Numerical Stability of Path-based Algorithms For Traffic Assignment

Olga Perederieieva, Matthias Ehrgott, Andrea Raith, Judith Y. T. Wang

June 12, 2014

Abstract

In this paper we study numerical stability of path-based algorithms for the traffic assignment problem. This type of methods are based on decomposition of the original problem into smaller sub-problems which are optimised sequentially. Previously, path-based algorithms were numerically tested only in the setting of moderate requirements to the level of solution precision. In this study we analyse convergence of these methods when convergence measure approaches machine epsilon of IEEE double precision format. In particular, we demonstrate that the straightforward implementation of one of the algorithms of this group (projected gradient) suffers from loss of precision and is not able to converge to highly precise solution. We propose a way to solve this problem and test the proposed fixed version of the algorithm on various benchmark instances.

Keywords: traffic assignment, path-based algorithms, convergence, numerical stability, floating point arithmetic.

1 Introduction

Various traffic assignment (TA) models are used in practice by transportation engineers and urban planners. These models allow to analyse of road networks, to predict impact of potential projects and policies and to control traffic: level of congestion, emission, toll revenue etc (Ortúzar and Willumsen, 2001). These models aim to predict how road users will decide to travel during a given period of time (usually morning and evening peaks are of interest). The task of predicting how a particular individual will travel is not trivial. It is addressed by making certain assumptions on how people usually make their route choices. The classical TA model assumes that drivers travel on their fastest (or shortest) paths and network equilibrium occurs when no traveller can decrease their travel time by shifting to a new path. This assumption is called the *user equilibrium condition* or Wardrop's first principle (Sheffi, 1985).

The key feature of TA models consists in taking into account congestion effects that occur if capacities of some roads are exceeded. In order to consider congestion, so-called link cost functions are used. They represent a travel time through a given link of a network depending on traffic flow on that particular link. Thus, given the transportation network, the number of drivers travelling, their origins and destinations and the link cost functions, the goal is to assign traffic flows to links of the road network in such a way that the user equilibrium condition is satisfied.

In several papers authors discuss importance of highly precise solutions for the traffic assignment problem. High precision is required for select link analysis and for consistent comparison between design scenarios (Slavin et al., 2010; Gentile, 2014). 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 (Ortúzar and Willumsen, 2001). Bar-Gera et al. (2013) explain that TA is hierarchically embedded in a model for network design and road pricing. Level of precision of TA in this case has a direct impact on precision of the corresponding master problem. This motivates researchers to propose algorithms capable to find accurate solutions in a reasonable amount of time.

While studying various algorithms for solving the traffic assignment problem, we noticed that one of the methods from our study was not able to reach the required level of precision (Perederieieva et al., 2013, 2014). In this paper we investigate reasons of this behaviour and propose a way to fix the algorithm.

This paper is organised as follows. Section 2 introduces preliminary definitions from floating point arithmetic. Section 3 presents the traffic assignment problem and the algorithms considered in this study. In Section 4 we discus floating point issues that may arise and how to avoid them. The numerical results are presented in Section 5. Finally, Section 6 summarises our findings.

2 Preliminary Definitions

For the matter of completeness, this Section introduces some definitions from floating point arithmetic and numerical computing. In the following, all discussions of computations involve real numbers or their computer-based counterparts – floating point numbers.

There are several different sources of approximation and errors that occur when we want to solve a numerical problem. Heath (1996) divides them into two main categories: the ones that happen before computation and the ones that occur during computation. The first group of approximations include errors that come from modelling (simplifications, assumptions, etc.), empirical measurements (finite precision of instruments) and previous computations that might have been approximate. The second group of approximations consists of errors coming from truncation or discretization (for example, when a derivative of a function is replaced with a difference quotient) and rounding (finite computer precision arithmetic). Because of different errors and approximations, the solution to a given computational problem is only approximate. *Accuracy of solution* is closeness of a computed solution to the true solution of the problem (Heath, 1996).

From the first sight, it seems that accuracy of solution depends solely on the errors that were incorporated into the model on different stages presented above. However, accuracy of solution of a computation problem also depends on the problem itself which is defined by conditioning of the problem (Heath, 1996). A problem is *well-conditioned* if a given relative change in the input data results in a reasonably commensurate relative change in the solution (Heath, 1996). If the relative change in the solution can be much larger than that in the input data, a problem is *ill-conditioned* (Heath, 1996).

It is easier to understand this concept on the example of finding a root of non-linear function $f(x)$. This example is taken from Stewart (1996). Because of the finite computer precision, function $f(x)$ can be implemented on computer only approximately by:

$$\hat{f}(x) = f(x) + e(x), \tag{1}$$

where $e(x)$ is error. Let us assume that we know bound ϵ on the size of the error, i.e.

$$|e(x)| \leq \epsilon. \tag{2}$$

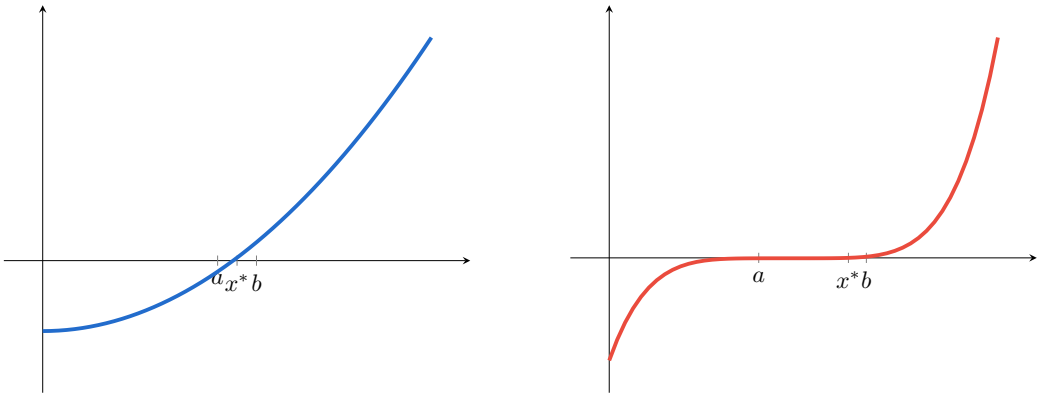
If x_1 is a point such that $f(x_1) > \epsilon$, then we have

$$\hat{f}(x_1) = f(x_1) + e(x_1) \geq f(x_1) - \epsilon > 0. \tag{3}$$

This means that $\hat{f}(x_1)$ has the same sign as $f(x_1)$. If $f(x_2) < -\epsilon$, then using the same reasoning we can show that $\hat{f}(x_2)$ is also negative as $f(x_2)$. It means that $f(x)$ has a zero between x_1 and x_2. Therefore, when $|f(x)| > \epsilon$, the values of $\hat{f}(x)$ are *meaningful* and allow to find a root of $f(x)$. However, the values of x for which $|f(x)| \leq \epsilon$, can result in $\hat{f}(x)$ being positive, negative or zero regardless of the sign of $f(x)$. Such values do not provide any useful information about the location of the root. Let $[a, b]$ be the largest interval about x^* (exact root of $f(x)$) for which

$$x \in [a, b] \implies f(x) \leq \epsilon. \tag{4}$$

This interval is called *interval of uncertainty*. If it is small (Figure 1a), then the problem is well-conditioned and a good algorithm will return a point in $[a, b]$. However, we cannot expect any further accuracy. If interval $[a, b]$ is large (Figure 1b), then the problem is ill-conditioned and no algorithm can be expected to provide an accurate solution to such a problem.



(a) Well-conditioned (b) Ill-conditioned

Figure 1: Conditioning of roots of non-linear equations

If the problem is well-conditioned, algorithm should return a reasonably accurate solution. Such an algorithm is called *stable*[1]. However, some algorithms can have intermediate steps that introduce ill-conditioning causing the solution to diverge (Overton, 2001). Such algorithms are called *unstable*. Robert Sedgewick and Kevin Daniel Wayne summarise these concepts as follows: *"Some algorithms are unsuitable for floating-point computation. Some problems are unsuitable to floating-point computation"* .

What can cause introduction of ill-conditioned steps into the algorithm that tries to solve a well-condition problem? Floating point arithmetic can answer this question. First, let us remind how real numbers are represented in computer. Any number x can be written in the following form with base 2:

$$x = \pm S \cdot 2^E, \quad 1 \leq S < 2, \tag{5}$$

where S is called *significant* and E is *exponent* (Overton, 2001). The binary expansion of S is

$$S = (b_0.b_1b_2b_3...)_2, \tag{6}$$

where b_0 is a hidden bit, $b_1b_2b_3...$ is a fractional part of the significant. For example, the number $11/2$ can be expressed as $(1.011)_2 \cdot 2^2$ (Overton, 2001). Hidden bit b_0 always equals 1 for normalised numbers. Any number (except zero) with zero hidden bit can be alternatively represented by a number with $b_0 = 1$ by changing exponent. For example, $(0.00011)_2 \cdot 2^6$ can be rewritten as $(1.1)_2 \cdot 2^2$. This process is called *normalisation*.

Since computers have limited memory, fractional part of the significant can have only a finite number of bits. This number defines *precision* of the floating point system. Any normalised floating point number with precision p can be expressed as:

$$x = \pm(1.b_1b_2...b_{p-2}b_{p-1})_2 \cdot 2^E. \tag{7}$$

Since zero cannot be normalised, a special format is used to store it. Overton (2001) presents a small example where all numbers have the form: $\pm(b_0.b_1b_2) \cdot 2^E$ and exponent E can only take values -1, 0 or 1. This floating point system has precision 3. All normalised floating point numbers of this system and zero can be plotted on real axis as in Figure 2. A gap between zero and the smallest normalised number is relatively big. Because of this, this gap is evenly filled with *subnormal numbers* that all have zero hidden bit.

[1]More formal definition is: "an algorithm is stable if the result it produces is relatively insensitive to perturbations resulting from approximations made during the computation" (Heath, 1996)

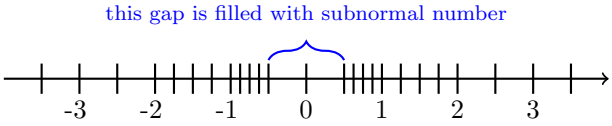


Figure 2: Example of a small floating point system

Floating point systems allow to approximate infinite set of real numbers by a finite set of floating point numbers. This means that in all computations based on floating point systems, real numbers are replaced with closest floating point numbers. This leads to round-off errors that can accumulate during computation process.

Let us come back to the question of ill-conditioned steps that might be introduced by an algorithm. Apart from round-off errors, so-called cancellation might lead to loss of precision. *Cancellation* occurs when one floating point number is subtracted from another number that is nearly equal to it (Overton, 2001). In this case all or almost all bits cancel each other. As a result, the number of significant digits is much lower compared to the numbers that were subtracted. Overton (2001) gives a good example of cancellation. Let us consider two numbers $x = 3.141592653589793$ and $y = 3.141592653585682$. The difference of these numbers is $z = x - y = 4.111 \cdot 10^{-12}$. If we now calculate this difference in C program using type *double* to store x, y and z, the result of computation will be $4.110933815582030 \cdot 10^{-12}$. This number agrees to the exact result only to four digits. All other digits are meaningless, since they do not reflect the original data x and y and should be ignored (Overton, 2001). The effect of cancellation can be disastrous for a computer program. If the error that occurred because of cancellation starts dominating true values, the result of the computation might be completely incorrect. Many examples of such behaviour of computer programs can be found in Heath (1996); Overton (2001); Stewart (1996). Cancellation can also be explained by the conditioning of the problem. In fact, computing $g(x + h) - g(x)$ is ill-conditioned for small values of h (Overton, 2001).

One of the ways to deal with cancellation is to find an alternative mathematical formulation that does not introduce ill-conditioned steps. However, it is not always possible. Another alternatives include using higher precision for ill-conditioned steps and choosing parameters of the problem in a more sensible way.

3 Traffic Assignment Problem

This Section introduces a mathematical formulation of the TA problem and some of the algorithms to solve it.

3.1 Problem Formulation

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

As was mentioned before, 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 $c_a(f_a)$ denote a link cost function of link a that depends on link flow f_a. Link flow represents a number of vehicles per time unit on each link.

Let $\mathbf{F} = (F_1, ..., F_{|K|})$ 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 expression

$$f_a = \sum_{p \in Z} \sum_{k \in K_p} \delta_a^k F_k, \quad \forall a \in A, \tag{8}$$

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. Travel time on each path is the sum of travel times of links belonging to this path, i.e. $C_k(\mathbf{F}) = \sum_{a \in A} \delta_a^k c_a(f_a)$.

The following optimisation problem (9) results in the link flows satisfying the user equilibrium condition (Sheffi, 1985).

$$\begin{aligned}
\min \quad & \sum_{a \in A} \int_0^{f_a} c_a(x)dx && && \text{(9a)}\\
\text{s.t.} \quad & \sum_{k \in K_p} F_k = D_p, && \forall p \in Z, && \text{(9b)}\\
& F_k \geq 0, && \forall k \in K_p, \forall p \in Z, && \text{(9c)}\\
& f_a = \sum_{p \in Z} \sum_{k \in K_p} \delta_a^k F_k, && \forall a \in A. && \text{(9d)}
\end{aligned}$$

Equations (9b) are flow conservation constraints, equations (9c) are non-negativity constraints and equations (9d) relate link and path flows.

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 (Florian and Hearn, 1995). In the following, we assume that these requirements are satisfied.

The formulation (9) is sometimes referred to as link-route or path flow formulation (Patriksson, 1994; Bertsekas, 1998). The path-based algorithms that are discussed in Section 3.2 use this mathematical programme. Other optimisation formulations of the TA problem based on a different set of decision variables can be found in Patriksson (1994) and Bertsekas (1998).

3.2 Path-based Algorithms

Path-based methods exploit O-D pair separability and path flow formulation (9) 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. For this purpose we use sets $K_p^+, \forall p \in Z$, that store paths between each O-D pair p that are currently in use, i.e. they carry positive flow. A framework describing this group of algorithms is presented in Figure 3.

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 (Patriksson, 1994). For a given O-D pair p the shortest path is calculated and added 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^+.

Initialisation is performed by all-or-nothing assignment. First, all link flows are set to zero and the corresponding travel times of each link are updated. Then, the shortest paths between each O-D pair are calculated and added to corresponding path sets $K_p^+, \forall p \in Z$. 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 (9), 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 (1994),

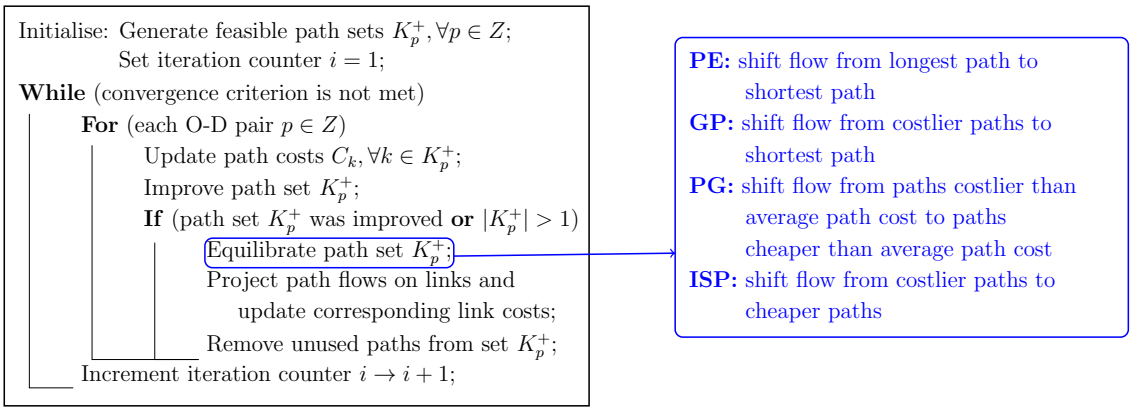


Figure 3: Framework for path-based algorithms

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 of the path costs of set K_p^+ are equalised.

In the following we discuss different path-based algorithms. 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 $C_k(\mathbf{F})$.

Path-based algorithms differ from each other in the way of how the path set is equilibrated. Usually, this involves two main algorithmic steps that depend on a particular algorithm. The first step is to calculate the direction of descent. In the following, we discuss in detail this step because this is of a particular interest for our study of numerical stability of the methods. The second step is to calculate step size λ which measures how far away the current solution must be moved in the direction of descent. In order to find an appropriate step size value we use quadratic approximation (Gentile, 2014; Cheng et al., 2009). Once the direction of descent and step size are known, the current solution $\mathbf{F}$ is updated as follows:

$$F_k = F_k + \lambda d_k, \quad \forall k \in K_p, \forall p \in Z. \tag{10}$$

However, some of the algorithms apply a different strategy of updating the current solution that does not involve step size.

The first path-based algorithm called path equilibration (PE) was proposed by Dafermos and Sparrow (1969). This 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 (Florian and Hearn, 1995):

$$\begin{aligned} d_l &= \frac{C_s - C_l}{\sum_{a \in A_{s,t}} \frac{\mathrm{d}c_a}{\mathrm{d}f_a}}, \\ d_s &= -d_l, \\ d_j &= 0, \quad \forall j \in K_p^+, j \neq s, j \neq l, \end{aligned} \tag{11}$$

where $A_{s,t}$ is the set of links belonging to path s and to path l but not to both of them. Once $\mathbf{d}$ is calculated, the solution is updated as follows:

$$\begin{aligned} F_l &= F_l - \min\{F_l, -d_l\}, \\ F_s &= F_s + \min\{F_l, d_s\}. \end{aligned} \tag{12}$$

Jayakrishnan et al. (1994) proposed a different approach called the gradient projection (GP) method and further studied it in Chen and Jayakrishnan (1998). It is similar to PE, but O-D flow

is moved from several non-shortest paths to the shortest path. In particular, the GP algorithm moves flow to the current shortest path $s \in K_p^+$ from all other paths of set K_p^+. Firstly, a direction of descent is calculated:

$$d_k = \frac{C_s - C_k}{\sum_{a \in A_{s,k}} \frac{\mathrm{d}c_a}{\mathrm{d}f_a}}, \quad \forall k \in K_p^+, k \neq s. \tag{13}$$

Secondly, a new solution is projected onto the feasible set:

$$\begin{aligned} F_k &= F_k - \min\{-\alpha d_k, F_k\}, \quad \forall k \in K_p^+, k \neq s, \\ F_s &= D_p - \sum_{k \in K_p^+, k \neq s} F_k, \end{aligned} \tag{14}$$

where α is a predefined constant that must be small enough in order to guarantee convergence of the algorithm (Jayakrishnan et al., 1994). Jayakrishnan et al. (1994) recommend setting α to 1. Chen et al. (2012) and Chen et al. (2013) also suggest a self-adaptive step size strategy for GP that we did not implement in our study.

Another path-based algorithm was proposed in Florian et al. (2009). It is called projected gradient (PG). 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:

$$d_k = \bar{C}_p - C_k, \quad \forall k \in K_p^+, \tag{15}$$

where $\bar{C}_p = \frac{\sum_{k \in K_p^+} C_k}{|K_p^+|}$ is the average cost of the paths belonging to K_p^+. The update of the current solution is then performed as in equation (10).

Another similar to PG approach called the improved social pressure (ISP) algorithm was developed in Kumar and Peeta (2011). 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. 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 (2011). 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 direction of descent $\mathbf{d}$ and moving the solution along this direction. Direction $\mathbf{d}$ is:

$$\begin{aligned} d_k &= C_s - C_k, && \forall k \in \overline{P_p}, \\ d_l &= \frac{-\sum_{k \in \overline{P_p}} d_k}{s_l(F_l) \cdot \sum_{m \in \underline{P}_p} \frac{1}{s_m(F_m)}}, && \forall l \in \underline{P}_p, \end{aligned} \tag{16}$$

where $s_m(F_m) = \sum_{a \in m} \frac{\mathrm{d}c_a}{\mathrm{d}f_a}$ is the sum of first derivatives of link cost functions of the links belonging to path m. The current solution is then updated according to equation (10).

4 High Precision and Floating Point Issues

This Section is devoted to the study of convergence of the algorithms presented above. The most common measure of convergence used in traffic assignment literature is relative gap (Slavin et al., 2006; Florian et al., 2009) which is defined as follows:

$$RGAP = 1 - \frac{\sum_{p \in Z} D_p \cdot C_{min}^p}{\sum_{a \in A} f_a \cdot c_a}, \tag{17}$$

where $C_{min}^p = \min_{k \in K_p} C_k$ is the shortest path between O-D pair p. The smaller is the value of relative gap, the closer is the solution to user equilibrium. We are interested in analysing behaviour

of the algorithms when relative gap is lower than 10^{-14}. Different methods for solving the traffic assignment problem were studied at this level of precision in the literature (Bar-Gera, 2002; Nie, 2010; Bar-Gera, 2010; Inoue and Maruyama, 2012). However, according to our knowledge only Inoue and Maruyama (2012) tested one path-based algorithm at high level of precision. Other numerical studies that analyse path-based algorithms limit the relative gap by values from the interval $[10^{-7}, 10^{-4}]$, see Mitradjieva and Lindberg (2013); Zhou and Martimo (2010); Slavin et al. (2006); Florian et al. (2009); Gentile (2014); Kumar and Peeta (2011).

Let us start by analysing the convergence patterns of path-based algorithms on Sioux-Falls instance (available at `http://www.bgu.ac.il/~bargera/tntp/`). Convergence in terms of relative gap is presented in Figure 4. As can be seen from this Figure, the line of convergence of PG starts oscillating when $RGAP = 10^{-5}$ and stops before reaching the required precision. We examined in more detail what causes such a behaviour.

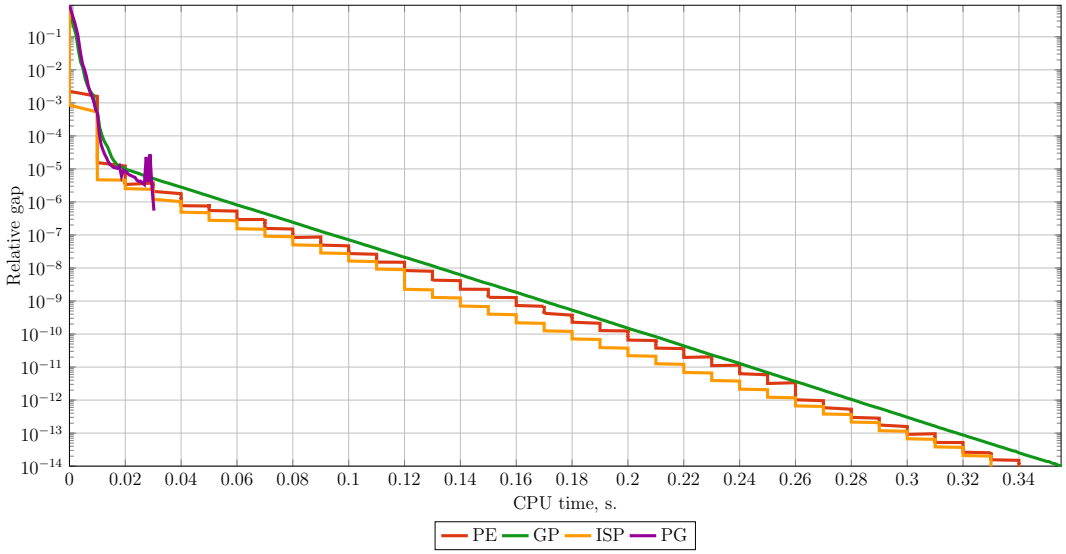


Figure 4: Convergence of path-based algorithms

Let us consider a small example presented in Figure 5 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

$$C_l - C_s = 43.708039514 \cdot 10^{-15}. \tag{18}$$

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:

$$\begin{aligned} d_l &= -2.35575448 \cdot 10^{-15}, \\ d_s &= 2.352285033 \cdot 10^{-15}. \end{aligned} \tag{19}$$

In the case of two paths, the following must hold $d_l + d_s = 0$. However, because of rounding errors we have that $d_l + d_s = -3.469446952 \cdot 10^{-18} \neq 0$. 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 $\lambda = 3.065555135 \cdot 10^{15}$. As a result, the error is multiplied by a large step size and some amount of flow is lost. In our example there are only two paths (l and s) with positive flow for this O-D pair. Hence, the flow conservation constraint after a flow shift is

$$F_l + \lambda \cdot F_l + F_s + \lambda \cdot F_s = 699.9893642 \neq 700, \tag{20}$$

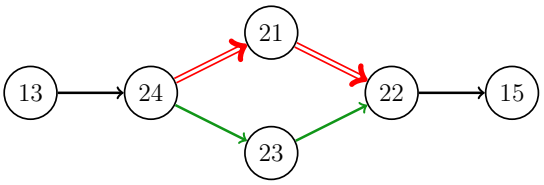


Figure 5: Example network.

where 700 is the demand of O-D pair 13-15. Hence, the amount of flow lost is 0.010635. This loss is significant and it is accumulated further during subsequent iterations. Because of this, after a few more iterations the solution becomes infeasible and the algorithm terminates.

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 because of cancellation. It seems that small path cost differences must be simply avoided. However, 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 in other path-based algorithms that we tested. The direction of descent of all path-based algorithms is based on subtracting path costs. Hence, this sub-problem becomes ill-conditioned if path costs are similar numbers.

In fact, all other path-based algorithms "correct" their directions of descent or the flow conservation constraint. Usually the directions of descent 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. In particular, we have that:

1. PE: from equation (11), we have that $d_s = -d_l$. Hence, $d_l + d_s = 0$;
2. GP: all elements of the direction of descent are calculated as in equation (13), but then the flow on the current shortest path is corrected as in equation (14) which keeps the flow conservation constraint satisfied;
3. ISP: the calculation of the elements of the direction of descent with cheaper costs (the ones that belong to $\underline{P_p}$) is based on the other elements of the direction of descent, see equation (16). This leads to the following situation: $\sum_{k \in K_p^+} d_k$ is a subnormal number less than 10^{-30} which is sufficient to eliminate the error when it is multiplied by a large step size.

Once the source of error is identified, the solution to the problem is not difficult to derive. In order to eliminate the error, we simply have to keep the direction of descent consistent, i.e. we have to make sure that $\sum_{k \in K_p} d_k = 0, \forall p \in Z$ or at least this sum is a subnormal number. To achieve this, we can "correct" the direction of descent of PG as follows:

$$\begin{aligned} d_k &= \bar{C}_p - C_k, \quad \forall k \in K_p^+, k \neq k^*, \\ d_{k^*} &= - \sum_{k \in K_p^+, k \neq k^*} d_k, \end{aligned} \tag{21}$$

where k^* can be any component of the direction of descent. For simplicity, we choose k^* to be the last element of $\mathbf{d}$.

This correction of the direction of descent of PG solves only one of many numerical issues that may occur. As mentioned before, the direction of descent itself is prone to cancellation, especially when the current solution is close to equilibrium. Another example includes link cost functions

evaluation. In many benchmark instances usual choice of link cost functions is BPR functions of the form:

$$c_a(f_a) = freeFlow \cdot \left(1 + B \cdot \left(\frac{f_a}{capacity}\right)^{power}\right), \tag{22}$$

where $freeFlow$, B, *capacity* and *power* are function parameters. Parameter *capacity* is of a particular interest for traffic assignment. It represents a capacity of a given link of a transportation network. If flow on a link exceeds *capacity*, then this link becomes congested and travel time of this link increases significantly. As pointed out in Spiess (1990), the use of high powers in BPR functions is not recommended for two reasons. During initial iterations of traffic assignment algorithms, many links can have high values of the ratio $\frac{f_a}{capacity}$. If the parameter *power* is also high, this might lead to numerical problems like overflow and loss of precision. In the situation when the ratio $\frac{f_a}{capacity}$ 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.

Therefore, floating point issues can have different origins and must be treated with care. The next Section discusses such issues in more detail based on our numerical tests.

5 Results

In order to demonstrate that fixed PG is able to reach high precision 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 1. The first five instances are of small to medium size, whereas the last three instances are large. All instances implement BPR link cost functions, see equation (22).

In this numerical tests we studied the algorithms presented in Section 3.2. All algorithms were implemented in the C++ programming language. For all of them we used extended floating point precision (C++ *long double* type). Parameter α of GP was set to 1 for all instances as recommended in Jayakrishnan et al. (1994). However, we reduced this value to 0.25 for the Winnipeg instance in order to make GP converge. On this particular instance GP was not able to reach the precision of relative gap smaller than 10^{-6} with $\alpha = 1$. The step size value influences performance of GP. Smaller values of α lead to slower convergence.

Table 1: 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	93135	medium
PRISM	14639	33937	898	470805	large
Philadelphia	13389	40003	1525	1149795	large
Chicago Regional	12982	39018	1790	2296227	large

Convergence patterns of the algorithms are presented in Figures 6-13. The required level of precision (10^{-14}) was reached for all instances except Philadelphia and Chicago Regional where the time limit of 24 hours was exceeded. The performance of fixed version of PG is comparable to other path-based methods. Convergence of the straightforward implementation of PG has a sudden decrease of relative gap before the algorithm stops. This happens because relative gap is calculated over infeasible solution leading to a smaller value of the convergence measure.

The fixed version of PG demonstrates stable numerical behaviour for the majority of instances. The only exception is the Barcelona instance where PG oscillates once relative gap reaches the

value of 10^{-6}. GP shows similar convergence pattern on this particular instance. It seems that the reason of such a behaviour is related to the Barcelona instance since it uses high powers in BPR functions that range in the interval $power \in [0, 16.83]$.

Overall, path-based methods demonstrate the ability to reach high level of precision in term of relative gap and show similar convergence patterns.

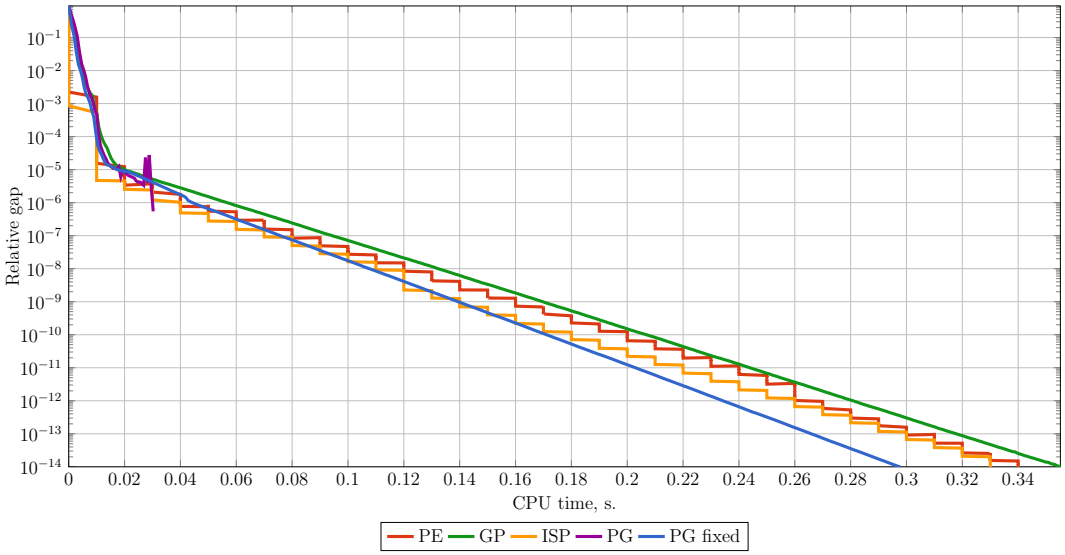


Figure 6: Convergence of the algorithms on Sioux-Falls instance

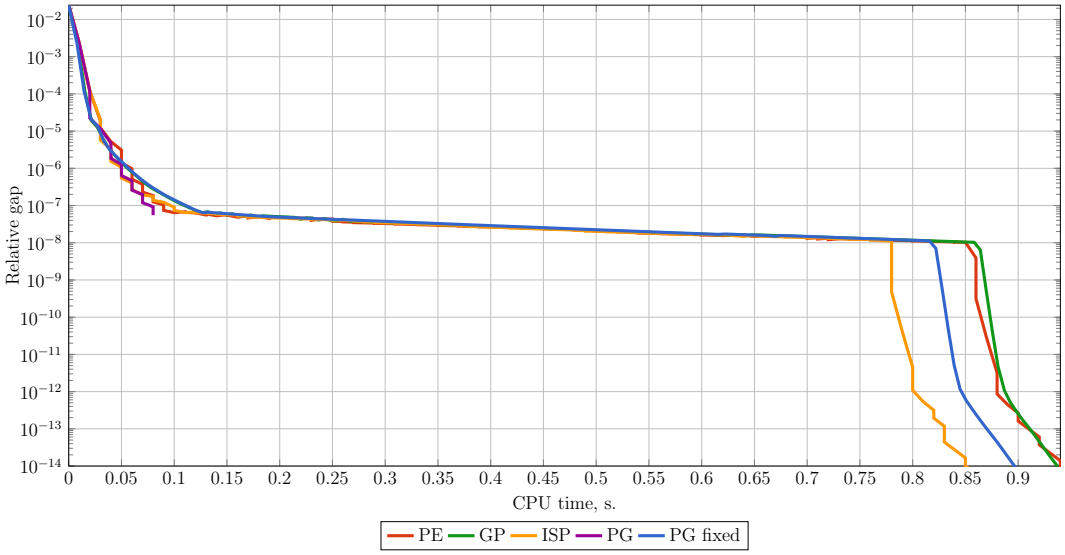


Figure 7: Convergence of the algorithms on Anaheim instance

6 Conclusion

In this study we demonstrate the importance of taking into account cancellation that might occur when highly precise solutions are required. In particular, we study path-based algorithms for solving the traffic assignment problem and their convergence behaviour. We identify that the

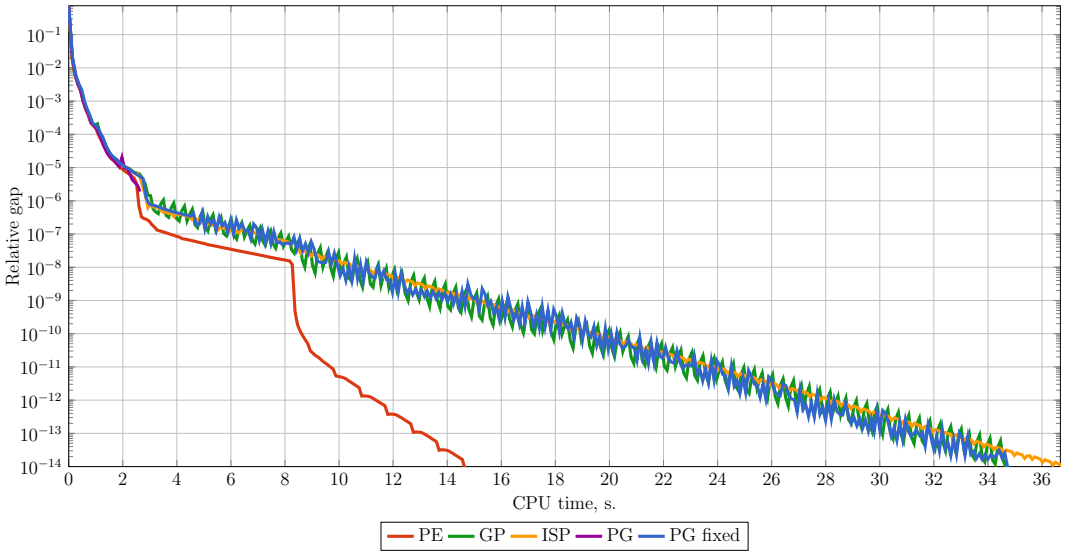


Figure 8: Convergence of the algorithms on Barcelona instance

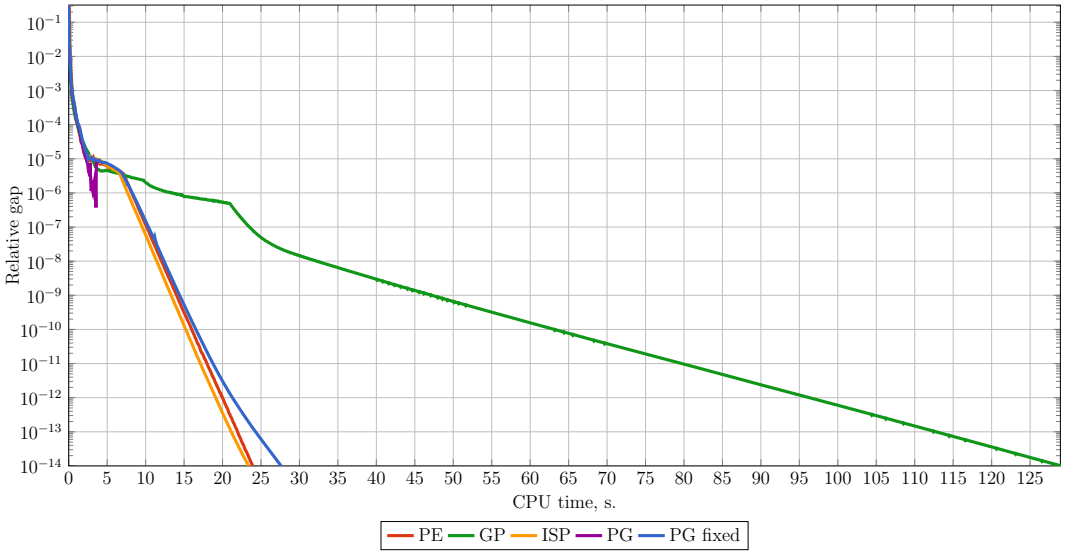


Figure 9: Convergence of the algorithms on Winnipeg instance. GP with a different value of $\alpha = 0.25$

straightforward implementation of PG introduces errors that are further magnified by the algorithm. This leads to unstable numerical behaviour. A more careful implementation takes into account the source of the errors and partially eliminates them resulting in a stable numerical behaviour of PG.

However, not all errors come from cancellation and round-off errors occurring because of finite floating point precision. Xie et al. (2013) studied a problem of convergence of another algorithm for traffic assignment called linear user cost equilibrium (LUCE). LUCE converges fast in the beginning. However, it does not improve the solution when the value of relative gap approaches 10^{-10} to 10^{-12} (the value is instance-dependant). As explained in Xie et al. (2013), the convergence issues of LUCE are related to the calculation of the direction of descent as is the case of PG. However, the source of errors is completely different. Xie et al. (2013) explain that the quadratic

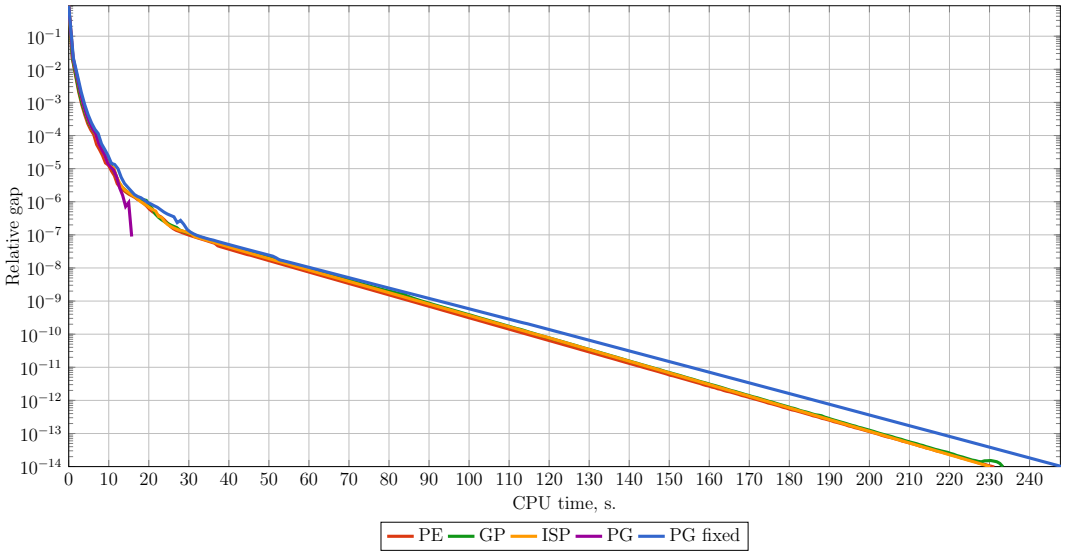


Figure 10: Convergence of the algorithms on Chicago Sketch instance

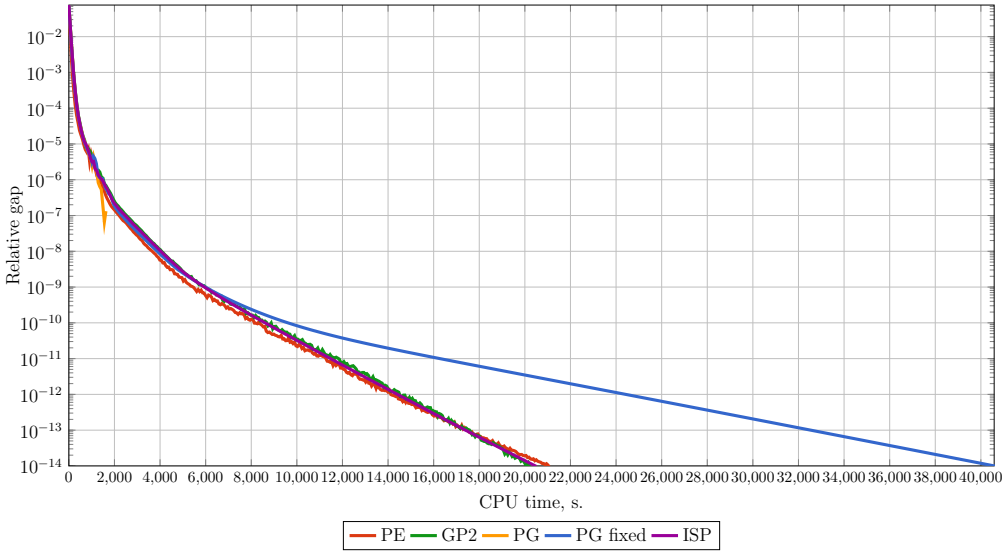


Figure 11: Convergence of the algorithms on PRISM instance

sub-problem, that is solved in LUCE to derive the direction of descent, is just an approximation (the closed form second order derivatives are unknown for the formulation of TA used in LUCE). As a result, the second order derivatives contain errors. Hence, the sub-problem is not precise enough. This limits the maximum possible precision that can be reached by LUCE. This example is a good demonstration of how the precision of solving a sub-problem can limit the master problem's precision. The TA problem itself is a sub-problem in network design and road pricing. Hence, for these problems an algorithm used to solve TA must be able to reach precision high enough for the master problem to converge.

This brings us to a conclusion that some errors originate from finite floating point arithmetic, others from model simplifications. Both of them impact the precision of solution that can be reached by an algorithm and its convergence and performance. The potential sources of errors and approximations must always be taken into account and examined, especially in the case when

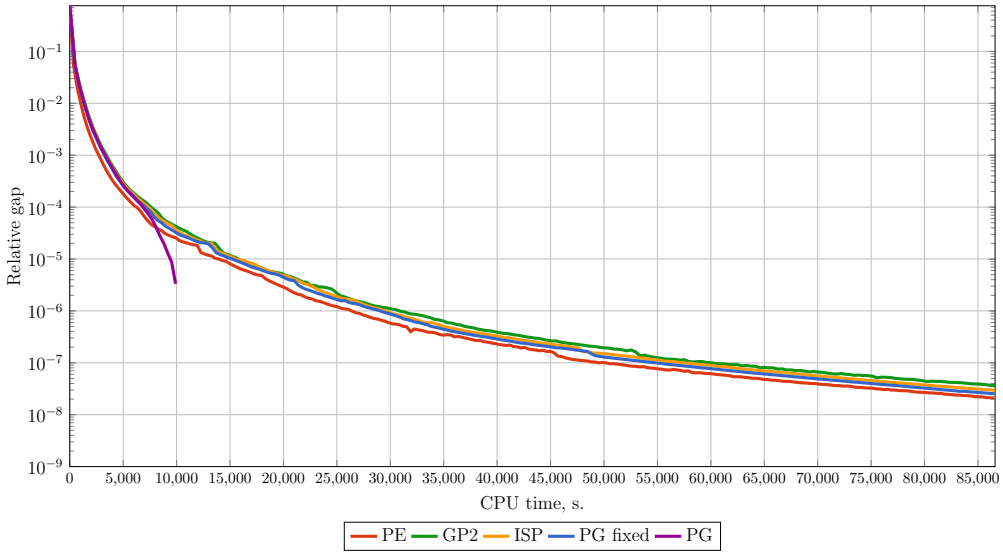


Figure 12: Convergence of the algorithms on Philadelphia instance

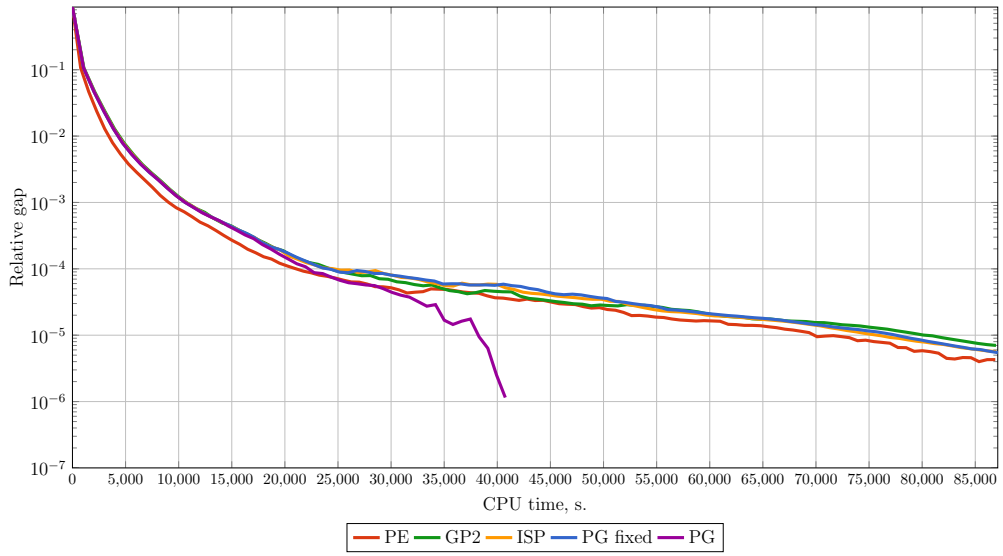


Figure 13: Convergence of the algorithms on Chicago Regional instance

the precision of solution is crucial.

References

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

H. Bar-Gera. Traffic assignment by paired alternative segments. *Transportation Research Part B: Methodological*, 44(8-9):1022–1046, 2010.

H. Bar-Gera, F. Hellman, and M. Patriksson. Computational precision of traffic equilibria sensitivities in automatic network design and road pricing. *Procedia - Social and Behavioral Sciences*, 80:41 – 60, 2013. ISSN 1877-0428. 20th International Symposium on Transportation and Traffic Theory (ISTTT 2013).

D. P. Bertsekas. *Network Optimization Continuous and Discrete Models*. Athena Scientific, 1998.

A. Chen and R. Jayakrishnan. 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*, pages 1–19, 1998.

A. Chen, Z. Zhou, and X. Xu. A self-adaptive gradient projection algorithm for the nonadditive traffic equilibrium problem. *Computers & Operations Research*, 39(2):127–138, 2012.

A. Chen, X. Xu, S. Ryu, and Z. Zhou. A self-adaptive Armijo stepsize strategy with application to traffic assignment models and algorithms. *Transportmetrica A: Transport Science*, 9(8):695–712, 2013.

L. Cheng, X. Xu, and S. Qiu. Constrained Newton methods for transport network equilibrium analysis. *Tsinghua Science & Technology*, 14(6):765–775, 2009.

S. C. Dafermos and F.T. Sparrow. The traffic assignment problem for a general network. *Journal of Research of the National Bureau of Standards. B, Mathematical Sciences.*, 73B:91–118, 1969.

M. Florian and D. Hearn. Network equilibrium models and algorithms. In C.L. Monma M.O. Ball, T.L. Magnanti and G.L. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 6, pages 485–550. Elsevier, 1995.

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

G. Gentile. Local user cost equilibrium: a bush-based algorithm for traffic assignment. *Transportmetrica A: Transport Science*, 10(1):15–54, 2014.

M. T. Heath. *Scientific Computing: An Introductory Survey*. McGraw-Hill Higher Education, 2nd edition, 1996. ISBN 0070276846.

S. Inoue and T. Maruyama. Computational experience on advanced algorithms for user equilibrium traffic assignment problem and its convergence error. *Procedia - Social and Behavioral Sciences*, 43:445 – 456, 2012. 8th International Conference on Traffic and Transportation Studies (ICTTS 2012).

R. Jayakrishnan, W. K. Tsai, J. Prashker, and S. Rajadhyaksha. A faster path-based algorithm for traffic assignment. *Transportation Research Record*, 1443:75–83, 1994.

A. Kumar and S. Peeta. An improved social pressure algorithm for the static deterministic user equilibrium traffic assignment problem. Transportation Research Board of the National Academics, Washington, D.C., 2011.

M. Mitradjieva and P. O. Lindberg. The stiff is moving – conjugate direction Frank-Wolfe methods with applications to traffic assignment. *Transportation Science*, 47(2):280–293, 2013.

Y. Nie. A class of bush-based algorithms for the traffic assignment problem. *Transportation Research Part B: Methodological*, 44(1):73–89, 2010.

J. D. Ortúzar and L.G. Willumsen. *Modelling Transport*. J. Wiley, Chichester New York, 2001.

M. L. Overton. *Numerical Computing with IEEE Floating Point Arithmetic: Including One Theorem, One Rule of Thumb, and One Hundred and One Exercises*. Society for Industrial and Applied Mathematics, 2001.

M. Patriksson. *The traffic assignment problem: models and methods*. Topics in transportation. 1994.

O. Perederieieva, M. Ehrgott, J. Y. T. Wang, and A. Raith. A computational study of traffic assignment algorithms. In *Australasian Transport Research Forum*, number 1C:2, pages 1–18, 2013.

O. Perederieieva, M. Ehrgott, A. Raith, and J. Y. T. Wang. A framework for and empirical study of algorithms for traffic assignment. Technical report, University of Auckland, 2014.

Y. Sheffi. *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*. Prentice-Hall, Englewood Cliffs, N.J., 1985.

H. Slavin, J. Brandon, and A. Rabinowicz. An empirical comparison of alternative user equilibrium traffic assignment methods. In *Proceedings of the European Transport Conference*, Strasbourg, France, 2006. Association for European Transport.

H. Slavin, P. Ricotta, J. Brandon, A. Rabinowicz, and S. Sundaram. A new traffic assignment method for small and medium communities. In *Tools of the Trade Conference*, pages 1–11, September 2010.

H. Spiess. Conical volume-delay functions. *Transportation Science*, 24(2):153–158, 1990.

G.W. Stewart. *Afternotes on Numerical Analysis*. Society for Industrial and Applied Mathematics, 1996. ISBN 9780898713626. URL `http://books.google.co.nz/books?id=w-2PWh01kWcC`.

J. Xie, Y. Nie, and X. Yang. Quadratic approximation and convergence of some bush-based algorithms for the traffic assignment problem. *Transportation Research Part B: Methodological*, 56(0):15–30, 2013. ISSN 0191-2615.

Z. Zhou and M. Martimo. Computational study of alternative methods for static traffic equilibrium assignment. In *Proceedings of the 12th World Conference on Transport Research (WCTRS), Lisbon, Portugal*, pages 1–15, 2010.