition is applied. Our implementation is not randomised as proposed in the paper of Bar-Gera [37] in order to avoid running TAPAS several times for our computational experiments.

4.4. Solving sub-problems

This section presents different methods that were implemented in our study for solving sub-problems that arise in the TA algorithms outlined in Sections 4.1–4.3.

4.4.1. Shortest path

Each TA method requires solving shortest path problems many times. Link-based approaches need a single-source shortest path algorithm for general graphs. The same type of shortest path algorithm is also required for the relative gap calculation. For this purpose, we implemented the label correcting algorithm presented in Sheffi [6].

Path-based approaches, on the other hand, require a point-to-point shortest path method since at each iteration only one O–D pair is considered. As discussed in Section 4.2, before shifting flows between paths of a particular O–D pair, we check if the set of active paths contains a path with smaller cost than the current shortest path. For this, we used the A* algorithm,[3] see Goldberg et al. [48].

Alternatively, instead of calculating the shortest path between each O–D pair before performing a flow move, we can calculate shortest paths from a given origin to all other nodes and use this shortest path tree in further calculations, as suggested in Chen and Jayakrishnan [30]. Once all O–D pairs coming from one origin are considered, a new shortest path tree is calculated for the next origin. Thus, the same shortest path tree is used for all O–D pairs coming from one origin, but the link travel times are updated after each flow move. The advantage of this approach is a smaller number of shortest path calculations: $|O|$ one-source shortest path calculations (where $|O|$ is the total number of origins) compared to $|Z|$ point-to-point shortest path calculations (where $|Z|$ is the total number of O–D pairs). However, this approach might lead to an increase of the total number of iterations required by a path-based algorithm since the information about new shortest paths is available only when the algorithm proceeds to the next origin. We did not implement this particular shortest path strategy in our study. It is subject of our future research.

For bush-based methods we implemented the shortest path algorithm for directed acyclic graphs that uses topological order, see Dasgupta et al. [46].

4.4.2. Line search

This section discusses how to find a step size that can be used to move a current solution along a given direction of descent. We denote by $\mathbf{y} = (y_1, \dots, y_{|A|})$ the direction of descent defined in terms of link flows. Determining step size λ is equivalent to solving the following optimisation problem [10]:

$$V(\lambda) = \min \sum_{a \in A} \int_0^{f_a + \lambda y_a} c_a(x)\, dx$$
$$\text{s.t.} \quad 0 \le \lambda \le \lambda^{ub}, \tag{11}$$

where f_a is the current flow of link a and λ^{ub} is an upper bound on the step size that ensures that the new solution will remain feasible.

For link-based approaches and LUCE, the direction of descent is already defined in terms of link flows and can be directly used as $\mathbf{y}$. For path- and other bush-based methods, the direction of descent must be projected onto links: $\mathbf{y} = \Delta \mathbf{d}$ where Δ is a matrix with elements δ_a^k such that δ_a^k is equal to one if link a belongs to path k and zero otherwise.

The value of λ^{ub} depends on the direction of descent. It is equal to one if the direction of descent cannot cause the solution to become infeasible (link-based methods and LUCE). If the solution can become infeasible, the upper bound is derived from feasibility constraints: $\lambda^{ub} = \min_k \{ -F_k^{max}/d_k | d_k < 0 \}$ where F_k^{max} is a maximum feasible flow shift (it might be path or origin flow depending on the algorithm) [12,45].

Problem (11) is a one-dimensional optimisation problem. It can be solved in one of the following ways:

1. *Exactly:* find a value of λ that minimises the objective function in (11), for example, by using bisection, see [6].
2. *Approximately:* find the value of λ that does not necessarily minimise the objective function in (11), but guarantees the decrease of the objective function value after applying this step size. For example, one of the following approaches can be applied:
   (a) *Armijo-like rule*: Determine the largest step $\lambda = \beta^k$ for any non-negative integer k, such that the directional derivative of the objective function $V(\lambda)$ (see mathematical programme (11))
   $$\frac{\partial V(\lambda)}{\partial \lambda} = \mathbf{c}(\mathbf{f} + \lambda \mathbf{y})^T \cdot \mathbf{y}, \tag{12}$$
   is negative [8]. The decrement β in our study was set to 2 as proposed in [8].
   (b) *Quadratic approximation*: The objective function along the direction of descent is approximated by a quadratic function and minimised using the analytic solution, i.e. $\lambda = \min\{\lambda^{ub}, 1/(1-(\partial V(\lambda^{ub})/\partial\lambda)/(\partial V(0)/\partial\lambda))\}$, see [8,45].

In our study, we implement bisection, the Armijo-like rule and quadratic approximation. The impact of different line search strategies on the performance of TA algorithms is discussed in Section 5.2.1.

4.4.3. Newton step

This section presents how to calculate a flow shift between two paths in order to equalise their costs using the Newton method.

Let l denote a path from which the amount of flow ΔF will be shifted to a path s. If $C_l(\Delta F)$ and $C_s(\Delta F)$ are the costs of path l and s, respectively, after the flow shift, then one has to solve the non-linear equation

$$C_l(\Delta F) - C_s(\Delta F) = 0, \tag{13}$$

with respect to flow shift ΔF. After applying the Newton method to Eq. (13), the appropriate flow shift is

$$\Delta F = \frac{C_l - C_s}{\sum_{a \in A'} \frac{\partial c_a}{\partial f_a}}, \tag{14}$$

where A' is the set of links that belong to path l and path s without links common to both these paths. It must be noted that flow shift ΔF might be infeasible causing F_l to become negative. That is why projection onto the feasible set is required:

$$F_l = F_l - \min\{F_l, \Delta F\},$$
$$F_s = F_s + \min\{F_l, \Delta F\}. \tag{15}$$

A Newton step can be applied iteratively to the two paths under consideration until the desired precision is reached. However, it is usually applied only once [47] and we follow this convention.

[3] An implementation of the A* algorithm was kindly provided by Boshen Chen.

4.4.4. *Equilibration strategies*

Path- and bush-based algorithms iterate between improving an element of decomposition (add better paths to a set of active paths, add better links to a bush, create new PASs) and equilibrating it. This set of steps can be performed once for each of the decomposition elements or several times before proceeding to the next one. In the following, the strategy of improving and equilibrating each element only once will be called the *Equilibration I* and the strategy of improving and equilibrating a current element several times will be called the *Equilibration II*. These approaches are also presented in the literature [32,49].

In order to implement Equilibration II we must decide how many iterations of improving and equilibrating should be applied to a given decomposition element before proceeding to the next one. We propose to switch to a new element if the current decomposition element is equilibrated to some desired precision. We also limit the maximum number of these iterations to 10 in order to have more control over the total running time of the algorithms. Equilibration I and II are studied in Section 5.2.2.

4.5. *Difficulties of comparison of groups of algorithms*

As can be seen from Sections 4.1 to 4.3, the algorithms that belong to the same group of methods (link-, path- and bush-based) share the same framework and differ only slightly. This allows fair comparison between the algorithms from the same group. However, the methods belonging to different groups require special algorithms specific only to that particular group. This might lead to difficulties when comparing groups of algorithms even though they use the same code as much as possible. For example, all path-based methods implement A* to solve point-to-point shortest path problems. However, there might exist more efficient algorithms for this purpose that might lead to a better performance of the entire group of path-based methods. Moreover, special implementation techniques might further improve the performance of some groups of algorithms or particular algorithms (for example, “thread following” used to update shortest paths [50], flow update schemes for FW [51], multi-threaded implementations [39], etc.). Therefore, the comparison of groups of algorithms should be considered with care as it possibly depends on particular implementations.

5. Computational study

5.1. *Problem instances and computational environment*

We performed computational tests on the instances available at the web-site http://www.bgu.ac.il/~bargera/tntp/. The main characteristics of these instances are presented in Table 3. The first five instances are of small to medium size, whereas the last three instances are large and are used in a separate numerical test (see Section 5.2.4).

Table 3
Problem instances.

Instance name	# of nodes	# of links	# of zones	# of O–D pairs	Class
Sioux Falls	24	76	24	528	Small
Anaheim	416	914	38	1406	Small
Barcelona	930	2522	110	7922	Medium
Winnipeg	1040	2836	147	4344	Medium
Chicago Sketch	933	2950	387	93,135	Medium
PRISM	14,639	33,937	898	470,805	Large
Philadelphia	13,389	40,003	1525	1,149,795	Large
Chicago Regional	12,982	39,018	1790	2,296,227	Large

All instances use BPR link cost functions of the form

$$c_a(f_a) = freeFlow \cdot \left(1 + B \cdot \left(\frac{f_a}{capacity}\right)^{power}\right),$$

where *freeFlow*, *B*, *capacity* and *power* are function parameters.

All algorithms were implemented in the C++ programming language in terms of the framework presented in Section 4 that ensures the usage of common code wherever possible. We used the compiler g++ 4.7.3 (Ubuntu/Linaro 4.7.3-1ubuntu1) with optimisation option −O3.

All algorithms were implemented with different options. For line search we implemented bisection, Armijo-rule and quadratic approximation; for the different algorithms we considered scaled and non-scaled directions of descent whenever possible (see Section 4). For all algorithms we used extended floating point precision (C++ long double type). All tests on small and medium instances were performed under the environment OS – Ubuntu Release 13.10 64-bit; CPU – Intel Core i5-2500 CPU, 4 Core, 3.30 GHz; RAM – 7.7 GB. All tests on large instances were performed under the environment OS – Ubuntu Release 12.10 64-bit; CPU – Intel Core i5-3570 CPU, 4 Core, 3.40 GHz; RAM – 15.6 GB.

5.2. *Results*

We use the relative gap *RGAP* (see Eq. (3)) as a convergence criterion because it is a common measure of convergence (see [10,32,35,39]), and it can be calculated for all tested algorithms.

We decided to perform several numerical tests. Since tests on large instances might require long computational time, we first performed tests on small and medium instances with different configuration options for all algorithms. Based on the results of these tests, we chose the best configuration options for each algorithm and eliminated methods that do not seem promising and performed numerical tests on large instances with the remaining algorithms.

It must be noted that not all algorithms can find highly accurate solutions in a reasonable amount of time. Because of this we developed several tests in order to choose the best options for every algorithm.

1. *Test*1: For this test, we chose the algorithms that can achieve high precision (PE, GP, ISP, B, TAPAS) and examined Equilibration I strategy. The required accuracy of the solution was set to $\epsilon = 10^{-14}$, i.e. the algorithms were stopped after the relative gap was less than ϵ.
2. *Test*2: In this test, we investigated the impact of Equilibration II on the algorithms from Test 1. Again, high precision of $\epsilon = 10^{-14}$ was used.
3. *Test*3: This test is devoted to all remaining algorithms that cannot achieve high precision (FW, CFW, BFW, PG, LUCE) in a reasonable amount of time. A time limit was used in addition to the stopping condition based on the relative gap.

Finally, *Test*4 is devoted to large instances. Only the best algorithms with the best options were considered and a time limit was used as an additional stopping criterion.

5.2.1. *Test 1: high precision*

Test 1 was performed on the five small and medium instances listed in Table 3. We tested the following algorithms with Equilibration I and high precision of 10^{-14} of relative gap: PE, GP, ISP, B and TAPAS with various options for line search and direction of descent.

Figs. 5 and 6 show CPU time in seconds for different instances where algorithms are grouped in different ways. In Fig. 5 all algorithms are grouped according to the way the step size is calculated: Armijo rule, bisection, quadratic approximation and other methods that do not use a line search (this category includes all methods that implement Newton step and GP Newton). In Fig. 6, all configurations are grouped according to a particular algorithm. However, GP Newton is not considered because it

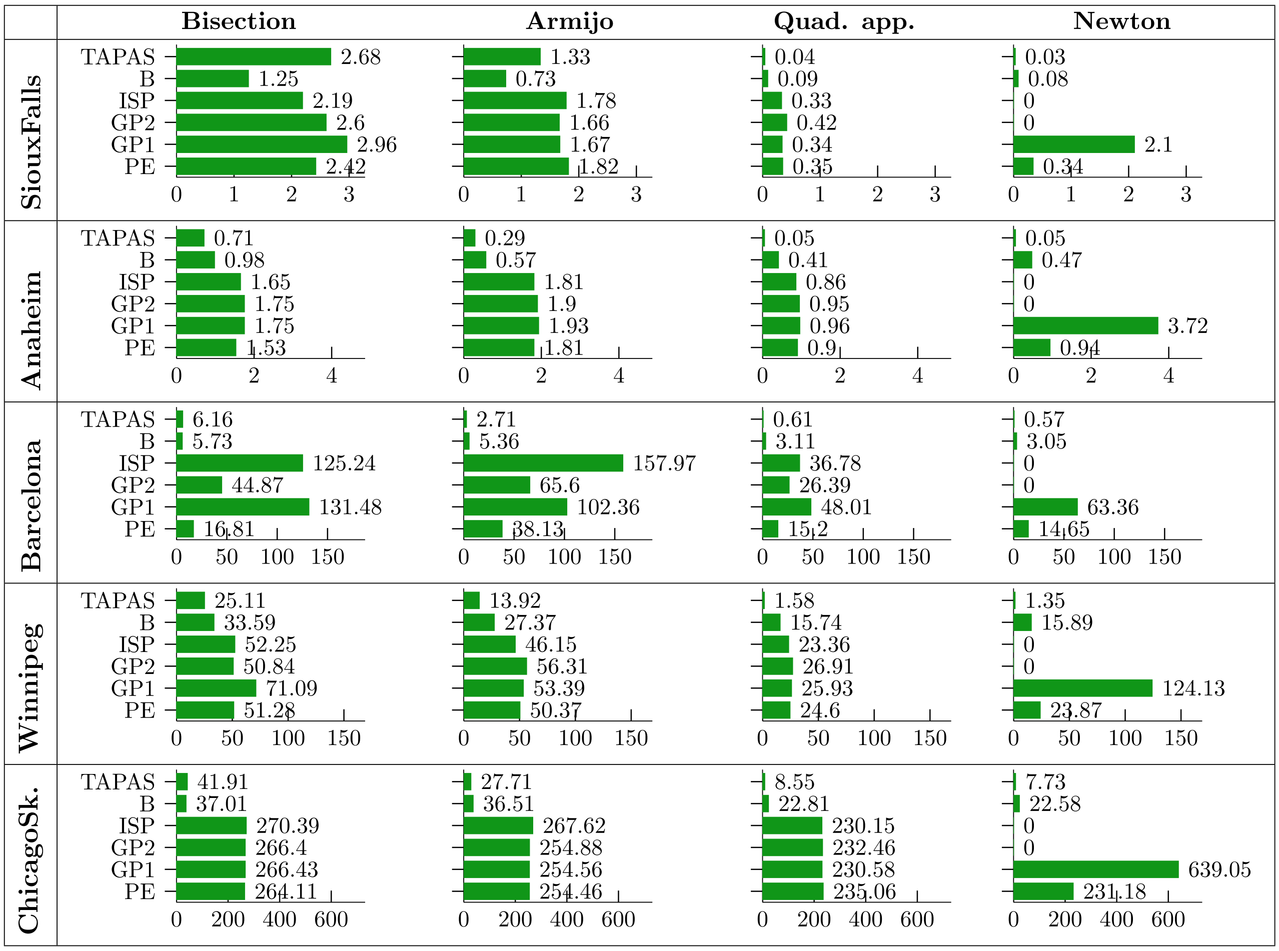


Fig. 5. Test 1. CPU time, s.

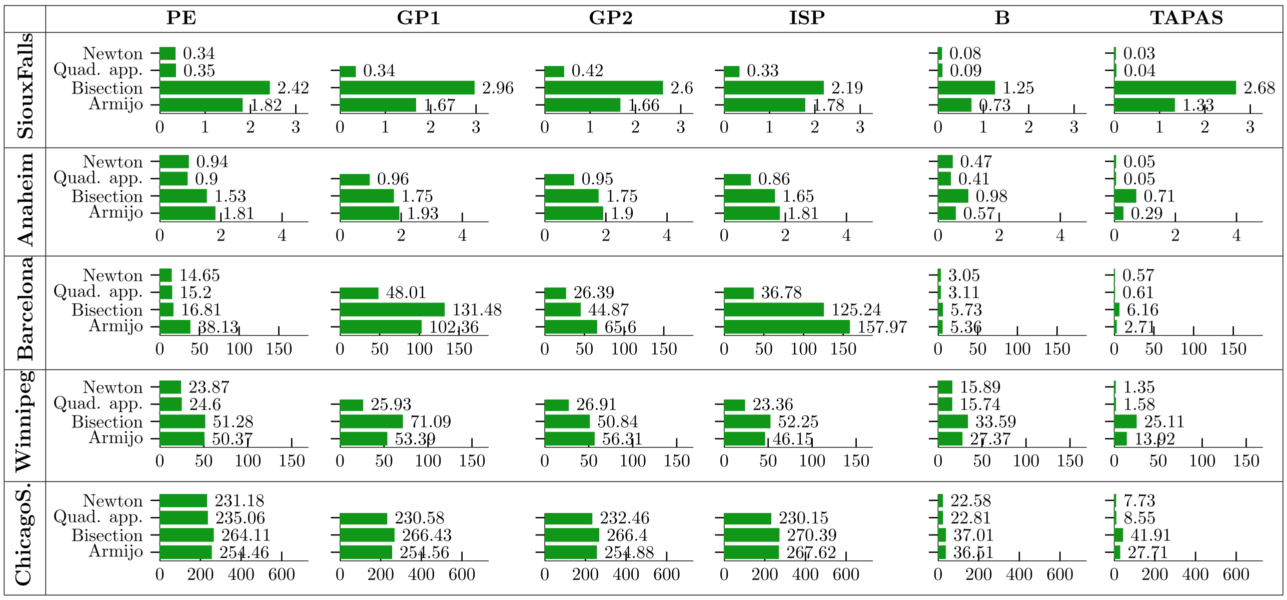


Fig. 6. Test 1. CPU time, s.

shows the worst performance compared to the other approaches. The time reported in Figs. 5 and 6 is *pure iteration time* – the time needed to perform the iterations of an algorithm, but it does not include any output operations or convergence check. For all the algorithms the time needed for object creation, output operations and convergence check is negligible compared to the pure iteration time. Because of this all conclusions are derived based only on pure iteration time.

After analysing different configurations of the algorithms we can conclude that for PE the best option is Newton step, for GP1 and GP2 it is quadratic approximation, for ISP – quadratic approximation, for algorithm B – Newton step, for TAPAS – Newton step. This observation regarding different configurations of the algorithms shows that if a line search is applied, it is better to approximate the step size using quadratic approximation. However, applying a Newton step is preferable to a line search, if it is possible. All path-based algorithms except GP Newton show a similar performance with the exception of the Barcelona instance, where PE is a better choice. GP Newton depends on an algorithm-specific constant that influences its performance and convergence [30]. In our study, this constant was set to a small enough value in order to achieve convergence for all instances. As a result of this choice, this particular configuration does not always perform well, but converges for all instances. A different choice of this constant for different instances can lead to better performance of GP Newton for those instances. In general, path-based algorithms are outperformed by bush-based algorithms represented by algorithm B and TAPAS. Overall, TAPAS with Newton step shows the best performance.

Considering only total running times needed for the algorithms to reach the required precision is not sufficient for the algorithm comparison since the convergence behaviour cannot be seen. Figs. 7–10 show the convergence of the algorithms on three selected instances with the best options identified in Figs. 5 and 6 (other figures of convergence behaviour can be obtained by request). CPU times reported on these figures consider only pure iteration time.

Let us analyse convergence patterns. On all instances, the algorithms converge fast to the level of relative gap around 10^{-6} and slow down after this point. TAPAS is an exception to this observation. Usually TAPAS requires more time during the first iterations compared to other algorithms and then converges faster. Initial iterations of TAPAS take more time because many PASs are created at this stage. For example, consider the Winnipeg instance. After the first iteration of TAPAS with Newton step 1288 new PASs were created, but after the fourth only 183 new PASs were created.

Several instances show specific convergence patterns. For example, on the Anaheim instance (Fig. 7), the majority of algorithms "stall" with the value of relative gap around 10^{-7} and after some time they suddenly improve performance significantly. Plateaus like this can be seen on other instances. However, they usually do not last for as long as for the Anaheim instance. After analysing in more detail what happens, we conclude that the situation is similar to "extreme coupling" discussed in Bar-Gera [37]. In this case, only small amounts of flow are moved between several paths and the algorithm does not significantly improve the current solution until a sufficient amount of flow has been shifted. An example can be found in Bar-Gera [37].

On the Winnipeg and Barcelona instances (Fig. 8) many algorithms show zigzag patterns when the value of relative gap is lower than 10^{-6}. These two instances use high powers in the BPR functions (for Barcelona $power \in [0, 16.83]$, for Winnipeg $power \in [0, 6.8677]$) compared to the other instances where *power* is bounded by 4. As pointed out in Spiess [52], the use of high powers in BPR functions is not recommended for two reasons:

1. During initial iterations of traffic assignment algorithms, many links can have high values of the ratio $f_a/capacity$. If the parameter *power* is also high, this might lead to numerical problems like overflow and loss of precision.
2. In the situation when the ratio $f_a/capacity$ is small, high values of *power* might cause the BPR function to become numerically not strictly increasing, i.e. it will return the same value for different link flows.

We investigated in detail algorithm GP1 with quadratic approximation on the Barcelona instance. In fact, if we change the tolerance of the direction of descent from 10^{-15} to 10^{-14} (it means that only 14 digits of the direction of descent are considered significant), the algorithm converges within 16 s compared to 48 s as demonstrated in Fig. 9. This fact means that the magnitude of error accumulated in the calculation of the direction of descent is more than 10^{-15}. This leads to an interesting conclusion that when high accuracy is required, the algorithms must be used with caution. In general, it is impossible to know the magnitude of error beforehand and adjust the algorithm accordingly. Long times of reaching a highly precise solution might be related to rounding errors and not to the performance of algorithms. However, some algorithms are more stable in terms of accumulated numerical errors than the others. This issue is discussed in Section 5.2.3.

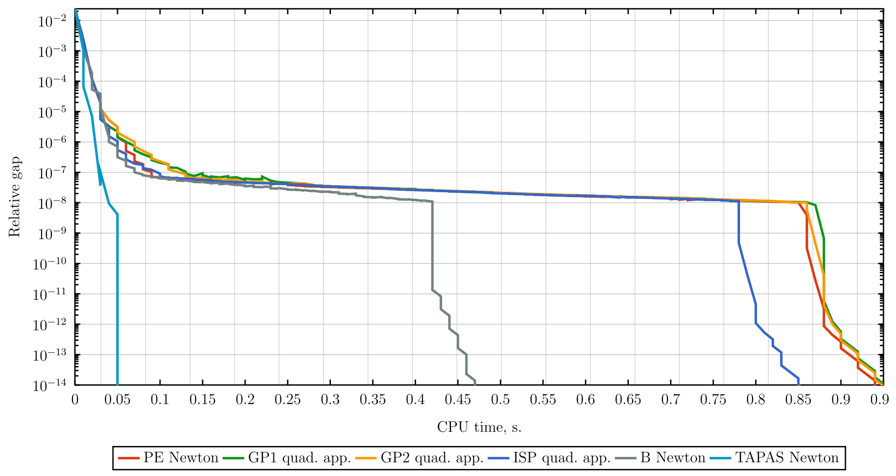


Fig. 7. Test 1. Convergence on the Anaheim instance.

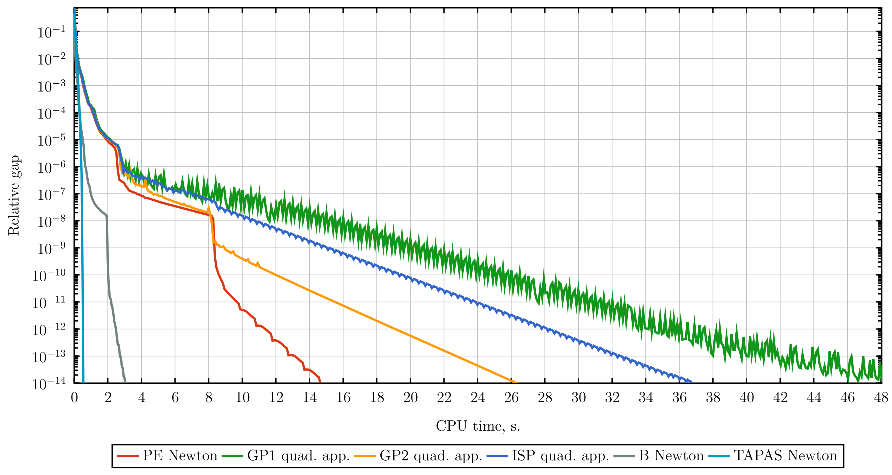


Fig. 8. Test 1. Convergence on the Barcelona instance.

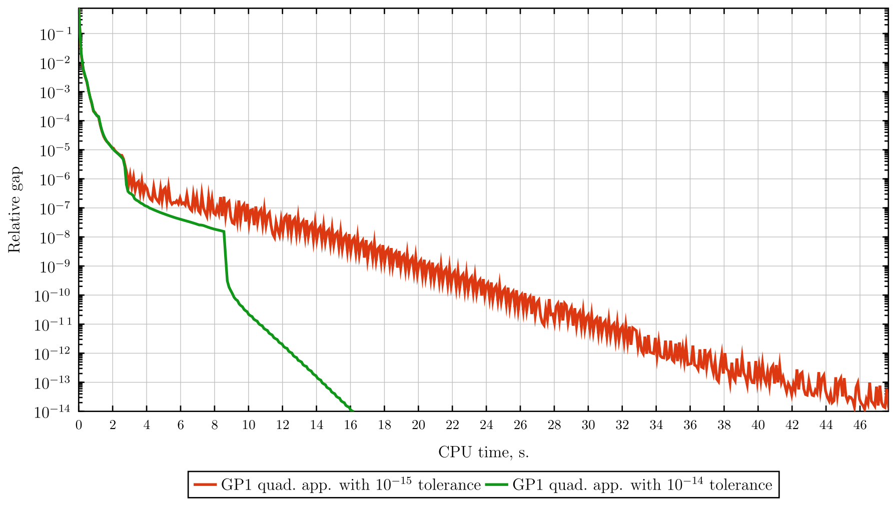


Fig. 9. Effect of change in tolerance of the direction of descent of GP1 on the Barcelona instance.

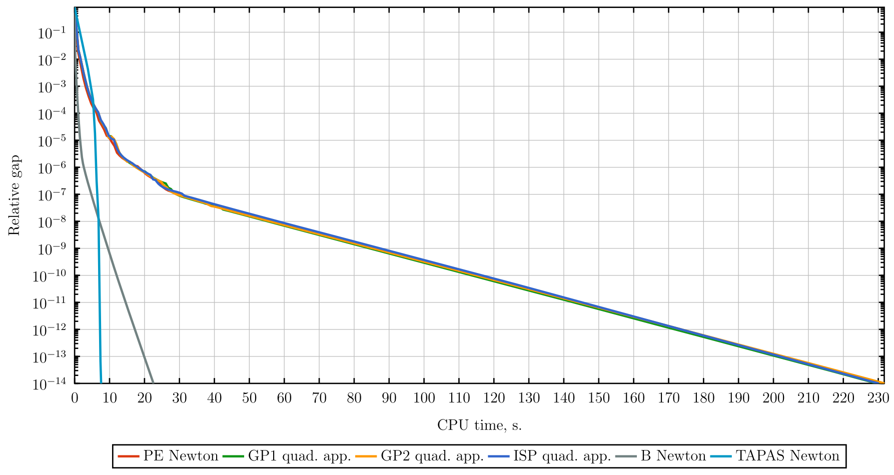


Fig. 10. Test 1. Convergence on the Chicago Sketch instance.

Bottom line:

1. All path-based algorithms show similar performance.
2. Algorithm B and TAPAS outperform path-based methods.
3. Overall TAPAS shows better performance than other approaches.
4. In the case of a line search, it is preferable to apply quadratic approximation, however, if Newton step is applicable it is preferable.
5. When a highly precise solution is required, the algorithms might experience instability because of the errors accumulated during computation.

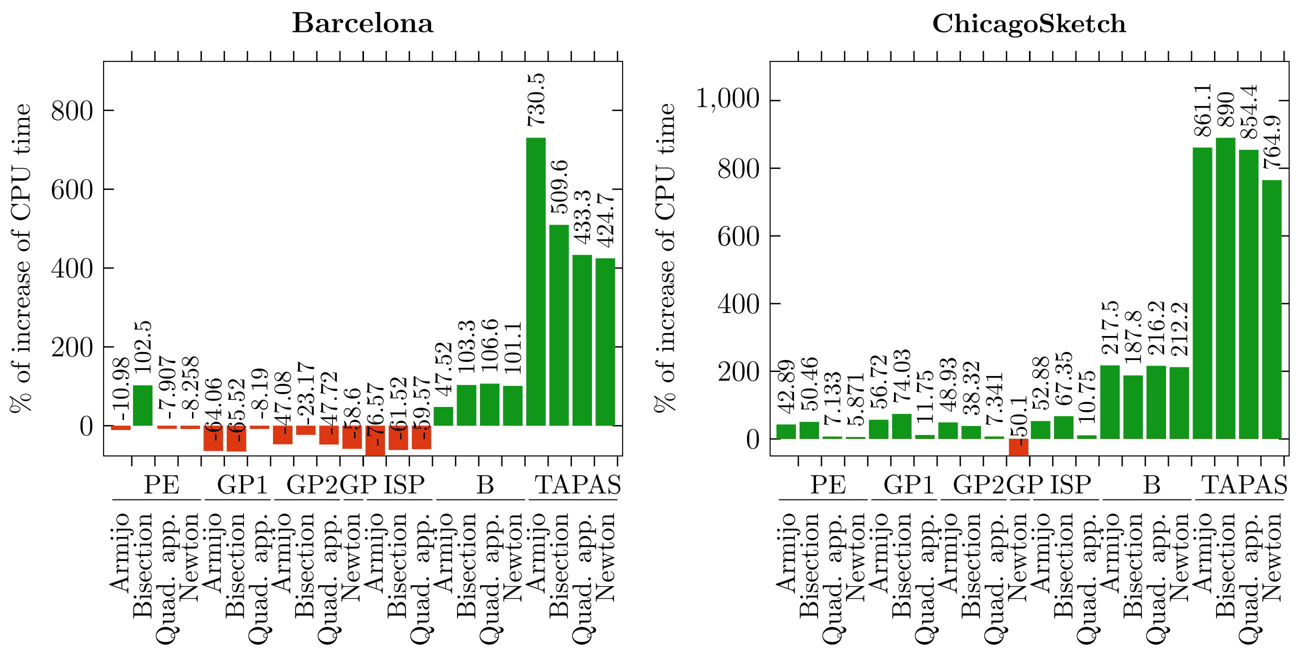


Fig. 11. Test 2. % of increase of CPU time compared to Equilibration I.

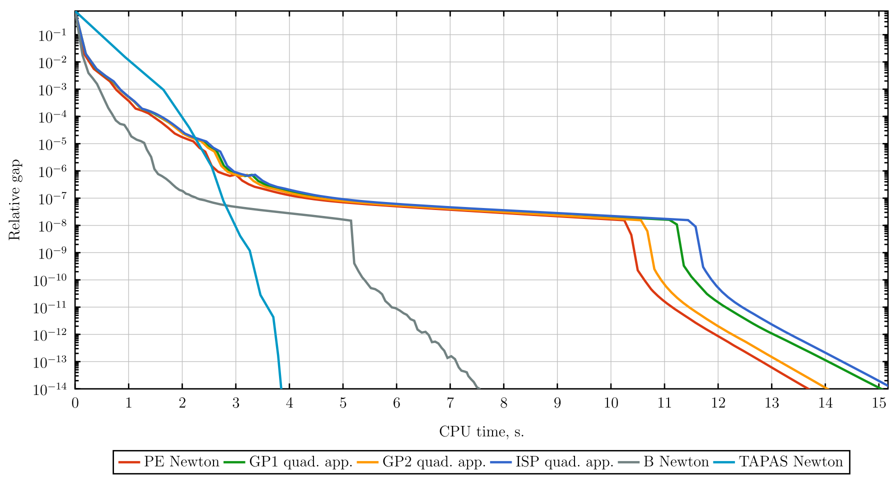


Fig. 12. Test 2. Convergence on the Barcelona instance.

5.2.2. *Test 2: Equilibration II*

This numerical test investigates the impact of Equilibration II (several steps of improving and equilibrating are applied to the element of decomposition under consideration before proceeding to the next one). This strategy was applied to the same algorithms as in Test 1.

Fig. 11 shows the percentage increase of CPU time when applying Equilibration II compared to Equilibration I

$$TimeInc_{EqII} = \frac{Time_{EqII} \cdot 100\%}{Time_{EqI}}, \tag{16}$$

where $Time_{EqI}$ and $Time_{EqII}$ represent CPU time (s) when Equilibration I and II are applied, respectively, on the Barcelona and Chicago Sketch instances (other figures can be provided on request). In general, Equilibration II leads to a significant increase of computational time with only a few exceptions: GP Newton performs better on all instances and the majority of algorithms perform better on the Barcelona instance with Equilibration II.

Fig. 12 presents convergence of the algorithms on the Barcelona instances. We observe that convergence patterns are much "smoother" compared to Equilibration I (Fig. 8). This is due to the fact that each element of decomposition is closer to an equilibrium since instead of just one equilibration step several steps are applied to a given element of decomposition before proceeding to the next one. This has a positive impact on relative gap since one iteration involves more flow shifts compared to Equilibration I. However, all other results of testing Equilibration II (Fig. 11) show that this strategy requires more time in order to reach the required precision.

It seems that Equilibration II is more stable to numerical errors. It can be seen from the Barcelona instance (Fig. 12) that algorithm GP1 with quadratic approximation performs much better than with Equilibration I (Fig. 8). This might be due to the fact that when one element of decomposition is first equilibrated to a given precision before proceeding to the next one, there is less chance that accumulated error will be put on this particular element from another one.

Bottom line:

1. Equilibration II takes more time in order to solve the problem to high precision, but it shows a smoother convergence pattern and is less prone to numerical errors.

5.2.3. *Test 3: time limit*

This numerical test considers the algorithms FW, CFW, BFW, PG and LUCE. These methods were not able to achieve the required high precision of relative gap of 10^{-14} in a reasonable amount of time. Therefore, a time limit was used as an additional stopping criterion. For each instance, the time limit was fixed as follows: we chose the longest time needed for the algorithms from Tests 1 and 2 to converge on a given instance and multiplied it by three.

Fig. 13 shows the convergence patterns of the algorithms on the Winnipeg instance (similar convergence patterns occur on other instances). Among all link-based approaches, BFW always demonstrates the best performance. Quadratic approximation seems to be the best line search strategy. However, when the algorithm starts tailing, it is difficult to say which line search performs better.

FW shows a zigzag pattern when the level of relative gap reaches 10^{-5}. This happens because the search direction is perpendicular to the gradient of the objective function when the algorithm approaches the equilibrium solution [13,20,29]. As mentioned in Bertsekas [41], when the constraint set is polyhedral, as is the case with TA, FW might converge sub-linearly. It seems that the conjugate versions of FW suffer from the same problems even though in general they are able to achieve a higher level of precision. This might be partially explained by the effect of loss of conjugacy when the method "jams" after several iterations [41].

LUCE converges fast in the beginning. However, it does not improve the solution when the value of relative gap approaches 10^{-10}–10^{-12} (the value is instance-dependant). A similar behaviour of this algorithm is discussed in Inoue and Maruyama [9] and in more detail in Xie et al. [47].

As explained in Xie et al. [47], the convergence issues of LUCE are related to the calculation of the direction of descent. Xie et al. [47] compared LUCE and OBA which are similar bush-based methods. The only difference consists in the way a quadratic sub-problem is solved and how line search is applied. Xie et al. [47] report that OBA is able to reach high precision whereas LUCE does not improve the solution after a certain level of relative gap is reached. OBA applies an approach similar to GP (however, it is applied to a link-based formulation, not to a path-based one as is the case for GP), whereas LUCE applies a greedy algorithm that uses average costs. Xie et al. [47] explain that the quadratic subproblem itself is just an approximation (the closed form second-order derivatives are unknown for the formulation of TA used in OBA and LUCE). As a result, the second-order derivatives contain errors. OBA manages to "correct" its direction of descent by directly embedding a line search into the calculation of the direction of descent. LUCE, however, does not correct its direction of descent which causes the issues with convergence. Xie et al. [47] do not find a way to address this particular problem for LUCE.

As can be seen from Fig. 13, the line of convergence of PG stops before reaching the time limit. This happens because this algorithm demonstrates unstable behaviour when the required precision is high. In particular, the problem becomes infeasible (flow conservation constraints are not satisfied) when the relative gap approaches 10^{-7}. We examined in more detail what causes such behaviour.

Let us consider a small example presented in Fig. 14 that is part of the Sioux Falls instance. We consider O–D pair 13–15. After several iterations two paths were identified and equilibrated to a certain precision: path l contains links $a_{[13,24]}$, $a_{[24,21]}$, $a_{[21,22]}$ and $a_{[22,15]}$ and path s contains links $a_{[13,24]}$, $a_{[24,23]}$, $a_{[23,22]}$ and $a_{[22,15]}$. The cost difference of these paths is $C_l - C_s = 43.708039514 \times 10^{-15}$. This cost difference seems to be insignificant. However, in order to achieve precision of 10^{-14} we must take small path cost differences into account. If we now calculate the average path cost of these two paths and calculate the corresponding direction of descent, we will get: $d_l = -2.35575448 \times 10^{-15}$, $d_s = 2.352285033 \times 10^{-15}$. In the case of two paths, the following must hold $d_l + d_s = 0$. However, because of rounding errors we have that $d_l + d_s = -3.469446952 \times 10^{-18} \neq 0$. It seems that the error is too small and cannot impact the computation. However, the next step of the algorithm is to multiply the direction of descent by a step size that might be a large number. For this example the step size is $\lambda = 3.065555135 \times 10^{15}$. As a result,

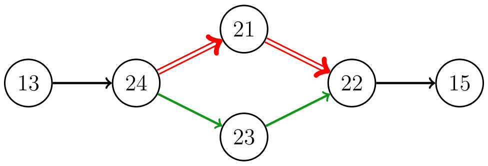


Fig. 14. Example network.

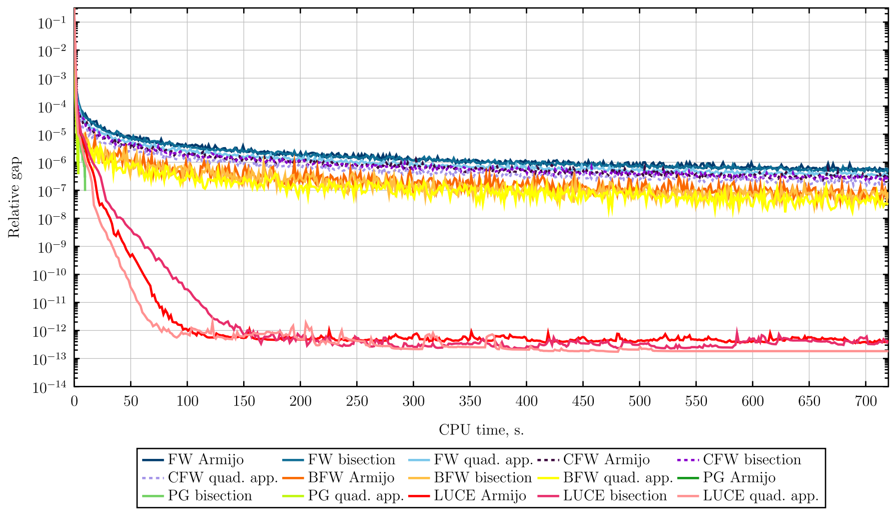


Fig. 13. Test 3. Convergence on the Winnipeg instance.

the error is multiplied by a large step size and some amount of flow is lost. In our example the flow conservation constraint is $F_l+\lambda \cdot F_l+F_s+\lambda \cdot F_s = 699.9893642 \neq 700$ where 700 is the demand of O–D pair 13–15. Hence, the amount of flow lost is 0.010635. This loss is actually significant and it is accumulated more during subsequent iterations.

Such a situation is more likely to occur when a high level of accuracy is required. When the algorithm is close to the equilibrium solution, the difference between path costs is very small. Thus, when the two similar numbers are subtracted in order to find a direction of descent, the precision is lost. This effect is called cancellation [53]. It seems that small path cost differences must be simply avoided. However, as mentioned earlier, small path cost differences must be taken into account in order to achieve a highly precise solution. For example, for the Sioux Falls instance, if we consider only the differences of order larger than 10^{-13}, infeasibility still occurs. However, if we increase this value to 10^{-12}, the algorithm is not able to reach a relative gap lower than 10^{-11}. This means that other techniques must be applied in order to solve this issue.

The natural question is why this situation does not occur for other path-based algorithms that we tested. In fact, they all "correct" their directions of descent which are constructed in such a way that the sum over all elements of the direction of descent is equal to zero or is a subnormal number.[4] In particular, we have that

1. PE: $d_l = C_l - C_s$ and $d_s = -d_l$, hence $d_l + d_s = 0$.
2. GP: all elements of the direction of descent are calculated in a certain way (see Section 4.2.2), but then the element corresponding to the current shortest path is corrected as follows $d_s = -\sum_{k \in K_p^+, k \neq s} d_k$. Therefore, the sum over all elements of the direction of descent is zero.
3. ISP: the elements of the direction of descent with cheaper costs are corrected as presented in Eq. (10). The sum over all elements of the direction of descent is a subnormal number less than 10^{-30} which is sufficient to eliminate the error when it is multiplied by a large step size.

It might seem that the convergence issues of LUCE are similar to those of PG. However, this is not the case. LUCE does not have a mechanism to correct its direction of descent in a similar way as PE, GP or ISP do. But for LUCE the step size is bounded by one. As a result, insignificant error that occurs during the calculation of the direction of descent is not multiplied by a large number and propagated further. However, the straight-forward implementation of LUCE that does not take into account accumulation of the error might also face the same problem that happens in PG when flow conservation constraints are not satisfied. Since the step size is bounded by one, this situation might occur during later stages compared to PG. In particular, for PG the solution becomes infeasible when relative gap is around 10^{-7}, for LUCE a quite significant amount of flow of order 10^{-9} is lost when relative gap approaches 10^{-11}.

These issues raise a question of numerical stability of the TA algorithms and their ability to find highly precise solutions. It is important to take into account the situation when the step size might be a large number causing the increase of rounding errors. The algorithms that correct their direction of descent are less prone to numerical errors even when the required precision is high. If the algorithm does not incorporate techniques for maintaining the number of significant digits of the direction of descent to be high enough, the solution might become infeasible. The incorporation of such techniques into the PG algorithm is the subject of our future research.

Bottom line:

1. Link-based algorithms represented by FW, CFW and BFW converge fast during the initial iterations, but start tailing in the vicinity of the equilibrium and cannot achieve highly precise solutions in a reasonable amount of time.
2. LUCE can achieve relatively high precision of around 10^{-12}. However, after a certain level it does not further improve the solution.
3. PG suffers from numerical errors and is not able to achieve highly precise solutions.

5.2.4. Test 4: large instances

Test 4 aims to compare the best algorithms on large instances (the three last instances in Table 3). For this test, we chose the algorithms BFW, GP2, PG, ISP and LUCE with quadratic approximation, PE, B and TAPAS with Newton step. CFW and FW are not present in this test since they always perform worse compared to BFW. In addition to relative gap, a time limit of 24 h was imposed as a stopping criterion.

Figs. 15–17 show convergence patterns of the algorithms on different instances. Let us examine them in more detail. The PG algorithm stops due to accumulated numerical error before reaching the time limit as in the case of small and medium instances for the same reasons discussed in Section 5.2.3. On the PRISM instance, all algorithms except BFW and LUCE reach the required level of precision in less than 24 h. TAPAS followed by algorithm B demonstrates the best performance as in the case of Tests 1 and 2. The Philadelphia instance, however, shows a different pattern. Only TAPAS and B reach the required precision. Moreover, B demonstrates the best performance among the algorithms. LUCE is able to achieve higher precision than any of the path-based approaches. A similar situation occurs on the Chicago Regional instance with the only exception regarding path-based algorithms which show worse performance compared to BFW. Results obtained for the Philadelphia and Chicago Regional instances contradict the results regarding TAPAS presented in Inoue and Maruyama [9]. The reason of worse performance of TAPAS in our implementation might be related to the fact that we did not randomise TAPAS and did not implement the path proportionality condition. As mentioned in Xie et al. [47], management of PASs complicates design and implementation of TAPAS and might be crucial to its performance. Improvement of this algorithmic step is subject of our future research.

Bottom line:

1. For large instances, TAPAS and algorithm B are usually the best choice if a highly accurate solution is required.
2. A careful implementation of TAPAS is necessary in order to achieve the best results.

5.3. Summary

This section summarises the main findings from the performed numerical tests. We conclude that different line search strategies have a significant impact on the performance of the algorithms. Methods like Newton step and quadratic approximation exploit the information about link cost function derivatives which enables them to achieve the same results as bisection or the Armijo-rule can produce, but in less time.

[4] Subnormal numbers fill the underflow gap around zero in floating-point arithmetic. Any non-zero number with magnitude smaller than the smallest normal number is subnormal [53].

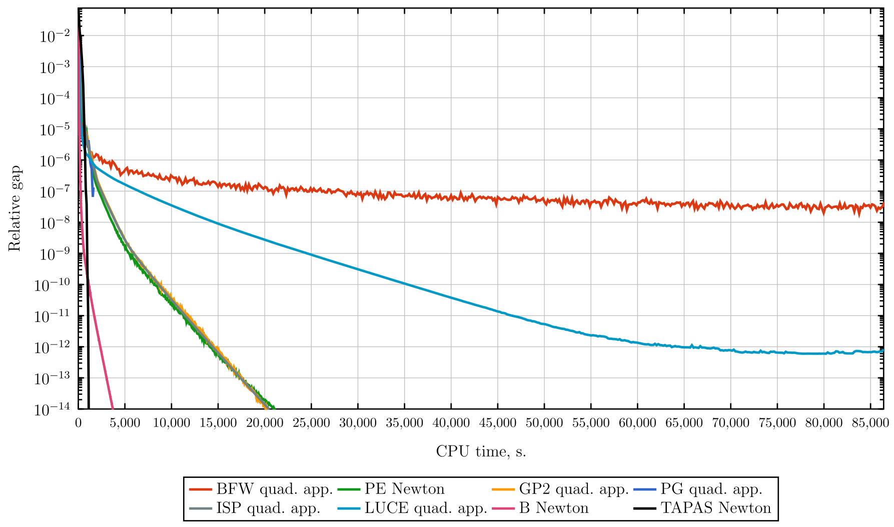


Fig. 15. Test 4. Convergence on the PRISM instance.

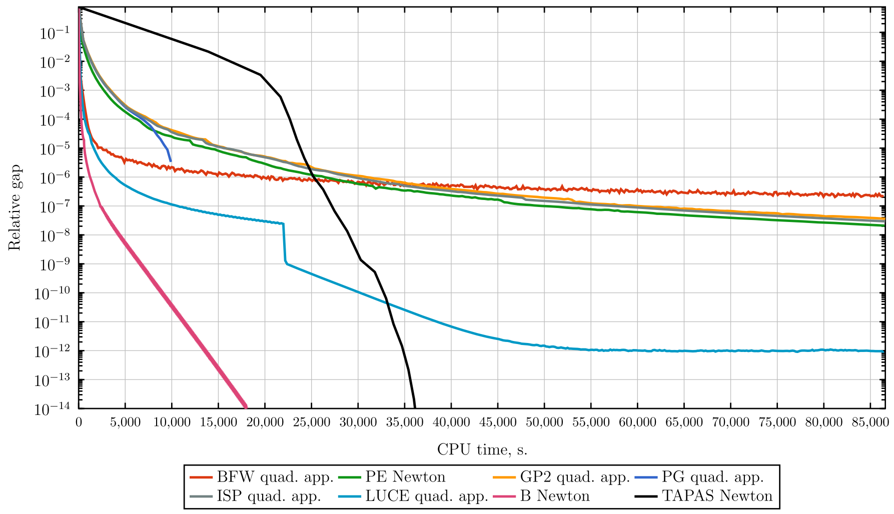


Fig. 16. Test 4. Convergence on the Philadelphia instance.

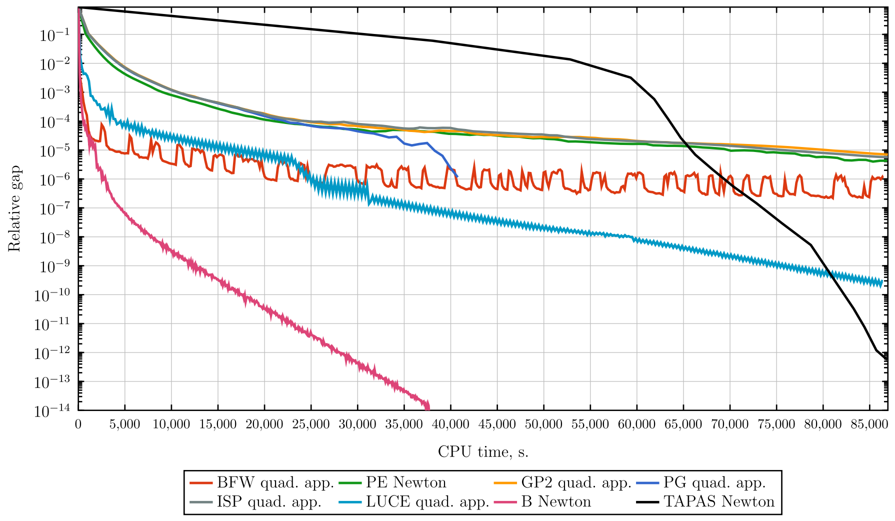


Fig. 17. Test 4. Convergence on the Chicago Regional instance.

Other key factors that impact the performance of the algorithms are related to numerical precision issues that play an important role when the required precision is high. In order to prevent the solution from becoming infeasible, the direction of descent must be corrected in such a way that a sufficient number of significant digits is maintained.

Algorithms similar to LUCE, whose sub-problems are based on a single origin TA formulation with flow portion variables, might not be able to reach high accuracy due to the errors in second-order derivatives. This issue underlines the fact that it is better to apply methods that are based on exact second-derivative information which is the case for all path-based methods and bush-based methods that shift flows between segments (B and TAPAS).

Bush-based methods usually outperform path- and link-based approaches. LUCE is an exception to this: sometimes LUCE demonstrates slower convergence compared to path-based algorithms. One of the main advantages of bush-based approaches is the fact that all bushes represent directed acyclic graphs. As a result, shortest path calculations can be performed much faster compared to other methods. An interesting exception to this fact is TAPAS which does not maintain bushes and requires single-source shortest path calculations on general directed graphs. In our opinion, the following feature of TAPAS enables its outstanding performance. For TAPAS, segment costs are directly equalised (not path cost or segment costs related to a particular bush). First, this enables the algorithm to achieve high accuracy. For example, if one considers the segment costs achieved after applying path-based algorithms, they are not as precisely equalised as in the case of TAPAS even though they correspond to paths of equal costs. It is a good example of when too much decomposition is not a good approach because equal path costs do not imply the same level of precision for path segments that aggregate these paths. Second, one flow move within a given PAS usually affects several origins. It is definitely a better strategy than making several separate flow moves for different origins or O–D pairs.

Algorithms also differ in the amount of information generated along with link flows. Path-based algorithms can provide path flows. However, since they are not unique, such information might be misleading. TAPAS aims at addressing this difficulty by providing consistent path flows that satisfy the condition of proportionality. Bush-based methods can also provide path flow information. However, as in the case of path-based approaches, additional methods must be applied in order to guarantee the uniqueness of such flows.

Another important feature of the algorithms is the amount of memory required that varies from low for link-based approaches to medium (bush-based methods) to high (path-based approaches). Regarding the difficulty of implementation, bush-based methods and TAPAS in particular require much more time and effort.

6. Conclusion and future work

This work starts with a literature overview of existing traffic assignment algorithms, based on which we selected and implemented the most promising ones. The main aim of our numerical study is to compare the algorithms under the same computational environment and in terms of a framework that ensures that common code is used wherever possible. We implemented and analysed several algorithms belonging to link-, path- and bush-based methods, namely FW, CFW, BFW, PE, GP, PG, ISP, B, LUCE and TAPAS.

Based on the performed computational study we can conclude the following. When choosing what algorithm to implement for solving the TA problem, the level of solution accuracy must be one of the most important factors to be taken into account. If the required precision is low or medium, simple algorithms represent a better choice. Path-based methods as well as BFW fall into this category and can achieve medium precision levels in a reasonable amount of time. For large instances, BFW represents a good trade-off between memory consumption and solution accuracy. For achieving highly precise solutions, advanced approaches like algorithm B and TAPAS should be implemented. Both algorithms show the ability to achieve high accuracy in a reasonable amount of time for instances of different sizes varying from small to large.

The future development of this research consists in further study of numerical issues when high accuracy is required. In particular, we want to investigate how the direction of descent of the PG algorithm can be corrected. It will also be interesting to investigate the impact of randomisation and PAS management strategies on the performance of TAPAS.

Acknowledgements

This research has been partially supported by the Marsden Fund under Grant number UOA1008, by the European Union Seventh Framework Programme (FP7-PEOPLE-2009-IRSES) under Grant agreement 246647 and by the New Zealand Government (Grant no. 649378 v2) as part of the Optimization and its Applications in Learning and Industry project.

We would like to thank Amit Kumar, Srinivas Peeta, Guido Gentile, Hillel Bar-Gera for helpful discussions on different traffic assignment algorithms, Michiel Bliemer for providing many insights on algorithm classification, Boshen Chen for providing an efficient A* implementation, and the reviewer of this paper for valuable comments and references.

References

[1] Ortúzar JD, Willumsen L. Modelling transport. Chichester, New York: J. Wiley; 2001.
[2] Wardrop J. Some theoretical aspects of road traffic research. London, UK: Institution of Civil Engineers; 1952.
[3] Ryu S, Chen A, Choi K. A modified gradient projection algorithm for solving the elastic demand traffic assignment problem. Comput Oper Res 2014;47:61–71.
[4] Carey M, McCartney M. An exit-flow model used in dynamic traffic assignment. Comput Oper Res 2004;31(10):1583–602.
[5] Davies J, Valero J, Young D. The Auckland transport models project: overview and use to date. In: Proceedings of the Australasian transport research forum 2009; 2009. p. 1–4.
[6] Sheffi Y. Urban transportation networks: equilibrium analysis with mathematical programming methods. Englewood Cliffs, NJ: Prentice-Hall; 1985.
[7] Slavin H, Ricotta P, Brandon J, Rabinowicz A, Sundaram S. A new traffic assignment method for small and medium communities. In: Tools of the trade conference; 2010. p. 1–11.
[8] Gentile G. Local user cost equilibrium: a bush-based algorithm for traffic assignment. Transportmetrica A: Transp Sci 2014;10(1):15–54.
[9] Inoue S, Maruyama T. Computational experience on advanced algorithms for user equilibrium traffic assignment problem and its convergence error. Proc—Soc Behav Sci 2012;43:445–56 Eighth international conference on traffic and transportation studies (ICTTS 2012).
[10] Florian M, Constantin I, Florian D. A new look at projected gradient method for equilibrium assignment. Transp Res Record: J Transp Res Board 2009;2090:10–6.
[11] Perederieieva O, Ehrgott M, Wang JYT, Raith A. A computational study of traffic assignment algorithms. In: Australasian transport research forum.1C:2; 2013. p. 1–18.
[12] Florian M, Hearn D. Network equilibrium models and algorithms. In: Ball, MO Magnanti TL, Monma CL, and Nemhauser GL, editors. Network routing. Handbooks in operations research and management science, vol. 8. Elsevier, Amsterdam; 1995. p. 485–550 [chapter 6].
[13] Patriksson M. The traffic assignment problem: models and methods. Top Transp 1994.
[14] Bertsekas D. Network optimization continuous and discrete models. Belmont, Massachusetts: Athena Scientific; 1998.
[15] Zhou Z, Martimo M. Computational study of alternative methods for static traffic equilibrium assignment. In: Proceedings of the 12th world conference on transport research (WCTRS), Lisbon, Portugal; 2010. p. 1–15.
[16] Frank M, Wolfe P. An algorithm for quadratic programming. Naval Res Logist Quart 1956;3(1–2):95–110.
[17] Lee DH, Nie Y, Chen A, Leow YC. Link- and path-based traffic assignment algorithms: computational and statistical study. Transp Res Rec: J Transp Res Board 2002;1783:80–8.
[18] Fukushima M. A modified Frank–Wolfe algorithm for solving the traffic assignment problem. Transp Res Part B: Methodol 1984;18(2):169–77.

[19] Florian M, Guélat J, Spiess H. An efficient implementation of the PARTAN variant of the linear approximation method for the network equilibrium problem. Networks 1987;17:319–39.
[20] Mitradjieva M, Lindberg PO. The stiff is moving—conjugate direction Frank–Wolfe methods with applications to traffic assignment. Transp Sci 2013;47(2):280–93.
[21] Powell WB, Sheffi Y. The convergence of equilibrium algorithms with predetermined step sizes. Transp Sci 1982;16(1):45–55.
[22] Weintraub A, Ortiz C, Gonzàlez J. Accelerating convergence of the Frank–Wolfe algorithm. Transp Res Part B: Methodol 1985;19(2):113–22.
[23] Hearn DW, Lawphongpanich S, Ventura JA. Finiteness in restricted simplicial decomposition. Oper Res Lett 1985;4(3):125–30.
[24] Larsson T, Patriksson M, Rydergren C. Applications of simplicial decomposition with nonlinear column generation to nonlinear network flows. In: Pardalos P, Hearn D, Hager W, editors. Network optimization. Lecture notes in economics and mathematical systems, vol. 450. Berlin, Heidelberg: Springer; 1997. p. 346–73.
[25] Dafermos S.-S.C., Traffic assignment and resource allocation in transportation networks, PhD thesis, Johns Hopkins University, Baltimore, MD, 1968.
[26] Gibert A. A method for the traffic assignment problem. Technical report. LBS-TNT-95. London Business School, London, UK; 1968.
[27] Leventhal T, Nemhauser G, Trotter JL. A column generation algorithm for optimal traffic assignment. Transp Sci 1973;7(2):168–76.
[28] Larsson T, Patriksson M. Simplicial decomposition with disaggregated representation for the traffic assignment problem. Transp Sci 1992;26:4–17.
[29] Jayakrishnan R, Tsai WK, Prashker J, Rajadhyaksha S. A faster path-based algorithm for traffic assignment. Transp Res Rec 1994;1443:75–83.
[30] Chen A, Jayakrishnan R. A path-based gradient projection algorithm: effects of equilibration with a restricted path set under two flow update policies. In: Transportation research board annual meeting; 1998. p. 1–19.
[31] Kumar A, Peeta S. An improved social pressure algorithm for the static deterministic user equilibrium traffic assignment problem. Washington, DC: Transportation Research Board of the National Academics; 2011.
[32] Nie Y. A class of bush-based algorithms for the traffic assignment problem. Transp Res Part B: Methodol 2010;44(1):73–89.
[33] Bar-Gera H. Origin-based algorithm for the traffic assignment problem. Transp Sci 2002;36(4):398–417.
[34] Nie Y. A note on Bar-Gera's algorithm for the origin-based traffic assignment problem. Transp Sci 2012;46(1):27–38.
[35] Dial R. A path-based user-equilibrium traffic assignment algorithm that obviates path storage and enumeration. Transp Res Part B: Methodol 2006;40(10):917–36.
[36] Zhang T, Yang C, Chen D. Modified origin-based algorithm for traffic equilibrium assignment problems. J Cent South Univ Technol 2011;18(5):1765–72.
[37] Bar-Gera H. Traffic assignment by paired alternative segments. Transp Res Part B: Methodol 2010;44(8–9):1022–46.
[38] Chen A, Lee DH, Jayakrishnan R. Computational study of state-of-the-art path-based traffic assignment algorithms. Math Comput Simul 2002;59(6):509–18.
[39] Slavin H, Brandon J, Rabinowicz A. An empirical comparison of alternative user equilibrium traffic assignment methods. In: Proceedings of the European transport conference. Strasbourg, France: Association for European Transport; 2006.
[40] Andréasson N, Evgrafov A, Patriksson M. An introduction to continuous optimization. Studentlitteratur AB; 2007.
[41] Bertsekas DP. Nonlinear programming. 2nd ed.Athena Scientific; 1999.
[42] Chen A, Zhou Z, Xu X. A self-adaptive gradient projection algorithm for the nonadditive traffic equilibrium problem. Comput Oper Res 2012;39(2):127–38.
[43] Chen A, Xu X, Ryu S, Zhou Z. A self-adaptive Armijo stepsize strategy with application to traffic assignment models and algorithms. Transportmetrica A: Transp Sci 2013;9(8):695–712.
[44] Lee DH, Nie Y, Chen A. A conjugate gradient projection algorithm for the traffic assignment problem. Math Comput Model 2003;37(7–8):863–78.
[45] Cheng L, Xu X, Qiu S. Constrained Newton methods for transport network equilibrium analysis. Tsinghua Sci Technol 2009;14(6):765–75.
[46] Dasgupta S, Papadimitriou C, Vazirani U. Algorithms. McGraw-Hill, New York: McGraw-Hill Education (India) Pvt Limited; 2006.
[47] Xie J, Nie Y, Yang X. Quadratic approximation and convergence of some bush-based algorithms for the traffic assignment problem. Transp Res Part B: Methodol 2013;56(0):15–30.
[48] Goldberg AV, Kaplan H, Werneck RF. Reach for A: efficient point-to-point shortest path algorithms. In: SIAM workshop on algorithms engineering and experimentation. MSR-TR-2005-132. Microsoft Research. Miami, FL: Society for Industrial and Applied Mathematics; 2006. p. 129–43.
[49] Nagurney A. Network economics: a variational inequality approach, advances in computational economics. Dordrecht, The Netherlands: Springer, Kluwer Academic Publishers; 1998.
[50] Holmgren J, Lindberg P. Upright stiff: subproblem updating in the FW method for traffic assignment. EURO J Transp Logist 2013:1–21.
[51] Chen A, Jayakrishnan R, Tsai W. Faster Frank–Wolfe traffic assignment algorithm: a new flow update scheme. ASCE J Transp Eng 2002;128(1):31–9.
[52] Spiess H. Conical volume-delay functions. Transp Sci 1990;24(2):153–8.
[53] Overton ML. Numerical computing with IEEE floating point arithmetic: including one theorem, one rule of thumb, and one hundred and one exercises. Soc Ind Appl Math 2001.