ABSTRACT

Title of Document: A COMPARISON BETWEEN AN ORIGIN BASED METHOD AND A NONLINEAR-COMPLEMENTARITY BASED METHOD FOR SOLVING THE TRAFFIC ASSIGNMENT PROBLEM

Rafael E. Olarte
M.Sc. in Civil and Environmental Engineering
2009

Directed By: Ali Haghani, Ph.D.
Chair of the Department of Civil and Environmental Engineering

This thesis compares Bar-Gera's Method and Aashtiani's Method for solving the static traffic assignment problem with fixed demand. Specifically, it compares the computational time spent by their corresponding algorithms in thirteen networks based on real cities. It also verifies whether the assumptions made by both methods and the data used allowed such a comparison. To implement Aashtiani's algorithm, a computer code was appropriately designed. To implement Bar-Gera's algorithm, a non-open source application was used. Numerical results showed mixed results but still showed the following trends: (1) Aashtiani's algorithm seems to be faster when solving complex networks, (2) Bar-Gera's algorithm is almost always faster for very high levels of accuracy while Aashtiani's algorithm is faster for lower levels of accuracy, and (3) Bar-Gera's algorithm almost always increases its speed consistently as more accuracy is demanded. Numerical results also showed that for small networks (specifically, when the number of arcs times the number of links is less than 10^7), both algorithms spent practically no more than one second, rending these networks not recommendable for carrying out future comparisons. As expected, Bar-Gera's method required less memory. This thesis also presents a unified terminology for both methods and adapted Aashtiani's formulation to this specific problem.

A COMPARISON BETWEEN AN ORIGIN BASED METHOD AND A NONLINEAR-COMPLEMENTARITY BASED METHOD FOR SOLVING THE TRAFFIC ASSIGNMENT PROBLEM

By

Rafael E. Olarte

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2009

Advisory Committee:
Professor Ali Haghani, Chair
Professor Gang-Len Chang
Professor Paul Schonfeld

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

The *traffic assignment problem* (TAP) is a classical problem in the field of transportation. Primarily, it interests decision makers when planning changes in a municipal street network. There are several versions of the TAP. This thesis focuses on the *static TAP with fixed demand* (S-TAP-F). The objective of this thesis is to compare two methods used for solving this type of problem based on numerical results obtained from thirteen city networks. Each method comprises a model –a manner in which the problem is conceived– and an algorithm –the sequence of steps used to solve that model–. In the first method, Beckman, McGuire and Winsten conceived (1956) the model and Bar-Gera (1999) proposed the algorithm. This thesis will refer to this first methodology as "Bar-Gera's method". In the second method, Aashtiani (1979) conceived both the model and the algorithm. Although Toobaie (1998) would later enhance the algorithm by improving the original data structures, this thesis will simply refer to this second methodology as "Aashtiani's method".

Three reasons explain why comparing both techniques is important. First, several authors recognize the effectiveness of Bar-Gera's method (Boyce, Mahmassani, and Nagurney 2005, p. 89) but nobody has yet compared it against Aashtiani's method. Second, Aashtiani's method uses an "asymmetric model" which, like other models of this type, is attractive for being able to solve a wider range of versions of the TAP, but have not reached popularity among practitioners because, as argued by some authors (Patriksson 1994, p. 34), they are more difficult to calibrate in practice. Therefore, if this thesis shows that Aashtiani's method generates similar results to Bar-Gera's, practitioners might acquire a revived interest in shifting towards asymmetric models. Finally, a recent comparison among almost any type of method is, in general, well received by the academic community due to (1) the continuous developments in computer technology, (2) the advancements in collection of data, and (3) the increasing size of the networks solved. This thesis takes into account these three trends by using (1) current computer technology, (2) an extensive number of networks representing real cities, and (3) networks of very different sizes.

Motivation and Background

Decisions related to transportation tend to be a priority for public officials. Examples of these decisions include constructing or widening roads, charging car users with tolls, reallocating traffic lights, adding new bus lines or adopting a new mode of transportation for a city. Some decisions are related to transportation but also reach other aspects such as the environment and the welfare of a community. As in any decision process, public officials need adequate information. Most of this information comprises specific quantifiable data. Sheffi (1985, p. 3) refers to these data as "measures of interest" and classifies them into four types as shown on *Table 1-1*.

Type	Measure of interest	Transportation Decisions
Level of service measures	Travel time and travel cost (of passing by a street segment)	Constructing or widening roads Charging car users with tolls Reallocating traffic lights Adding new bus lines Adopting a new mode of transportation Providing cheaper transportation by reducing transit fees or giving easier access to some neighborhoods. Charging vehicle users in order to reduce pollution
Operating characteristics	Revenues and profits (collected from tolls or from a public transportation fee)	
Flow by-products	Exhaust emissions Changes in land values	
Welfare measures	Accessibility measures Equity measures	

Table 1-1. Types of measures of interest, as suggested by Sheffi (1985), and examples of transportation decisions that require these measures.

Every measure of interest serves in almost every decision process. For example, exhaust emissions not only allow public officials to determine tax increases on gasoline but also the adoption of new bus lines or the restriction in the use of some streets. From the time that it takes a vehicle to travel across a street to how much of a change in value would a new road generate, these measures of interest comprise a long list. But what is striking about this apparent myriad of data is that they all share a common feature: calculating them is only possible if the *traffic flow* of the network, that is, the number of vehicles that pass through all the streets in a period of time, is known beforehand. As shown in *Figure 1-1*, knowing the traffic flow (or just the *flow*) allows the subsequent calculation of measures of interest by feeding different types of problems such as environmental models, *link performance functions*, and other types of mathematical computations.

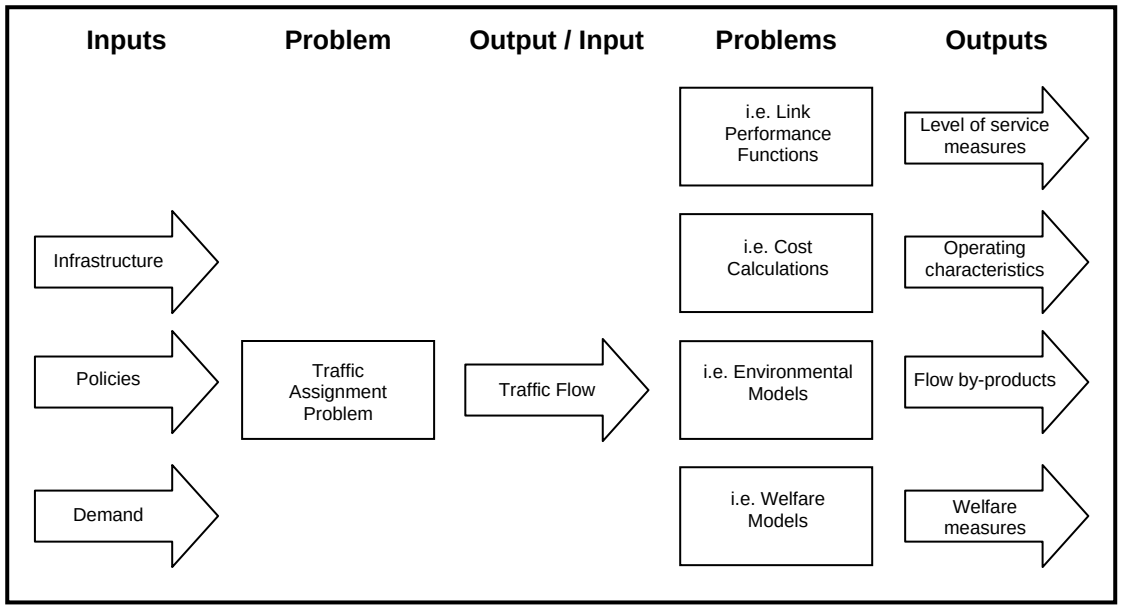


Figure 1-1. How the traffic assignment problem is a pivotal step for calculating the measures of interest.

This thesis distinguishes three types of flow. The term *total link flow* will refer to the number of vehicles that pass through a *link*, that is, a street bounded by two street intersections (or other points of interest). The term *origin-based link flow* will refer to the number of vehicles that pass through a link but share a common origin. The term *route flow* will refer to the number vehicles that start at the same origin, pass through the same links and end in the same destination. *Figure 1-2* shows the relationships between total link flows, origin-based link flows and route flows. Arrows of the same color indicate the links that belong to the same route.

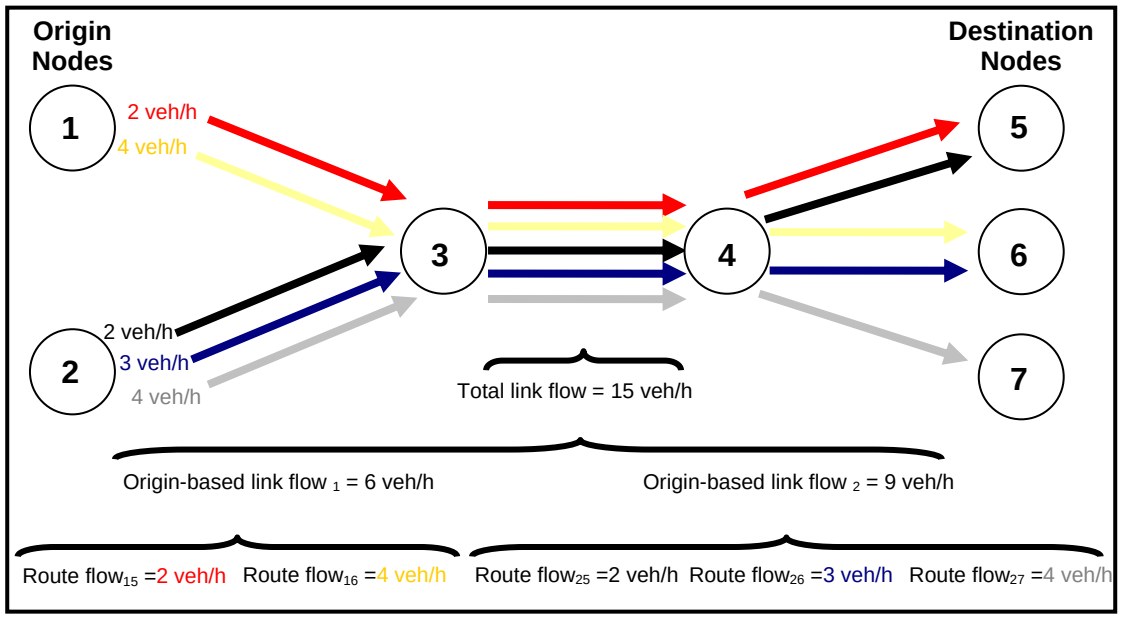


Figure 1-2. Difference between total link flow, origin-based link flow and route flow. The values shown below the brackets correspond to flows measured on link 34.

The different degrees of complexity needed in the measures of interest require the calculation of different types of flows. For example, route flows allow more detailed values because they specify the flow at every route. Total link flows on the other hand, are aggregated values that do not allow complex calculations. Route-link flows offer several advantages such as allowing (1) more accurate pollution models, (2) more consistent models for determining fees on users and (3) flexibility in scaling solutions (Bar-Gera 1999, p. 11-12). Origin-link flows fall in the middle in terms of complexity.

Finally, the reader can observe in *Figure 1-1* that the traffic flow is the result of solving the traffic assignment problem (TAP). Strictly speaking, solving the TAP requires using many inputs regarding the infrastructure (roads, capacity of the roads, intersections, transit lines and interaction with other modes such as light rail and subways), the operating and control policies (traffic lights and traffic signs, tolls) and the demand (from which origin to which destination users need to travel). Nevertheless, this thesis will consider only four inputs and also, it will focus in the S-TAP-F, a narrower version of the TAP. The S-TAP-F can be defined as the follows:

Given

(1) a street network,
(2) an *origin-destination matrix* (or the *demand*),
(3) the *link performance functions*, and
(4) assumptions on the behavior of vehicle drivers,

determine

the flow, for a specific period of time.

The above definition does not specify whether the problem requires determining total link flows, origin-based link flows or route flows. Unless specified, solving the S-TAP-F can mean obtaining any of them. The specific period of time refers to the hours of a particular day for which the problem needs to be solved and in which the flows can be assumed to be constant. Examples of typical periods of time are (1) from 6:00 am to 8:00 am on weekdays for a city with traditional rush hour patterns, (2) from 6:00 am to 6:00 pm on weekdays for a small town without traffic congestion, or (3) from 6:00 am to 6:30 am on Mondays for a city with high levels of congestion. When traffic patterns change within very small periods of time or in very unpredictable manners, solving the S-TAP-F becomes inappropriate. In such cases, other types of TAPs such as the "TAP with elastic demand" or the "dynamic TAP" become the proper problems to solve.

Figure 1-3 shows in more detail the four inputs required for solving the S-TAP-F. The *graph* (a mathematical concept brought from graph theory), refers to the set of links and nodes that represent the actual street network. The *origin-destination matrix* (or *OD matrix*) indicates the origins where users start their trips and it indicates their final destination. The OD matrix also provides the *demand* or *trips*, that is, how many vehicles per unit of time need to travel from an origin to a destination during the period of time considered. The static TAP with fixed demand is named as such due to the fixed values, not variables, that the OD matrix contains. The *link performance function* refers to how travel time (or travel cost) evolves on a link as this link gets more congested with vehicles. Finally, the fourth input corresponds to the assumptions regarding how vehicle drivers choose their routes. This thesis will follow the widely accepted assumption that (1) drivers will choose the route that minimizes their time (or the cost) to arrive to their destination and that (2) they have access to all the information needed to make that choice (this thesis does not consider so-called "system-optimum" models).

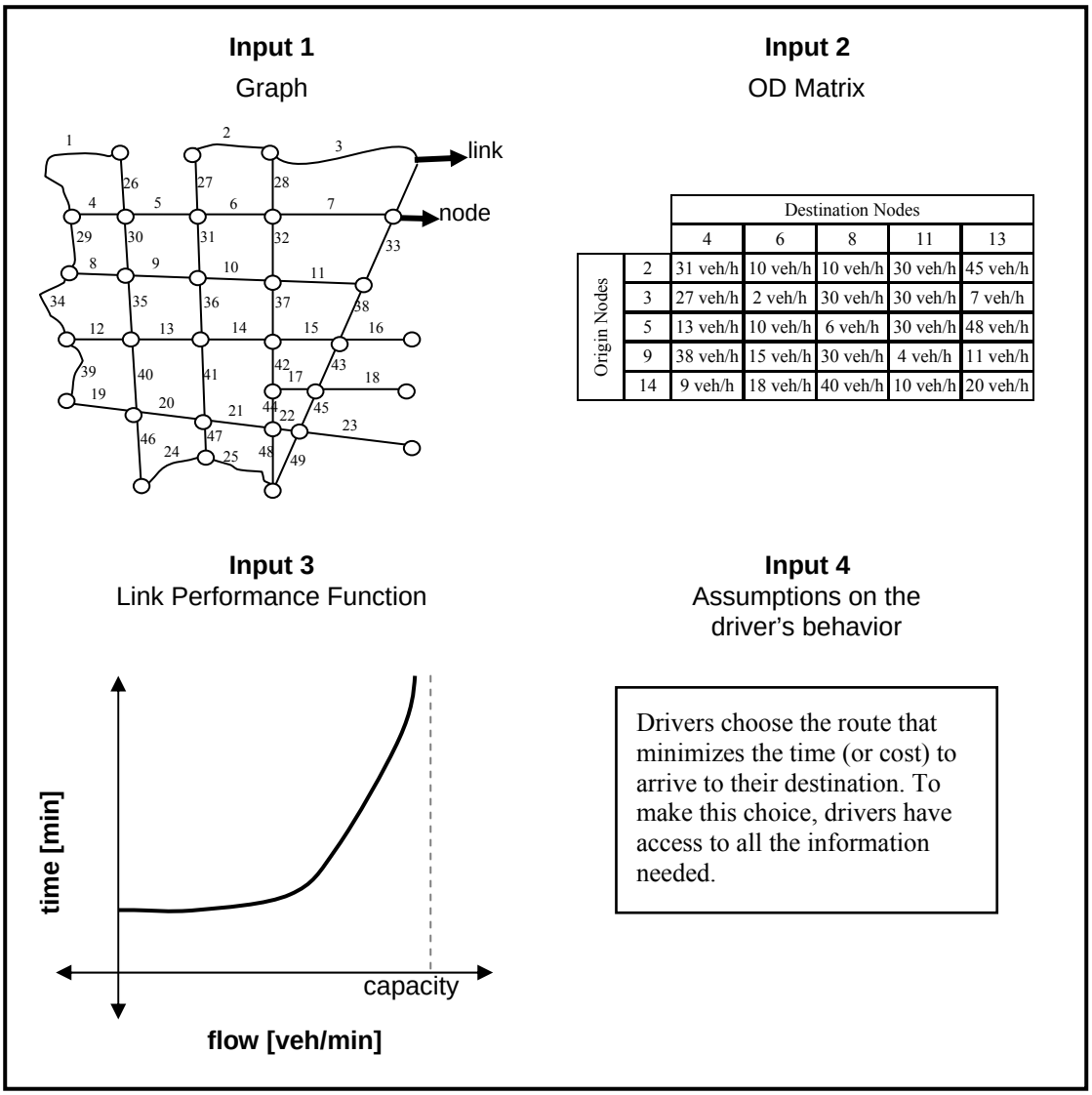


Figure 1-3. Examples of the inputs needed for solving the S-TAP-F.

Literature Review

Prior to the appearance of any model that could describe the TAP mathematically, Wardrop (1952) proposed a "principle", that is, a condition that any traffic flow should satisfy in order to be considered a legitimate solution to the TAP (he actually proposed two principles but the second one is not within the scope of this thesis). This condition, widely known as Wardrop' first principle, states that the time spent by travelers on their chosen routes should be equal or less than the time spent if they were using any other routes. This principle was later transformed by Prager (1954) into a mathematical model. As with the first models of the 1950s, it is an optimization problem composed of an objective function and a set of constraints. He derived his model from comparing traffic flows with electrical currents. Nevertheless, his assumptions (according to Patriksson 1994, p. 34) were too restrictive to be useful. The real breakthrough came soon after with the work of Beckmann, McGuire and

Winsten (1956). Their book, "Studies in Economics of Transportation", was the first to translate successfully Wardrop's first principle into a rigorous mathematical model widely known today as Beckmann's transformation (Boyce, Mahmassani, and Nagurney 2005, p. 81; Sheffi 1985, p. 61). Originally, Beckman's transformation was a model for the static TAP with elastic demand but (as mentioned by Patriksson 1994, p. 36) Dafermos was the first person to simplified it for the S-TAP-F (Dafermos 1968; Dafermos and Sparrow 1969). Nevertheless, the book does mention examples when the demand is fixed (see Boyce, Mahmassani, and Nagurney 2005).

A second generation of models, known as *asymmetric models*, appeared during the 1970s. Contrary to the first generation, asymmetric models are not optimization problems in the classical sense. They are mathematical problems of other nature that include "complementarity problems", "variational inequality problems" and "fixed point problems". They consist in finding a (and not the best) solution that complies with a set of constraints. Usually this solution is unique. But these problems do not have an objective function. As with the previous optimization models, they are based solely on Wardrop's first principle. The main advantage of these asymmetric models is that they have a wider application because they allow travel costs to be "non-separable". In other words, the travel time on each link is not a function solely of the link but of any subset of links. In consequence, these models allow more realistic scenarios. For example, links that share a common intersection have travel times that are interrelated. Another example is the relation of the travel time of traffic traveling on opposite directions within the same link. Nevertheless, these more realistic scenarios are more difficult to construct and to translate into adequate performance functions. As a result, practitioners and software companies tend not to use asymmetric models. The historical appearance of these models is as follows. Sender and Netter (1970) and Asmuth (1978) formulated the S-TAP-F as a fixed point model. Smith (1979) presented a formulation that later, Dafermos (1980) recognized as a variational inequality problem. In the same year, Aashtiani (1979) proposed his nonlinear complementarity problem.

Figure 1-4 describes the classification of the five models mentioned above as well as their major contributors. Beckman's transformation and Aashtiani's nonlinear complementarity model are the two models of concern for this thesis since they are part of Bar-Gera's and Aashtiani's methods. *Figure 1-4* also shows that the five models correspond to formulations of Wardrop's first principle and do not include other conditions that might be also important (for a further discussion see Patriksson 1994, p. 58-60).

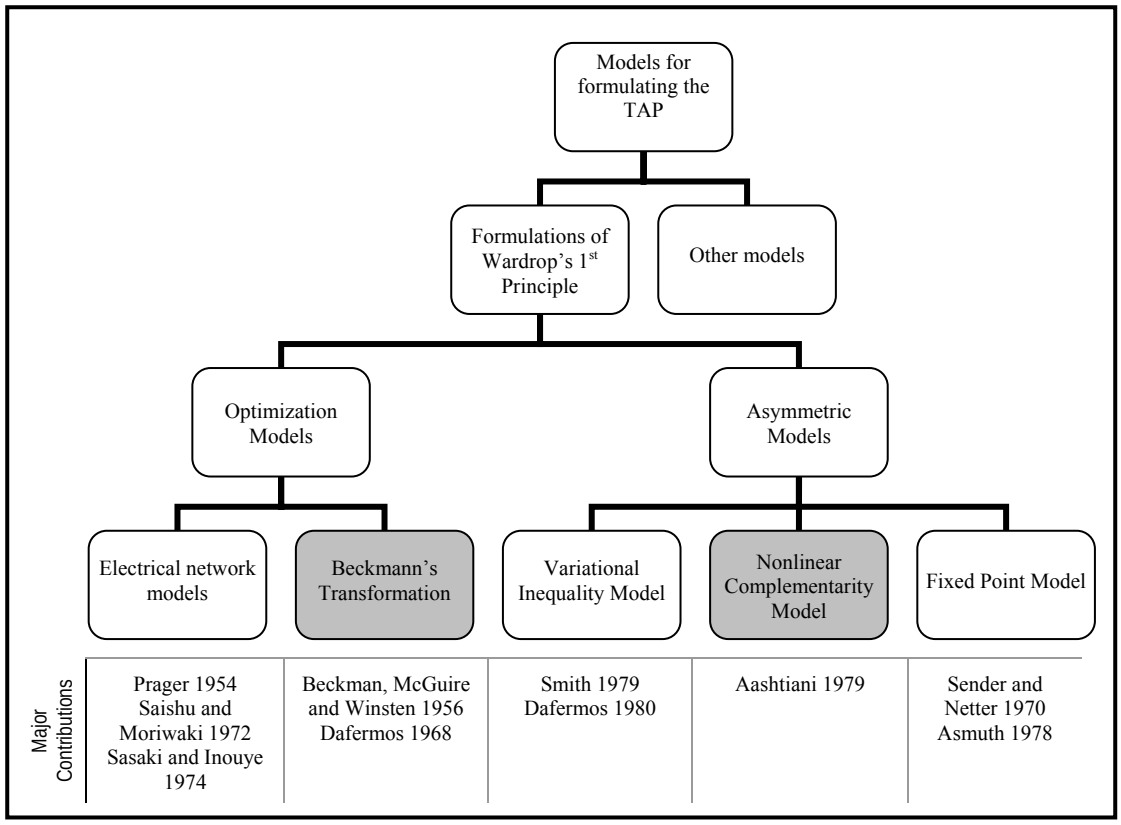


Figure 1-4. Classification of models for formulating the TAP.

Patriksson (1994, p. 105) suggests that although the number of algorithms used for solving the above models is extensive, almost all of them are based on one or on a combination of the following mathematical concepts: partial linearization, decomposition and column generation. For solving Beckman's transformation, the most widely used is the Frank-Wolfe algorithm (Frank and Wolfe 1956) which uses partial linearization. Its original purpose was to solve quadratic optimization problems but then, LeBlanc, Morlok and Pierskalla (1975) and later, Nguyen (1976) made this algorithm popular by applying it to the TAP. The Frank-Wolfe algorithm became the basis for popular software such as UROAD-UTPS (Ruiter 1974), TRAFFIC (Nguyen and James 1975), EMME/2 and EMME/3 (INRO 2008). Its main drawback is its slow convergence which prompted further research in trying to ameliorate this characteristic (for a review of these contributions, see Patriksson 1994, p. 102-104).

Another way to classify the algorithms used for the TAP is according to the type of flows that they provide (recall *Figure 1-2*). Therefore, there are *total link flow algorithms*, *origin-based link flow algorithms* and *route link algorithms*. *Total link flow algorithms* require the least amount of data storage. *Route flow algorithms* are the most expensive in terms of data storage but as explained before, a route flow solution is more detailed and can better be used for obtaining the necessary measures of interest. The Frank-Wolfe algorithm is a total link flow algorithm which, with its slow convergence, proves that low data storage does not imply a faster algorithm.

Nevertheless, if the solution needed does not require to be very accurate, the Frank-Wolfe algorithm is considered to be fast enough. *Origin-based link flow algorithms* can be seen as a trade-off between costs in data storage and great detail in its solution.

Bar-Gera's algorithm (1999) is, for example, an origin-based algorithm. Besides not requiring as much data storage as route flow algorithms, Bar-Gera's algorithm offers faster convergence than the Frank-Wolfe algorithm and has received recognition from experts in the field of transportation (Boyce, Mahmassani, and Nagurney 2005, p. 180). Bar-Gera's algorithm was designed to solve the S-TAP-F using Beckman's transformation.

On the other hand, Aashtiani's algorithm was designed to solve the nonlinear complementarity model and it is a route flow algorithm. Aashtiani's algorithm uses techniques of partial linearization and decomposition. Aashtiani (1979) showed successful results not only for solving the S-TAP-F but also with elastic demand and more complex problems where the static TAP is combined with another classical problem in transportation, known as mode choice. Although a formal proof of the convergence of his algorithm is still pending, Toobaie continued with his work by challenging his algorithm against the Frank-Wolfe method (Toobaie 1998). He improved the data storage by using link-based data structures and showed its superiority over the Frank-Wolfe algorithm in terms of convergence and accuracy. As a result, his work raises the question on whether the algorithm proposed by Aashtiani and improved by Toobaie can be faster and more accurate than Bar-Gera's origin-based link flow algorithm.

Looking at the second classification of the algorithms, another question arises. In today's world where computers are becoming less expensive, is data storage really the challenge that practitioners want to overcome? In other words, do decision makers want a faster and more accurate solution? Or are decision makers more concerned with a solution that requires less data storage? Most probably, they prefer the first alterative. Therefore, as computers become faster, route flow models (and their algorithms) will become more attractive to software developers and therefore, transportation decision makers.

Scope

This thesis compares Bar-Gera's and Aashtiani's methods based on experimental results. For this purpose, this thesis compares the time spent by Bar-Gera's algorithm with the time spent by Aashtiani´s algorithm in providing a solution to the S-TAP-F. To assure the reader that this comparison is feasible, this thesis revisits the theoretical underpinnings of not only the algorithms but also the formulations.

Although Bar-Gera's method generates origin-based link solutions and Aashtiani's, route flow solutions, this thesis transforms both solutions into total link flows and, in this manner, compares the computational times. By solving thirteen problems based on real cities, the reader will obtain a sense of which method to prefer for the S-TAP-F. Nevertheless, the reader should be aware that these problems (and their graphs) are just approximations of the real city grids.

Framing a TAP into a mathematical formulation varies greatly with the assumptions made. Also, the appropriateness of some algorithms depends on more assumptions than the ones previously considered for the formulation. Therefore, it is important to capture what this thesis does not intend to solve. For the convenience of the reader, this thesis presents here the assumptions made when referring to the S-TAP-F, the performance functions and the features of the networks used.

- The S-TAP-F is, by definition, a static TAP. Contrary to the dynamic TAP, the demand in the static TAP does not change with the period of time considered.

- The S-TAP-F is, by definition, a TAP with fixed demand. Contrary to the so-called TAP with elastic demand, the demand is not dependant on the shortest time (or cost) between each origin and destination. It is fixed, it is constant.

- The S-TAP-F considered in this thesis is a deterministic TAP. The problem assumes that users perceive precisely what the costs are of choosing any of the available routes.

- The S-TAP-F considered in this thesis requires obtaining a solution in terms of total route flows.

- The solution (the total link flows) does not need to be an integer solution. Although flow can only be measured in whole numbers of vehicles over a period of time (a period of time which is in some way arbitrarily), this simplification of the problem is a good approximation for real sized networks.

- Every performance function considered in this thesis depends solely on the total link flow that passes through it and not on other total link flows.

- Every performance function considered in this thesis is monotonically positive, continuous and strictly increasing. The graph in *Figure 3* serves as an example.

The reader might not be familiar with some of the terminology used in the last three assumptions. It is not important to understand their complete meaning for the moment. The *Section "Discussion on the Assumptions Required by both Methods"* revisits these assumptions. It also explains the special performance functions used for some types of links, the so-called *connectors*.

Organization

This thesis is organized as follows. *Chapter 2* explains briefly the two methods. It starts by defining the concepts that both methods use and by providing a unified notation. It then explains each of the methods allowing a better interpretation of the numerical results. This chapter is also important because it adapts Aashtiani's formulation to the S-TAP-F, because it discusses the importance of the assumptions made regarding the formulations and algorithms, and because it discusses the impact of the data structures used. *Chapter 3* focuses on the numerical comparison between both methods. First, it gives information regarding the software and the data sources.

Second, it explains very briefly the metrics used for the comparison. Third, it presents the results and the corresponding analysis. Finally, *Chapter 4* concludes on the performance of both methods by linking the numerical results to the theoretical aspects. It also presents lessons learned and provides suggestions for further research.

A brief description of the two methods as presented on *Chapter 2* would not allow the reader to really grasp how they function. Also, the reader might not obtain from this chapter all the theoretical elements necessary for contrasting the assumptions made by both methods. Therefore, this thesis contains an *Appendix* which presents a much more detailed explanation of the two methods.

CHAPTER 2: DESCRIPTION OF THE TWO METHODS

This chapter summarizes how Bar-Gera's method and Aashtiani's method operate. It starts by establishing a common notation. It then explains the two methods briefly. It finalizes by discussing the assumptions made by both methods and the importance of the data structures used for their implementation. The reader can refer to the *Appendix* in order to understand the mechanisms of the methods in more detail (the *Appendix* is completely self contained and therefore, does not require reading any other chapter or appendix from this thesis).

Notation

The notation used by Bar-Gera (1999) is not flexible for describing also Aashtiani's method. Vice versa, Aashtiani's notation (1979) is not rich enough for capturing the concepts that Bar-Gera's method uses. This thesis adopted a notation that facilitates the comparison between both methods and that facilitates the reading of summations heavily used in this topic. *Tables 2-1* to *2-5* present this notation allowing the reader to easily refer to Bar-Gera's (1999) and Aashtiani's (1979) original works. The reader can refer to the *Appendix* for a clear description of what the following terms mean. Any of the vectors mentioned on this thesis are row vectors unless they appear as transposed.

Term as assigned in this Thesis	Notation as assigned		
	in this Thesis	by Bar-Gera	by Aashtiani
node	n	i j u v	
set of nodes	$\boldsymbol{N}$	$\boldsymbol{N}$	$\boldsymbol{N}$
arc (or link)	a	a	a
set of arcs	$\boldsymbol{A}$	$\boldsymbol{A}$	$\boldsymbol{A}$
tail	a_t	a_t	
head	a_h	a_h	

Table 2-1. Notation used in this thesis, Bar-Gera's work (1999) and Aashtiani's work (1979) regarding the geometry of the networks.

Term as assigned in this Thesis	Notation as assigned		
	in this Thesis	by Bar-Gera	by Aashtiani
Origin node (or origin)	p	p	IO
set of origin nodes	$\boldsymbol{N}_o$	$\boldsymbol{N}_o$	
destination node (or destination)	q	q	
set of destination nodes	$\boldsymbol{N}_d$	$\boldsymbol{N}_d$	
set of destinations for a given origin	$\boldsymbol{N}_d(p)$	$\boldsymbol{N}_d(p)$	
OD pair	i (p,q)		i
set of OD pairs	$\boldsymbol{I}$		$\boldsymbol{I}$
Route (or path)	r_i $r_{(p,q)}$ $[p,n,n',\ldots,q]$	r $[v_1,\ldots,v_n]$	p
set of routes	$\boldsymbol{R}$	$\boldsymbol{R}$	
set of routes that connect an OD pair	$\boldsymbol{R}_i$ $\boldsymbol{R}_{(p,q)}$	$\boldsymbol{R}_{pq}$	$\boldsymbol{P}_i$
demand	d_i $d_{(p,q)}$	d_{pq}	d_i (fixed demand) D_i (variable demand)

Table 2-2. Notation used in this thesis, Bar-Gera's work (1999) and Aashtiani's work (1979) regarding information given by the OD matrix.

Term as assigned in this Thesis	Notation as assigned		
	in this Thesis	by Bar-Gera	by Aashtiani
route flow	h_{r_i} $h_{r_{(p,q)}}$ $h_{[n,n',n'',\ldots]}$	h_{rpq}	h_p
route flow sub-vector	$\mathbf{h}_i$		
route flow vector	$\mathbf{h}$		$\mathbf{h}$
origin-based link flow	$f_{p,a}$	f_{ap}	
origin-based link flow vector	$\mathbf{f}_p$	$\mathbf{f}_p$	
origin-based link flow matrix	$\mathbf{f}$	$\mathbf{f}$	
total link flow	$f_{\bullet a}$	$f_{a\bullet}$	f_a
total link flow vector	$\mathbf{f}_\bullet$	$\mathbf{f}_\bullet$	$\mathbf{f}$

Table 2-3. Notation used in this thesis, Bar-Gera's work (1999) and Aashtiani's work (1979), related to the traffic flow.

Term as assigned in this Thesis	Notation as assigned		
	in this Thesis	by Bar-Gera	by Aashtiani
link cost	t_a	t_a	t_a
route cost (or path cost)	c_{r_i}	c_s (route segment cost)	
route cost vector	$\mathbf{c}_i$		
minimum route cost	u_i	C_{pq}	u_i
minimum route cost vector	**u**		**u**
arc-route incidence value	$\delta_{a\, r_i}$ $\delta_{a\, r_{(p,q)}}$	δ_{ra}	δ_{ap}

Table 2-4. Notation used in this thesis, Bar-Gera's work (1999) and Aashtiani's work (1979) regarding costs.

Term as assigned in this Thesis	Notation as assigned	
	in this Thesis	by Bar-Gera
restricting subnetwork	$\boldsymbol{A}_p$	$\boldsymbol{A}_p$
topological order	$o(n)$	$o(j)$
maximum cost to a node	k_n	k_j
last common node	lcn_n	lcn_j
approach proportion	α_a	α_a
origin-based node flow	g_n	g_j
average approach cost	μ_a	μ_a
average cost to a node	σ_n	σ_j
approximated derivative of μ_a cost with respect to f_a	ν_a	ν_a
approximated derivative of σ_n with respect to g_n	ρ_n	ρ_j
flow shift	$z_{a\to b}$	$z_{a\to b}$ (desirable shift)
basic approach	b	b
set of non-basic approaches to a node	$\boldsymbol{NB}_n$	$\boldsymbol{NB}_j$
step size	λ	λ

Table 2-5. Notation used in this thesis and in Bar-Gera's work (1999) for particular concepts not needed in Aashtiani's method.

Formulations

Bar-Gera and Aashtiani use formulations that are very different. But in reality, they are equivalent, under certain assumptions, to the following formulation:

Find a vector **h** such that:

$$h_{r_i} \cdot \left[c_{r_i}(\mathbf{h}) - u_i(\mathbf{h}) \right] = 0 \qquad \forall r_i \in \boldsymbol{R}_i, \forall i \in \boldsymbol{I} \quad [2\text{-}1a]$$

$$c_{r_i}(\mathbf{h}) - u_i(\mathbf{h}) \geq 0 \qquad \forall r_i \in \boldsymbol{R}_i, \forall i \in \boldsymbol{I} \quad [2\text{-}1b]$$

$$\left(\sum_{\forall r_i \in \boldsymbol{R}_i} h_{r_i} \right) - d_i = 0 \qquad \forall i \in \boldsymbol{I} \quad [2\text{-}1c]$$

$$h_{r_i} \geq 0 \qquad \forall r_i \in \boldsymbol{R}_i, \forall i \in \boldsymbol{I} \quad [2\text{-}1d]$$

The expressions above are simply a mathematical interpretation of Wardrop's first principle ([2-1a] and [2-1b]), conditions of conservation of flow [2-1c] and the need for route flows to be non-negative [2-1d] (following the terminology used for Beckmann's transformation, the above formulation is simply the so-called *Kuhn-Tucker optimality conditions*).This formulation does not state any assumptions regarding the nature of the performance functions. Notice that in this formulation, there are several optimal solutions **h** but just one optimal solution $\mathbf{f}_\bullet$. In consequence, the following two formulations also do not have a unique solution **h** but do have a unique solution $\mathbf{f}_\bullet$.

Beckman's transformation

Bar-Gera uses the following Beckmann's transformation as the model for his method. Beckmann's transformation, a mathematical programming problem with linear constraints and a nonlinear objective function, is as follows.

Find a vector **h** such that

$$\text{minimizes } T[\mathbf{f}_\bullet(\mathbf{h})] = \sum_{\forall a \in \boldsymbol{A}} \int_0^{f_{\bullet a}(\mathbf{h})} t_a(x)\,\mathrm{d}x \quad [2\text{-}2a]$$

subject to

$$\sum_{\forall r_i \in \boldsymbol{R}_i} h_{r_i} - d_i = 0 \qquad \forall i \in \boldsymbol{I} \quad [2\text{-}2b]$$

$$h_{r_i} \geq 0 \qquad \forall r_i \in \boldsymbol{R}_i; \forall i \in \boldsymbol{I} \quad [2\text{-}2c]$$

T represents the objective function and it is directly defined in terms of total link flows $\mathbf{f}_\bullet$. But every total link flow $f_{a\bullet}$ is, by definition, a function of route flows h_{r_i}. This formulation is an artificial optimization problem because T does not have a physical interpretation. Nevertheless, its optimal solution **h** complies with the conditions shown in [2-1] and in this way, it becomes a solution to the S-TAP-F (for a demonstration, see Bar-Gera 1999, pp. 6-7; or Sheffi 1985, pp. 63-65). In fact, conditions shown in [2-1] are simply the so-called *Kuhn-Tucker optimality*

conditions. For conditions [2-1] to hold and for the optimal solution $\mathbf{f}_\bullet$ to be unique, the model makes four important assumptions regarding the performance functions. The second to last section of this chapter discusses these assumptions.

Aashtiani's formulation

Aashtiani reframes the basic formulation shown in [2-1] by expanding the solution vector **h** with the vector **u**. Therefore, every u_i becomes a new unknown variable and not simply a function of **h**. His formulation is as follows:

Find a vector $\mathbf{h} \mid \mathbf{u} =$

$$\left[h_{1_1}\ h_{2_1} \ldots h_{|\boldsymbol{R}_1|_1}\ h_{1_2}\ h_{2_2} \ldots h_{|\boldsymbol{R}_2|_2}\ \ldots h_{1_{|I|}}\ h_{2_{|I|}} \ldots h_{|\boldsymbol{R}_{|I|}|_{|I|}}\ u_1\ u_2 \ldots u_{|I|} \right] \text{such that}$$

$$h_{r_i} \cdot \left[c_{r_i}(\mathbf{h}) - u_i \right] = 0 \qquad \forall r_i \in \boldsymbol{R}_i, \forall i \in \boldsymbol{I} \quad [2\text{-}3a]$$

$$c_{r_i}(\mathbf{h}) - u_i \geq 0 \qquad \forall r_i \in \boldsymbol{R}_i, \forall i \in \boldsymbol{I} \quad [2\text{-}3b]$$

$$\left(\sum_{\forall r_i \in \boldsymbol{R}_i} h_{r_i} \right) - d_i = 0 \qquad \forall i \in \boldsymbol{I} \quad [2\text{-}3c]$$

$$h_{r_i} \geq 0 \qquad \forall r_i \in \boldsymbol{R}_i, \forall i \in \boldsymbol{I} \quad [2\text{-}3d]$$

$$u_i \geq 0 \qquad \forall i \in \boldsymbol{I} \quad [2\text{-}3e]$$

Aashtiani proved that as long as the performance function t_a is positive (the second to last section of this chapter discusses these assumptions in detail), the above formulation is a nonlinear complementarity problem.

Algorithms

Using the above formulations, Bar-Gera (1999) and Aashtiani (1979) proposed the following algorithms. While Bar-Gera proved that his algorithm converges, Aashtiani showed its convergence through a range of examples. *Chapter 3* will show, nevertheless, that Aashtiani's algorithm always converged.

Bar-Gera's Algorithm

The main characteristics of Bar-Gera's algorithm are the following: (1) it is an iterative algorithm, (2) it obtains a solution in terms of origin-based link flows f_{pa}, (3) it carries out a Newton-type search procedure, and (4) it does not manipulate the whole network but a restricting subnetwork $\boldsymbol{A}_p$ for each origin *p*. A restricting subnetwork contains all the nodes *n* belonging to the original set $\boldsymbol{N}$ and a subset of arcs *a*, enough so that a chosen origin p is connected to all the other nodes. The

Appendix clearly explains this concept. In other words, Bar-Gera's algorithm decomposes the problem by origins.

Figure 2-1 shows a simplified version of Bar-Gera's algorithm. Roughly, Bar-Gera's algorithm works as follows. It starts with an initial origin-based link flow $\mathbf{f}_p$ for every origin p (the sum of all these origin-based link flows is equal to the solution of the problem, that is, the vector of total link flows $\mathbf{f}_\bullet = \sum_{\forall p \in \boldsymbol{N}_q} \mathbf{f}_p$). Every initial $\mathbf{f}_p$ contains only a subset of links (with positive flow) which define a subnetwork $\boldsymbol{A}_p$. Having now an initial $\mathbf{f}_p$ and an initial $\boldsymbol{A}_p$ for every origin p, the algorithm starts a series of iterations. At every *cycle* (the most external loop), for each origin p, the algorithm finds a new (and better) feasible solution by answering two questions: (1) Which links should be removed or included? In other words, how to update $\boldsymbol{A}_p$? (2) How much flow should be assigned to the links? In other words, how to update $\mathbf{f}_p$? To answer the first question, the algorithm executes a sub-algorithm that modifies the existing restricting subnetwork $\boldsymbol{A}_p$. To answer the second question, the algorithm executes a second sub-algorithm that shifts existing origin-based link flows among the links of the subnetwork $\boldsymbol{A}_p$. In order to carry out this type of shifts, this second sub-algorithm uses a *Newton-type procedure* that due to its particular features, Bar-Gera denominates as *boundary search*. After each cycle, the algorithm evaluates expression [2-2a] with the new solution $\mathbf{f}_\bullet$ and checks if it generates a satisfactory minimum value of T. When the algorithm no longer finds a lower value of T, it terminates. Since the algorithm does not obtain route flows h_{r_i}, it cannot evaluate condition [2-2b] directly. Simply, the algorithm guarantees that its procedure does not violate that condition [2-2b].

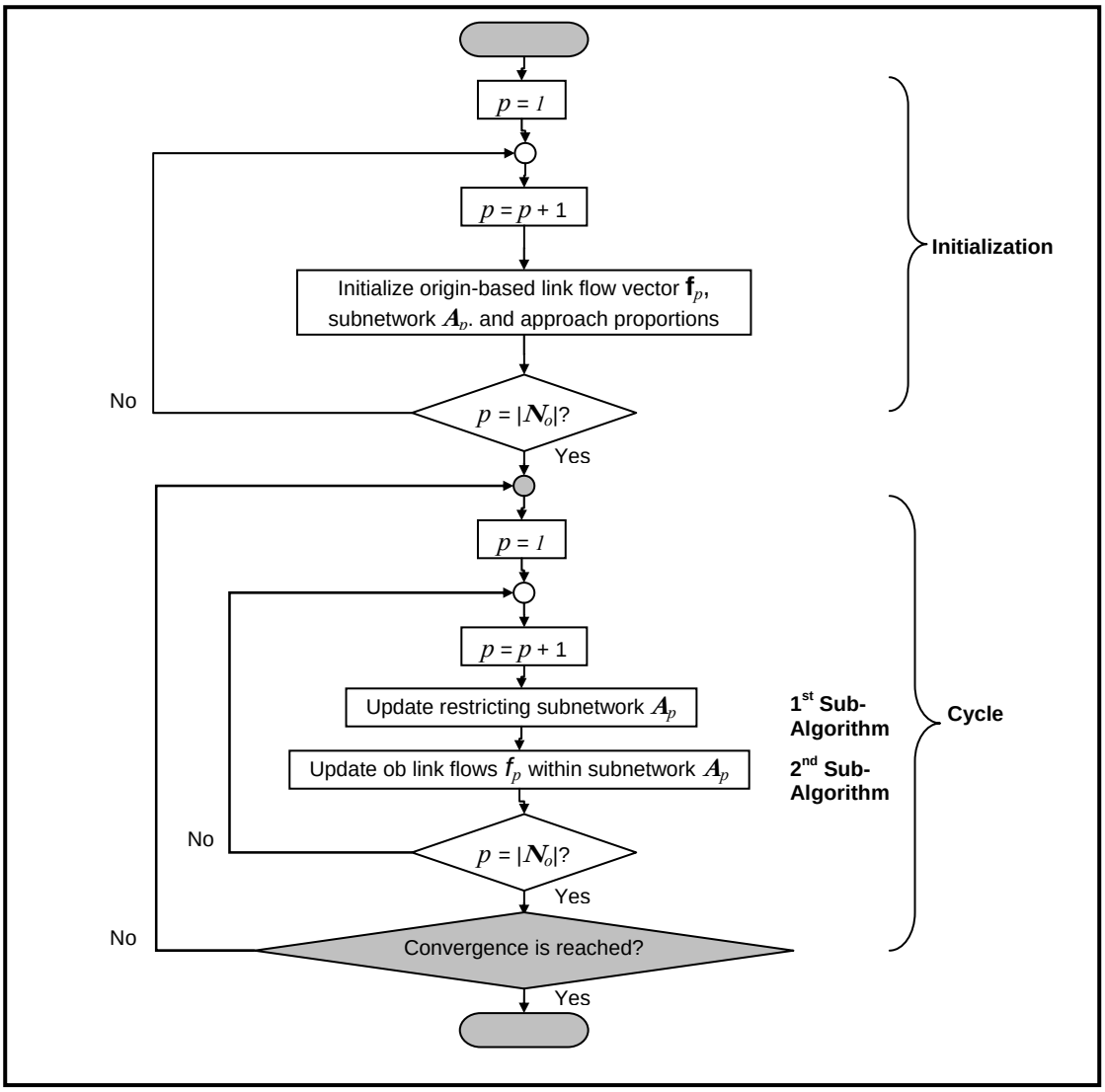


Figure 2-1. Simplified version of Bar-Gera's origin-based algorithm. "ob" stands for "origin-based".

When comparing the first sub-algorithm with the second sub-algorithm, Bar-Gera concludes that the former requires more computational time than the latter (for an explanation of this phenomenon, see *Section "Discussion on the Data Structures Recommended for the Implementation").* Therefore, he adds a modification on the algorithm so that it runs the second sub-algorithm *m* more times than the first one (for more details on this modification, see the *Appendix*). For a very large value of *m*, say 4 or 5, the computational time could increase considerably without generating better results (the term can be viewed as a parameter that increases the *local search*). Therefore, the user needs to have a guideline. Bar-Gera's software suggests a value for *m*, but the user has to run the algorithm at least one time in order to obtain that suggestion. Nevertheless, as shown in *Chapter 3*, in most of the networks tested, the value suggested was always equal to one or to two.

Aashtiani's Algorithm

Although Aashtiani (1979) presented an algorithm for different types of TAPs, this thesis adapts it to the S-TAP-F. In essence, this algorithm decomposes the network by OD pairs and then linearizes every subproblem. Interestingly, every subproblem continues being an nonlinear complementarity problem. *Figure 2-2* depicts the general framework that his algorithm follows.

Aashtiani's algorithm starts with an initial solution **h** which is calculated through the well-known "all-or-nothing" assignment (Sheffi 1985, p. 111). It then decomposes the problem into $|I|$ subproblems, where the solution to each of them is a sub-vector $\mathbf{h}_i$. It then solves each subproblem through a linearization and iterative procedure. Finally, the algorithm verifies whether the group of sub-vectors $\mathbf{h}_i$ construct a final solution **h** that complies with an additional set of conditions. If these final conditions are not met, the algorithm iterates until it reaches a satisfactory **h**. Aashtiani refers to these outer iterations as "cycles".

Although the algorithm's general framework is simple, *step 8* and *step 11* presented challenges that Aashtiani solved by the introduction of "working paths" and the execution of one-to-all shortest path algorithms by grouping the OD pairs by origins. For more details, the reader can refer to the *Appendix*.

For solving the linearized subproblem at *step 13*, Aashtiani (1979) recommended Lemke's algorithm (Lemke 1965). Aashtiani (1979) also recommended Bellman's shortest path algorithm (Bellman 1958) which was considered the best at the time (Golden 1975). This thesis used the *L-deque algorithm* (Pape 1974) instead because according to a more recent study by by Pallottino and Scuttelà (1998) , it is the fastest for transportation networks.

As with Bar-Gera's algorithm, there is a parameter that controls the number of iterations within each cycle (As mentioned in the *Appendix*, an increase in this parameter, intensifies the local search). Aashtiani (1979) and Toobaie (1998) defined it in different ways. This thesis identifies this parameter as m_A and explains it in detail in the *Appendix*.

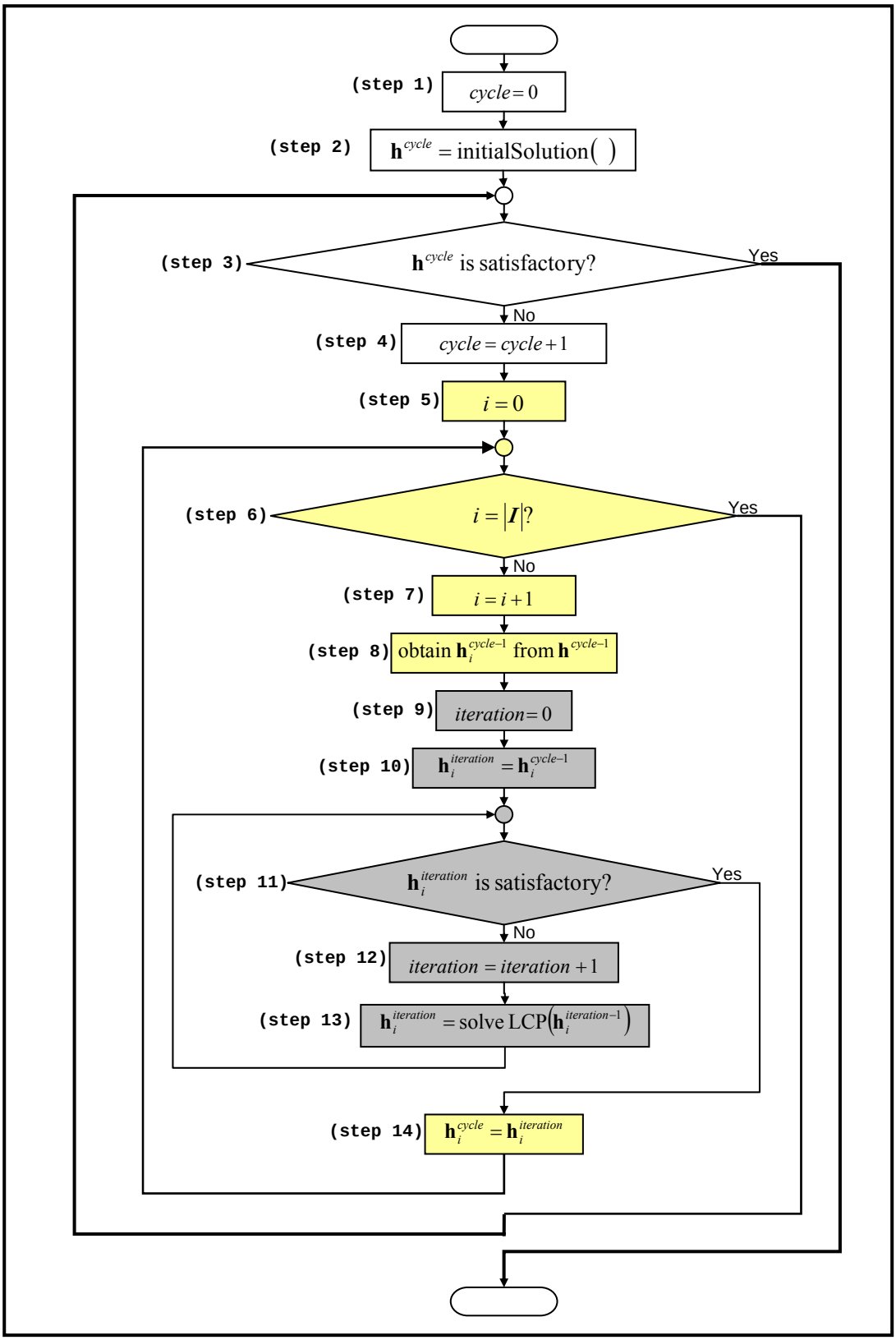


Figure 2-2. The basis of Aashtiani's algorithm: decomposition (yellow boxes) and linearization (gray boxes).

Discussion on the Assumptions Required by both Methods

Beckman's transformation has a wider application than solving the S-TAP-F. In fact, the original version of that model also solves the static TAP with variable demand. On the other hand, Aashtiani's formulation has even wider applications.

Now, restricting both formulations to the S-TAP-F, Beckman's transformation still requires stronger assumptions than Aashtiani's. Most of these restrictions relate to the nature of the performance functions.

To begin with, Beckman's transformation requires the following assumptions:

1. Every performance function t_a has to be a function of only the total link flow on link a.

$$t_a[\mathbf{f}_\bullet(\mathbf{h})] = t_a[f_{\bullet a}(\mathbf{h})] \qquad \forall a \in \boldsymbol{A} \qquad \text{[2-4a]}$$

2. Every performance function t_a has to be positive.

$$t_a[f_{\bullet a}(\mathbf{h})] > 0 \qquad \forall a \in \boldsymbol{A} \qquad \text{[2-4b]}$$

3. Every performance function t_a has to be differentiable with respect to the total link flow $f_{\bullet a}$.

4. Every performance function t_a has to be strictly increasing.

Assumptions 3 and 4 can be stated as follows:

$$\frac{\partial t_a[f_{a\bullet}(\mathbf{h})]}{\partial f_{a\bullet}(\mathbf{h})} > 0 \qquad \forall a \in \boldsymbol{A} \qquad \text{[2-4c]}$$

The first assumption is equivalent to stating that "costs are separable" or that "performance functions are independent". The reader can verify the need for the first assumption by simply observing that in Beckman's transformation, the definition of the objective function T (as indicated in [2-2a]) explicitly states that t_a is a function of f_a. The second assumption, if added to the requirement that t_a should be non-decreasing, guarantees the convexity of the objective function T and therefore the existence of the solution $\mathbf{h}$. The fourth assumption is stronger than simply requiring the performance function t_a to be non-decreasing. The fourth assumption guarantees the uniqueness of $\mathbf{f}_\bullet$ as the vector that minimizes T (but as mentioned before, the formulation does not guarantee a unique solution $\mathbf{h}$). To these assumptions, Bar-Gera's algorithm does not add any additional restrictions.

Now, Aashtiani's formulation and algorithm are applicable to the S-TAP-F even with the above four assumptions. Here are the more relaxed assumptions needed by Aashtiani's method:

1. Every performance function t_a does not have to be a function of only the total link flow on link a. A performance function t_a can be a function of the whole vector $\mathbf{f}_\bullet$.

2. Every performance function t_a still has to be positive. This restriction guarantees the equivalence between the S-TAP-F and its corresponding nonlinear complementarity problem. This restriction also guarantees the existence of at least one solution **h**.

3. Every performance function t_a does not have to be differentiable with respect to the total link flow $f_{\bullet p}$. Nevertheless, every performance function t_a does have to be continuous in order to guarantee the existence of at least one solution **h**.

4. Every performance function t_a still has to be strictly increasing in order to guarantee a unique solution $\mathbf{f}_\bullet$.

Aashtiani also mentions (1979, p. 66) that the fourth assumption is important to guarantee a unique solution **u**. This observation is important in his formulation because **u** is, contrary to Beckman's transformation, part of the unknown variables that need to be found. Aashtiani also assumes that the demand is variable, that is, it is a function of **u**. In consequence, one could speculate that perhaps, for the S-TAP-F, the assumptions regarding the performance functions could become even more relaxed. Nevertheless, the important conclusion for this thesis is that if a performance function complies with the restrictions for Bar-Gera's method, then it complies with the restrictions of Aashtiani's method.

The reader should know that nobody has already tested the convergence of Aashtiani's algorithm from a theoretical point of view. Therefore, it is still pending to know whether Aashtiani's algorithm converges without including any additional assumptions.

Now, as mentioned in the first section, connectors have a performance function that is constant and that sometimes is equal to zero. Nevertheless, if the algorithms used for obtaining the solution to [2-1] guarantee routes where connectors are only located at the beginning or at the end of each route, then there is no need for connectors to comply with the assumptions above. Their only requirement is that they do not have negative performance functions.

One last assumption that both methods require is that the performance functions t_a and the solutions **h** should not be restricted to integer values. According to Aashtiani (1979, p. 41), only-integer solutions as a requirement could make a S-TAP-F become infeasible.

Discussion on the Data Structures Recommended for the Implementation

The structures used for storing the data may affect not only the memory requirements on the machine but also the rate of convergence of an algorithm. Shortest route

algorithms, for example, can reduce their speed by containing an appropriate data structure. This section will allow the reader to have a sense of how the data structures recommended by Bar-Gera (1999) and Toobaie (1998) may affect the memory requirements and the performance of the algorithms used in this thesis.

Toobaie and Bar-Gera use extensively arrays for storing most of the data: travel link costs, OD trips, etc. Nevertheless, they both recommend special data structures (1) for the link flow solution (origin-based link solution for Bar-Gera's algorithm and route link solution for Aashtiani's algorithm) and (2) for the minimum cost routes. For the first type of information, Bar-Gera does not specify the design used for his data structure but he does not recommend storing the origin-based link flow solution in one array with one element for each arc and for each origin. It is very probable that his data structure allows at the same time defining the restricting subnetworks and the topological orders. The reader should recall that the data structures that define the restricting subnetworks experience a major change in the last step of the first sub-algorithm. In the case of Aashtiani's method, Toobaie recommends a three-level linked list: each node of the first level corresponds to a different OD pair, each node of the second level corresponds to a different route of an OD pair, and each node of the third level corresponds to a different arc of a route of an OD pair. This design stores route flows at the second level.

For the minimum cost routes, Bar-Gera uses, perhaps, a tree structure while Toobaie uses an array of link lists. Nevertheless, these special data structures, at least in Aashtiani's method, do not have the same important impact in the memory requirements and computational speed as data structures used for the link flow solutions.

Toobaie's three-level linked list reduces the memory requirements and probably the speed of the algorithm originally proposed by Aashtiani (1979). Nevertheless, the factor that affects Aashtiani's method substantially is the shortest-path algorithm that is chosen. Aashtiani's algorithm needs to execute a shortest-path algorithm every time it starts solving a different subproblem i with a different origin. Aashtiani recommended using the Bellman's shortest path algorithm (Bellman 1958) based on comparative results made by Golden (1975). But this thesis will use the *L-deque algorithm* (Pape 1974) as recommended in a more recent study by Pallottino and Scuttelà (1998) for transportation networks. They recommend it as the most appropriate for transportation networks, that is, those networks characterized by "nonnegative arc costs and structured, quasi-planar, sparse graphs".

In conclusion, the data structures play an important role in storing the flow solutions in both algorithms because they can reduce the memory requirements. In Bar-Gera's algorithm, they are also important in reducing the computational time. In Aashtiani's algorithm, it is the choice of the shortest path algorithm that mostly affects the computational time.

As re-stated in the following chapter, the algorithm implemented for this thesis is a modified version of the original code used by Toobaie (1998) and therefore, uses the same data structures explained above.

CHAPTER 3: COMPARATIVE RESULTS

Using thirteen networks based on real cities, this section presents how the two methods perform when solving the S-TAP-F. Specifically, the main objective is to determine which algorithm reaches the optimal solution within the shortest period of time. This chapter starts by describing the data sources and the software applications used for carrying out the comparison. It then briefly mentions the metrics chosen for the comparison. It then presents the performances of each method in terms of speed. It also presents the computational memory used by each method. Finally, it analyzes the results.

Software and Data Sources

For the implementation of Bar-Gera's method, this thesis used the downloadable software that appears on a website elaborated by Bar-Gera (2008). For Aashtiani's method, this thesis used the code that Toobaie implemented in his master's thesis (1998). Since Bar-Gera's software is not open source, this thesis required adapting Toobaie's code in order to (1) read the input data in the same format that Bar-Gera's software uses, (2) calculate the performance functions in the same manner that Bar-Gera's software does, and (3) include the metrics that Bar-Gera's software uses as stopping criteria. Also, Toobaie's code used Bellman's shortest path algorithm instead of the L-deque algorithm (Pape 1974) used for this thesis. The compiler used was Visual C++ 2005. The machine used was a laptop computer with 2 GB of RAM and an Intel Centrino Duo processor with a speed of 2 GHz.

#	Network	Code	Arcs	Nodes	OD pairs	Complexity (OD pairs x Arcs)
1	Chicago Regional	CHIC_R	39,018	12,979	3,134,670	122,308,554,060
2	Philadelphia	PHILAD	40,003	13,389	1,149,795	45,995,249,385
3	Berlin Center	BERL_C	28,376	12,981	49,688	1,409,946,688
4	Chicago Sketch	CHIC_S	2,950	933	142,512	420,410,400
5	Mitte, Prenzlauer Berg,	M_P_F	2,184	974	9,505	20,758,920
6	Barcelona	BARCEL	2,522	930	7,922	19,979,284
7	Winnipeg	WINNIP	2,836	1,040	4,344	12,319,584
8	Anaheim	ANAH	914	416	1,406	1,285,084
9	Mitte Center	MIT_C	871	397	1,260	1,097,460
10	PrenzlauerBerg Center	PR_C	749	352	1,406	1,053,094
11	Tiegarten Center	TIEG_c	766	359	644	493,304
12	Friedrichshain Center	FR_C	523	224	506	264,638
13	Sioux-Falls	SIOUX	76	24	552	41,952

Table 3-1. Networks employed and key futures used for measuring their size and complexity.

While the software application used for Aashtiani's algorithm starts with an initial solution obtained from executing an all-or-nothing assignment, Bar-Gera's software does not start with that type of initial solution. Bar-Gera's software simply assigns all

the traffic flow to a path that connects an OD pair but not necessarily the shortest path. Bar-Gera's software and the computer program used for Aashtiani's algorithm required one parameter, m and m_A correspondingly, that controls the number of iterations within a cycle. After executing Bar-Gera's software with any value of m, this software recommends a value to be used for future executions. After running some of the smallest networks, the software recommended a value of m equal to 2, and therefore, this was the value adopted for the rest of the networks. For Aashtiani's algorithm, a value of m_A equal to 10 was used which showed similar results to values used by Toobaie (1998) but aimed at reducing the execution of shortest path algorithms (for details on m and m_A, see the *Appendix*, page 107 and 114).

This thesis used thirteen networks which are also downloadable at Bar-Gera's website. *Table 3-1* shows their size as well as their complexity. As recommended in previous literature (Jahn et al. 2005; Holmberg and Di Yuan 2003), a common approach for measuring the complexity of a network is by multiplying the number of arcs by the number commodities (in this case, the commodities are the same OD pairs). Following this approach, *Figure 3-1* proposes one alternative of classifying the thirteen networks. The reader can observe in this classification that there is at least one network for each level of complexity.

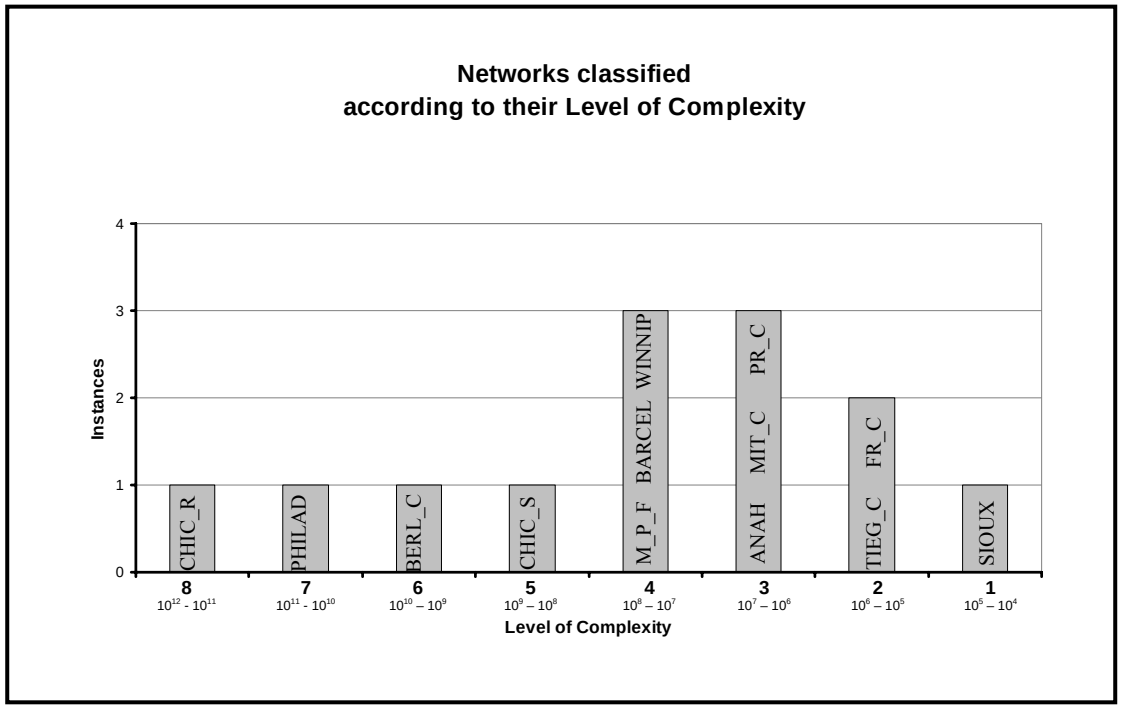


Figure 3-1. Classification of the networks used for this thesis according to their level of complexity.

All the networks use the standard form of the BPR performance function (Bureau of Public Roads 1964) for each link a. Nevertheless, some links require the addition of a value corresponding to the cost of the toll as well as a cost proportional to the length of the link. Therefore, the general performance function used for this thesis is as follows:

$$t_a(f_a) = freeFlowTime \cdot \left[1 + B \cdot \left(\frac{f_a}{capacity} \right)^{power} \right] + tollFactor \cdot tollFlag + distanceFactor \cdot length \quad [3\text{-}1]$$

where t_a is given in units of time, *freeFlowTime* is given in units of time, B is dimensionless, f_a is given in units of vehicles per units of time, *capacity* is given in units of vehicle per units of time, *power* is adimensional, *tollFactor* is given in units of time per units of money, *tollFlag* is given in units of money, *distanceFactor* is given in units of time per units of length, and *length* is given in units of length.

Although the variable *tollFlag* has units of money, its value is always either zero or one. Only the CHIC_R network contains values for the *tollFlag* different from zero. Also, only three networks used nonzero values for the variable *distanceFactor*: CHIC_R, PHILAD and CHIC_S.

Table 3-2 shows the resulting units that [3-1] generates for each of the networks and the units of the vehicle flow f_a. An extensive inquiry conducted for this thesis allows to state that for some networks, there is no knowledge of the units used for t_a and f_a. This inquiry also allows to state that some networks are the result of modifications made in the scale of the original city grids and therefore, their units could well be fractions of standard units such as "half of a minute", "0.01 hours", etc. At first glance, the reader could find this absence of units detrimental for understanding the orders of magnitude and the coherence in the results. Nevertheless, as shown in the next subsection, there are ways to circumvent this lack of knowledge in the units used.

Level of Complexity		**#**	**Network Code**	**Units for f_a**	**Units for t_a**
8	10^{11} to	1	CHIC_R	veh/hour	minutes
7	10^{10} to	2	PHILAD	veh/day	minutes
6	10^{9} to 10^{10}	3	BERL_C	*unknown*	*unknown*
5	10^{8} to 10^{9}	4	CHIC_S	veh/hour	minutes
		5	M_P_F	*unknown*	*unknown*
		6	BARCEL	*unknown*	*unknown*
4	10^{7} to 10^{8}	7	WINNIP	*unknown*	*unknown*
		8	ANAH	veh/hour	minutes
		9	MIT_C	*unknown*	*unknown*
3	10^{6} to 10^{7}	10	PR_C	*unknown*	*unknown*
		11	TIEG_C	*unknown*	*unknown*
2	10^{5} to 10^{6}	12	FR_C	*unknown*	*unknown*
1	10^{4} to 10^{5}	13	SIOUX	veh/day	minutes

Table 3-2. A caveat on the quality of the data used: for most of the networks, this thesis required estimating the units used for the link costs.

The link performance function used for the networks, as described in [3-1], complies with the assumptions needed as mentioned in the *Section "Discussion on the Assumptions Required by Both Methods"*. This section also mentions that the only links allowed to have a different kind of performance functions are the so-called connectors. The performance function of these links is equal to zero: $t_a(f_a) = 0$.

Procedure for Comparing both Methods

The specific characteristics of the networks and of the methods, and the fact that Bar-Gera's software is not open source generated some challenges for trying to reach a reliable numerical comparison between Bar-Gera's method and Aashtiani's method. In the following section, the reader will observe that a simple and perhaps natural approach of carrying out the comparison falls short in achieving the desired reliability. After analyzing other less simple approaches, this section will end by explaining the final approach (or procedure) that this study used.

The first (and simplest) approach considered consisted in comparing the computational times that both algorithms required in reaching the minimum value of Beckman's objective function *T*, as defined in equation [2-2a]. For convenience to the reader, it is rewritten below:

$$T[\mathbf{f}_\bullet(\mathbf{h})] = \sum_{\forall a \in \boldsymbol{A}} \int_0^{f_{\bullet a}(\mathbf{h})} t_a(x)\,\mathrm{d}x \qquad [2\text{-}2a]'$$

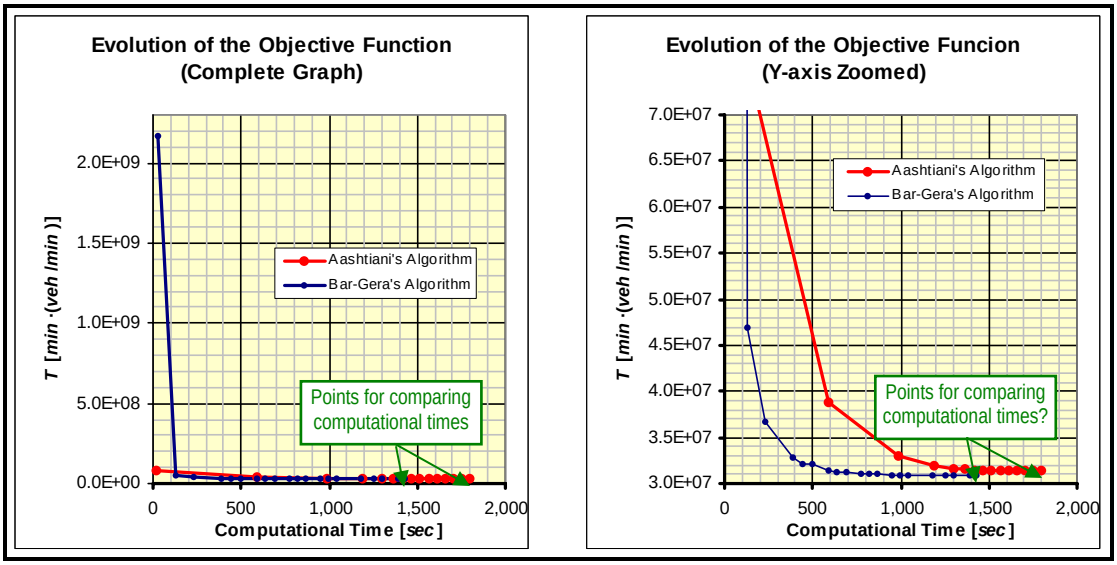


Figure 3-2. Example of a simple graph used for comparing how, for each method, the value of the objective function *T* evolves with time. The network used in this example is CHIC_R.

Figure 3-2 presents this approach using the CHIC_R network as an example. The disadvantage of this approach is that for a certain order of magnitude (the one used on the first graph of *Figure 3-2*), the algorithms seem to have reached the minimum value of *T*. But for a lesser order of magnitude (as in the second graph of *Figure 3-2*),

one could conclude that the algorithms need more computational time to reach the minimum value of *T*.

Since *T* does not have any physical interpretation, one cannot establish what a slight decrease of *T* means numerically for the solution **f.**. Without having a clear stopping criterion, what seems to be the slowest algorithm on a first comparison could become the fastest algorithm on a second comparison (see for example, *Figure 3-3*). Also, as shown on *Figure 3-3*, more computational time could mean a more satisfactory solution **f.** in terms of accuracy, but the sole value of *T* does not say much about how accurate the solution is.

A second approach considered is using a metric different from *T* that does have a physical interpretation. One convenient metric is the *average excess cost* (or AEC) which Bar-Gera's software uses as its stopping criterion. Its mathematical definition is as follows:

$$AEC = \frac{\sum_{\forall i \in \boldsymbol{I}} \sum_{\forall r_i \in \boldsymbol{R}_i} \left[\left(c_{r_i} - u_i \right) \cdot h_{r_i} \right]}{\sum_{\forall i \in \boldsymbol{I}} \sum_{\forall r_i \in \boldsymbol{R}_i} h_{r_i}} \qquad [3\text{-}2]$$

For every OD pair, the AEC calculates the difference in time between taking any route from the least costly route and it averages these differences using the route flows as weights: the lower the value of the AEC, the more similar are the traveling times between paths of a same OD pair, just as Wardrop's first principle requires. We can observe here that the AEC has a physical interpretation and it is expressed in units of time. The reader can notice also that the lower the value of the AEC, the higher the accuracy of the solution **f.**. There are other metrics such as the *maximum excess cost* (or MEC), or the one that Aashtiani's algorithm uses as its stopping criterion which he dominates simply as the *error*. As the reader can observe from the following mathematical definitions, the "error' seems to have a much better correlation with the MEC than with the AEC.

$$MEC = \max\left\{ \left(c_{r_i} - u_i \right) : r_i \in \boldsymbol{R}_i, i \in \boldsymbol{I} \right\} \qquad [3\text{-}3]$$

$$error_{Aashtiani} = \max\left\{ \left[\left(h_{r_i} - u_i \right) / h_{r_i} \right] : r_i \in \boldsymbol{R}_i, i \in \boldsymbol{I} \right\} \qquad [3\text{-}4]$$

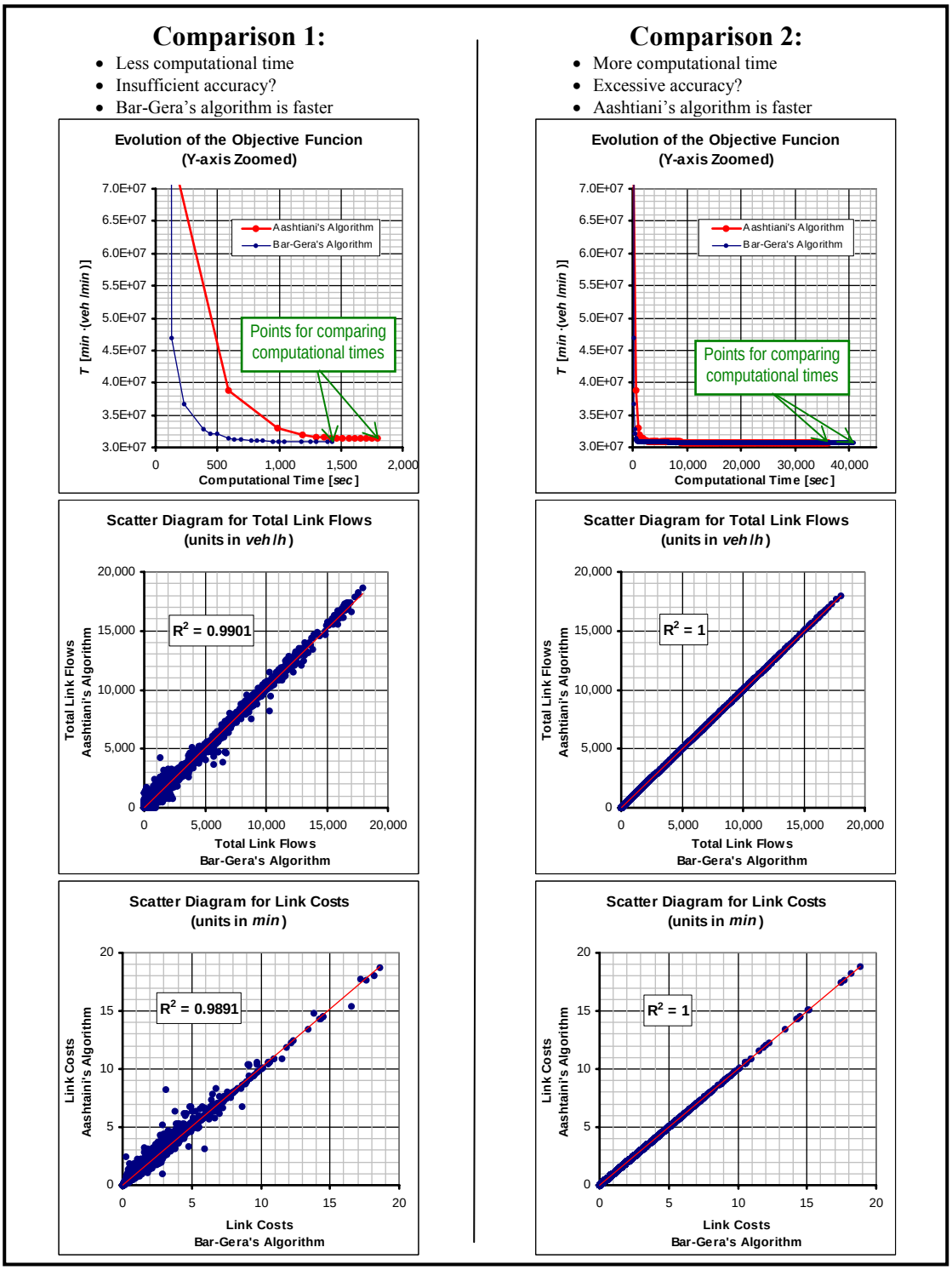


Figure 3-3. Example (PHILAD network) that shows that without a clear stopping criterion, the result of the comparison between the two algorithms can be very different. On the left side, the targeted AEC was 10^{-1} minutes. On the right side, the targeted AEC was 10^{-6} minutes.

Unfortunately, Bar-Gera's software does not calculate the "error". Therefore, the second approach considered in this thesis consisted in (1) running the two computer

programs until they reach a predefined AEC, (2) comparing the computational times, and (3) verifying whether a similar result can be drawn from looking at the MEC reached by each algorithm. In this thesis, a *targeted AEC* will refer to the AEC that the algorithms are set to reach before rendering the final solution **f.**. Since the algorithms cannot stop at the very instance they reach the targeted AEC (but after they have ended a complete cycle), their final solution will usually have a lower AEC than the targeted AEC. *Figure 3-4* shows the results for the PHILAD network when targeting an AEC of 10^{-2} minutes.

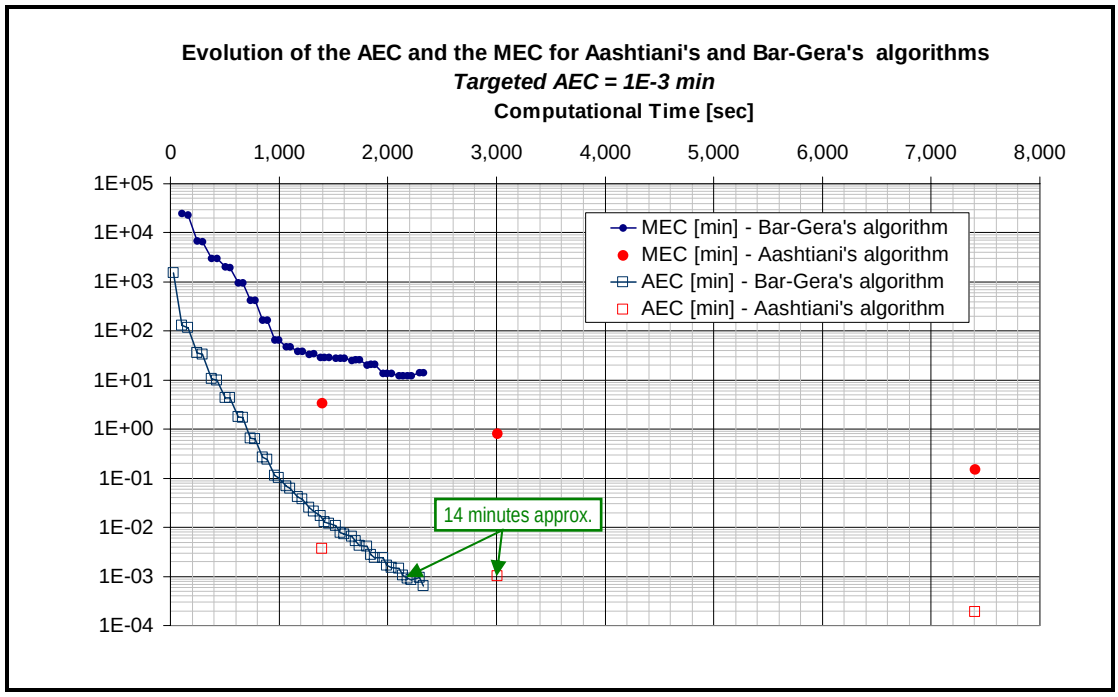


Figure 3-4. Example of how to compare the two algorithms using the second approach: at the targeted AEC, the difference in computational time is measured. In this example where the PHILAD network was used, the difference was 14 minutes. The values in the MEC are used to corroborate the numerical difference previously obtained. Here, the trend in the MEC does not contradict the superiority of Bar-Gera's algorithm.

Figure 3-4 shows the values of the AEC and the MEC for Aashtiani's algorithm only at the end of each cycle. Since Aashtiani's algorithm decomposes the problems by OD pairs and then solves each subproblem in sequential order, it cannot generate an AEC before the cycle ends. For this reason, the values corresponding to Aashtiani's algorithm appear more scattered.

The second approach presents one difficulty. It requires knowing the units of the link costs t_a. In the example of *Figure 3-4*, the units are known. As a result, the AEC would be calculated in minutes and therefore, one could state that a targeted AEC of 10^{-3} minutes is sufficiently small. As *Table 3-2* indicates, not all the units are known for this study. For this reason, this thesis considered a third approach.

The third (and final) approach uses the same metrics that the second one but it proposes observing other aspects of the results. The third approach consists in a set of six steps and therefore, the reader can view it as a simple procedure:

1. For each method, execute the algorithms targeting a very low value in the AEC.

2. Plot (like with the first and second approaches) the evolution of T.

3. For various targeted AECs, plot one scatter diagram that compares the total link flows f_a and another scatter diagram that compares the link costs t_a between the two algorithms.

4. Use the scatter diagrams to determine the maximum targeted AEC that guarantees a straight line (the reader can observe here that this verification is somewhat subjective, especially when the units are unknown. The coefficient of determination R^2 is recommended for this verification but it can be heavily influenced by outliers whose magnitudes are very large compared to the rest of the points).

5. Plot (like with the second approach) the evolution of the AEC and the MEC against the computational time.

6. Look for *trends* in the evolution of the AEC and verify those trends with the MEC. Make sure that the values of the AEC are less than or equal to the maximum targeted AEC that was determined on step 4.

Unlike in the second approach, one does not look for a "number" that measures the difference in computational time between both algorithms. Instead, one looks for "trends". This thesis used the third approach for all the networks. The results allowed to conclude that the most common trends relate to answering the following questions: (1) For a very low AEC, which algorithm spends less computational time? (2) For a very high AEC, which algorithm spends less computational time? (3) Are the answers the same to the previous two questions? (4) The differences in computational time fall within what range? (5) Does the difference in computational time for every value in the AEC increase at every subsequent iteration? (6) Are the differences in computational time significant? (7) Can one reach the same answers if looking at the MEC instead of the AEC?

A good idea for answering the sixth question is by expressing the differences in terms of percentages and not in terms of absolute seconds. One can quickly spot these percentages by using a logarithmic scale for the computational time.

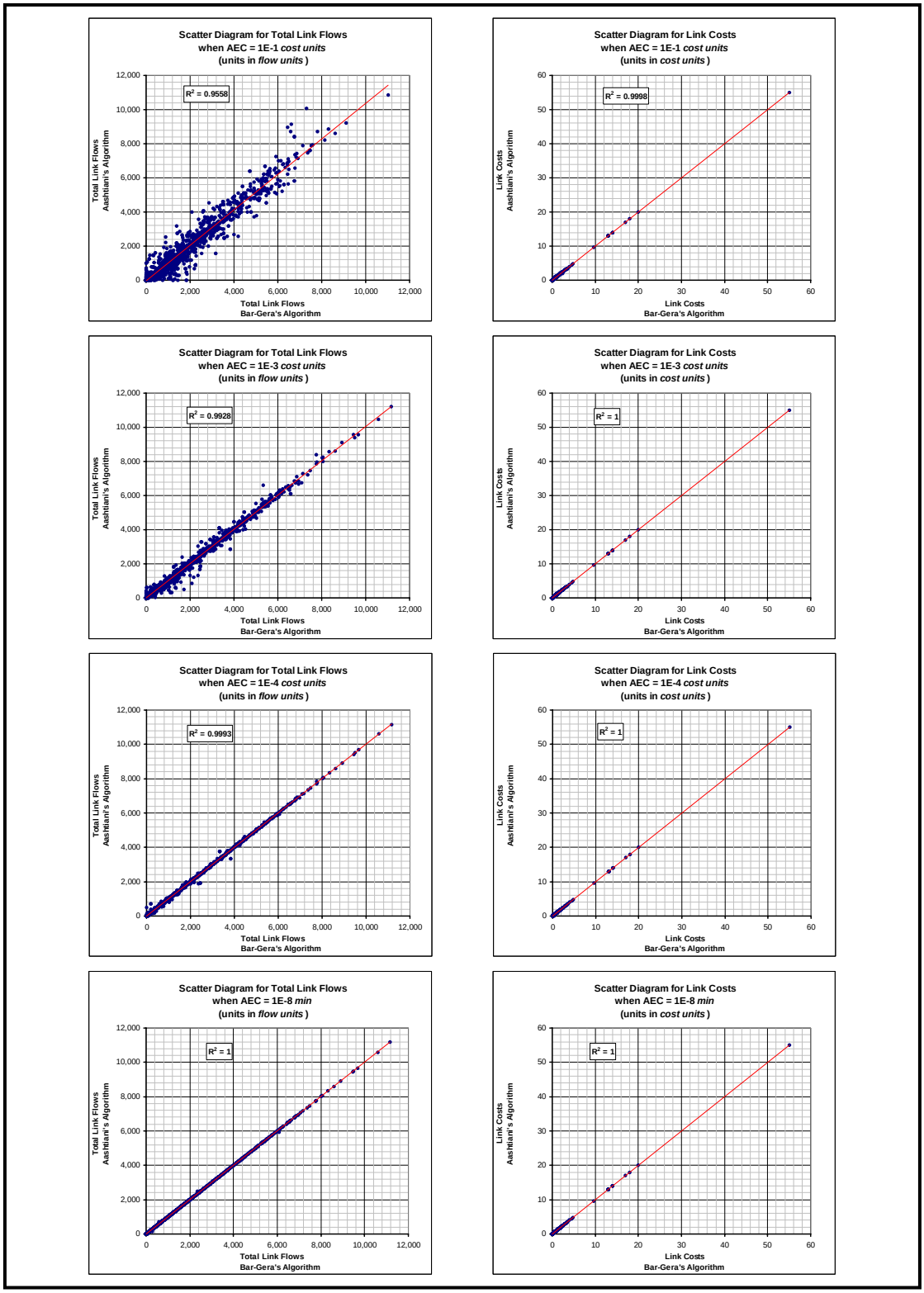


Figure 3-5. Scatter diagrams and coefficients of determination for the BARCEL network that compare the total link flows and the link costs calculated by Aashtiani's and Bar-Gera's algorithms for targeted AECs of 10^{-1}, 10^{-3}, 10^{-4} and 10^{-8} cost units.

As an example to the procedure explained above, one can look at the results obtained for the BARCEL network. *Figure 3-5* presents the scatter diagrams mentioned on the first step. There, one can observe that when the targeted AEC is 10^{-8}, the solution seems to show enough precision, even if the units of the AEC are unknown. When the AEC is 10^{-3} (or more), the solution is not so precise in terms of link flows. Nevertheless, to argue that an AEC of 10^{-3} is not satisfactory would be difficult because there is still a linear correlation and also, the straight line is still clear in the scatter diagram of the link costs. In sum, one can conclude that the scatter diagrams indicate that it is useful to look for trends starting at an AEC of 10^{-3}.

Now, continuing with the fourth step, let us plot the evolution of the AEC and the MEC as shown on *Figure 3-6*. The reader can observe here one of the advantages of using a logarithmic scale for the computational time: a distance that represents a 100% difference is constant all along the abscissa. Likewise, a distance that represents 200% difference is constant all along the abscissa. This difference expressed as a percentage is equal to:

$$\%\ \text{difference} = \frac{\text{largest comp. time} - \text{smallest comp. time}}{\text{smallest comput. time}} \cdot 100\% \qquad [3\text{-}5]$$

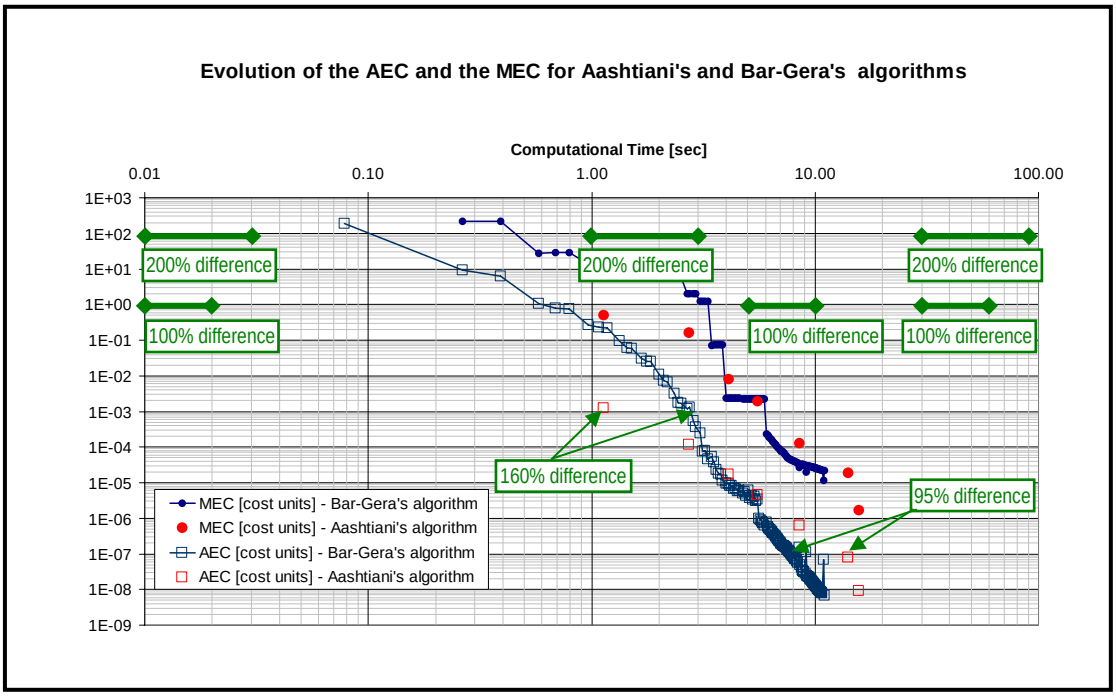


Figure 3-6. Example of how to recognize the trends in the AEC using the third approach: when Bar-Gera's algorithm is the slowest, its computational time is measured against Aashtiani's and it is expressed as a percentage (as defined on [3-5]). Likewise, the difference in which Aashtiani's algorithm is the lowest is also recorded. The values in the MEC are used to corroborate the numerical difference previously obtained. In this example where the BARCEL network was used, the MEC confirms the trend seen in the AEC. Horizontal green bars in this graph are presented to show how the logarithmic scale allows to directly recognize differences as defined in [3-5].

After looking at *Figure 3-6*, one can spot the following trends. For an AEC less than or equal to 10^{-3}, Aashtiani's algorithm starts by being the fastest. When the AEC is approximately 10^{-3}, Bar-Gera's algorithm is slower. No algorithm is faster than the other always; there is no strong superiority of one algorithm over the other. While Aashtiani's algorithm ends by being 95% slower than Bar-Gera's, Bar-Gera's algorithm starts by being 160% slower. Both algorithms seem equally fast for an AEC between 10^{-5} and 10^{-6}. Overall, we can observe that Bar-Gera's algorithm becomes faster as one allows more computational time. The differences are fairly significant since they represent values greater than 90%. The values of the MEC corroborate the values of the AEC. When Aashtiani's algorithm is slower in terms of the AEC, so it is in terms of the MEC.

In sum, one could conclude that Aashtiani's algorithm is the fastest algorithm for a not very demanding accuracy while Bar-Gera is the fastest for very demanding accuracies.

Numerical Results

For each city, we will follow the procedure explained in the previous section. Therefore, for each city, this section will present three figures (the evolution of *T*, the evolution of the AEC and the MEC, and the scatter diagrams) followed by an analysis on the trends and additional observations. The following networks are organized in descending order of complexity. For convenience, the main characteristics of the networks are mentioned again at the beginning of each description.

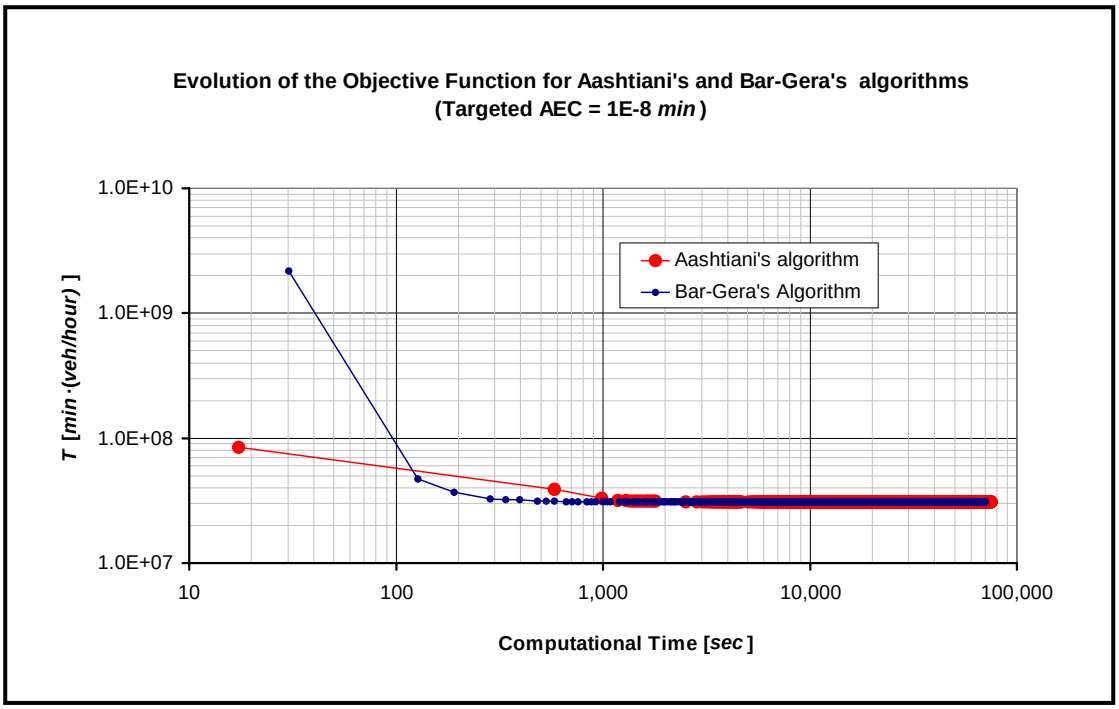


Figure 3-7a. CHIC_R network: Evolution of the objective function value for Aashtiani's and Bar-Gera's algorithms.

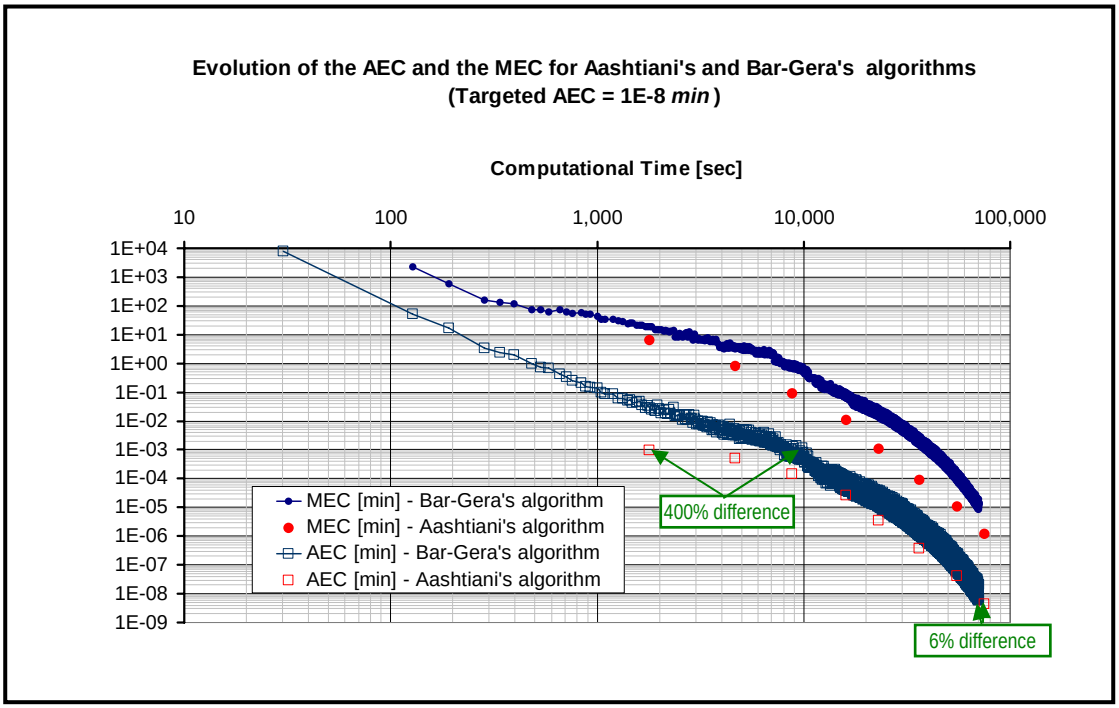


Figure 3-7b. CHIC_R network: Evolution of the average excess cost and the maximum excess cost for Aashtiani's and Bar-Gera's algorithms (values in green indicate maximum differences).

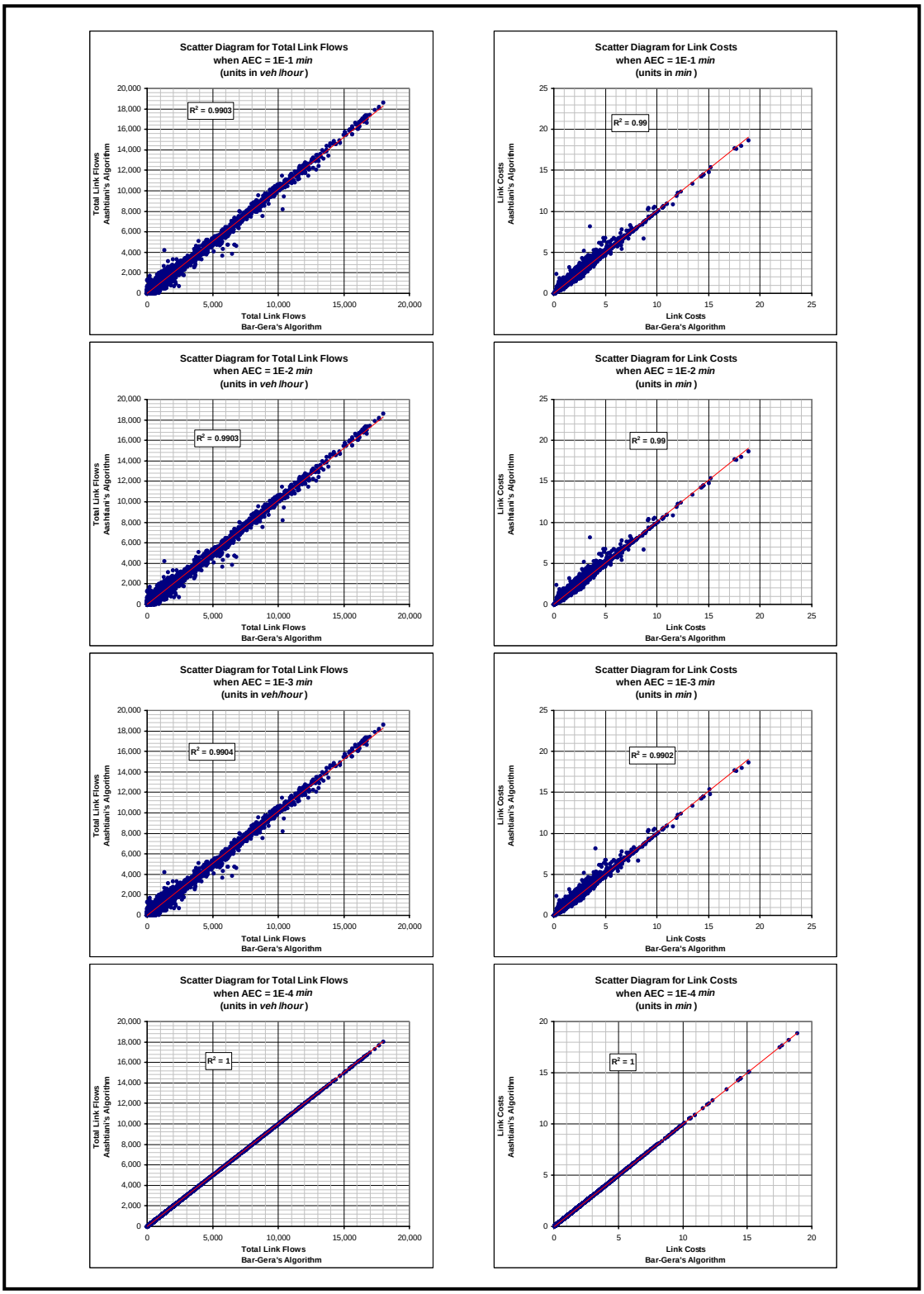


Figure 3-7c. CHIC_R network: scatter diagrams and coefficients of determination that compare the total link flows and the link costs calculated by Aashtiani's and Bar-Gera's algorithms when targeting an AEC of 10^{-1}, 10^{-2}, 10^{-3} and 10^{-4} minutes.

Description of the results shown on Figure 3-7a to Figure 3-7c

Network: Chicago Regional
Code: CHIC_R
Number of Nodes: 12,979
Number of Zones: 1,790
Number of Arcs: 39,018
Number of OD Pairs: 3,134,670
Complexity: 122,308,554,060 arcs·OD pairs
Level of Complexity: 8 (10^{11} to 10^{12})
Total flow (total demand): 1,360,428 vehicles/hour
Units for the total link flows f_a: vehicles/hour
Units for the link costs t_a: minutes
Existence of tolls: Yes
Distance factor equal to zero: No

Since the units are known, one could state that a targeted AEC of 10^{-1} minutes will always be sufficient to guarantee an acceptable solution, especially for a network as big as CHIC_R. Nonetheless, the reader can also look at the scatter diagrams (*Figure 3-7c*) to verify that the total link flows of the two algorithms describe a very linear trend ($R^2 = 0.9903$). Therefore, when looking for important trends, one can consider any values in the AEC less than or equal to 10^{-1} minutes.

The trend that *Figure 3-7b* presents is the following. For an AEC of approximately 10^{-3} minutes, Bar-Gera's algorithm is almost 400% slower than Aashtiani's. Then, as the algorithms spend more time for obtaining more accurate solutions, Bar-Gera's algorithm is still the slowest but the difference gets reduced. Only at the very last minutes, Aashtiani's algorithm gets surpassed becoming more or less 6% slower. The differences in time are very significant at the beginning: a 400% difference represents in this case two hours of additional computational time. The values of the MEC corroborate partly the trend seen on the AEC: the trend of the MEC portrays Aashtiani's algorithm as always being the fastest.

Due to the great size of the network, the computer time ranges from 30 minutes to 23 hours. One interesting feature to observe in *Figure 3-7a* is that Bar-Gera's algorithm spends more time calculating the initial solution at the same time that its objective function is poorer (that is, is greater) than the value obtained by Aashtiani's algorithm. Nevertheless, before Aashtiani's algorithm finishes its first cycle, the T in Bar-Gera's algorithm has already become lower than in Aashtiani's algorithm.

Given the above results, one could conclude that the practitioner would prefer Aashtiani's algorithm in order to deliver faster results.

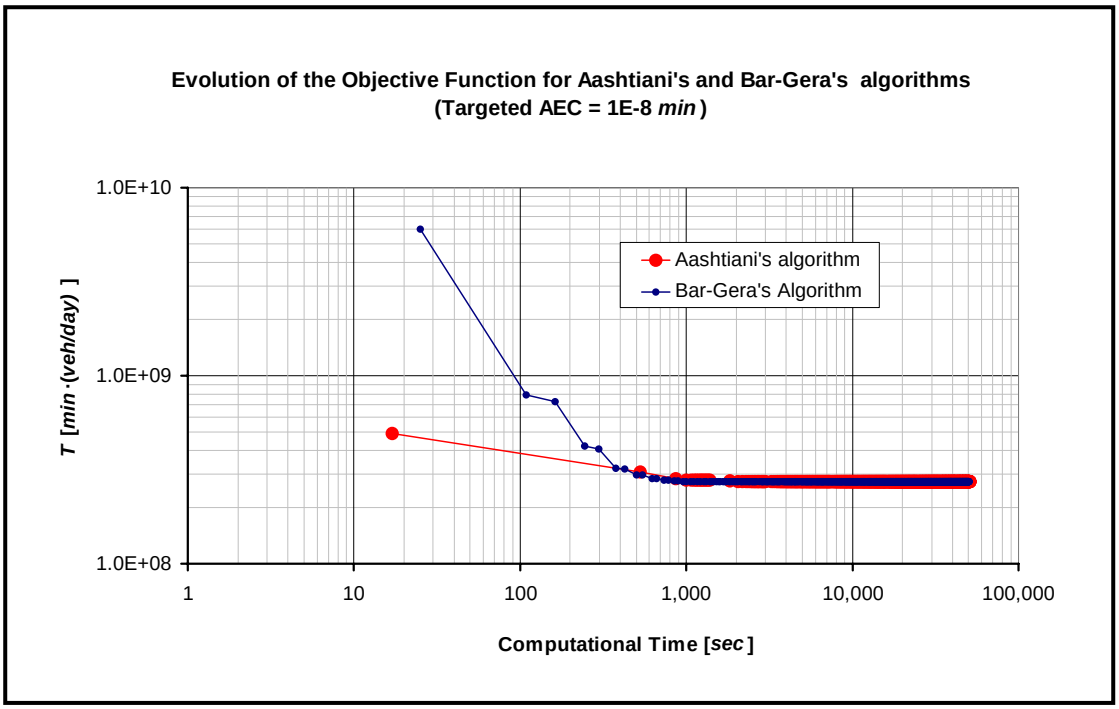


Figure 3-8a. PHILAD network: Evolution of the objective function value for Aashtiani's and Bar-Gera's algorithms.

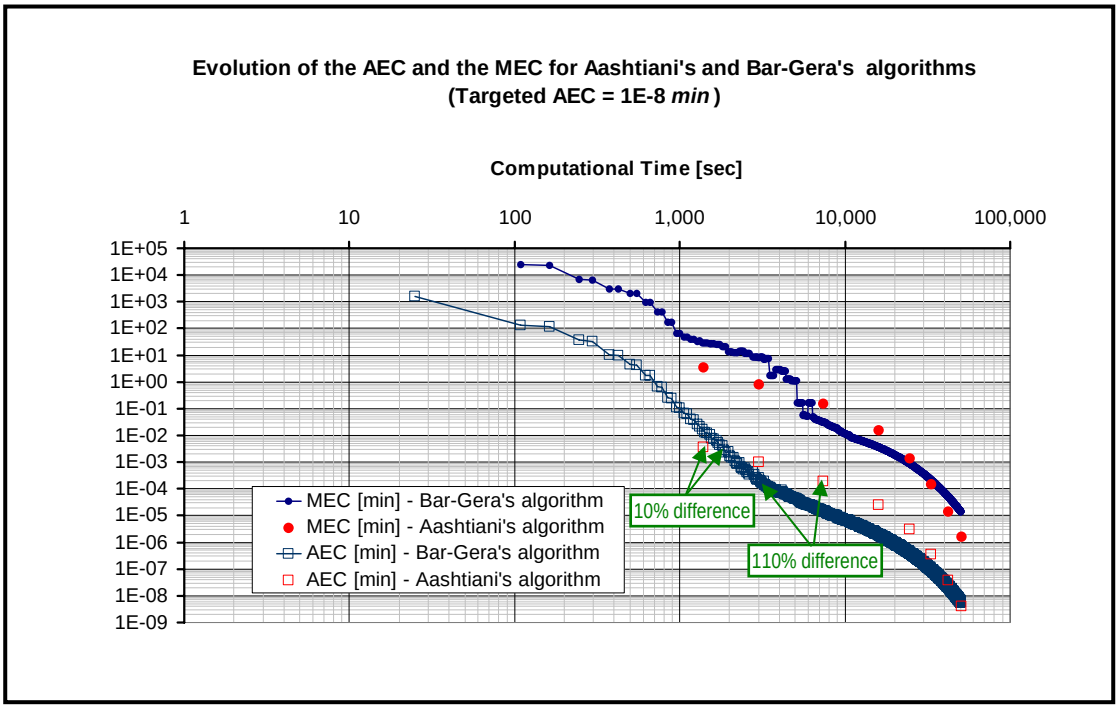


Figure 3-8b. PHILAD network: Evolution of the average excess cost and the maximum excess cost for Aashtiani's and Bar-Gera's algorithms (values in green indicate maximum differences).

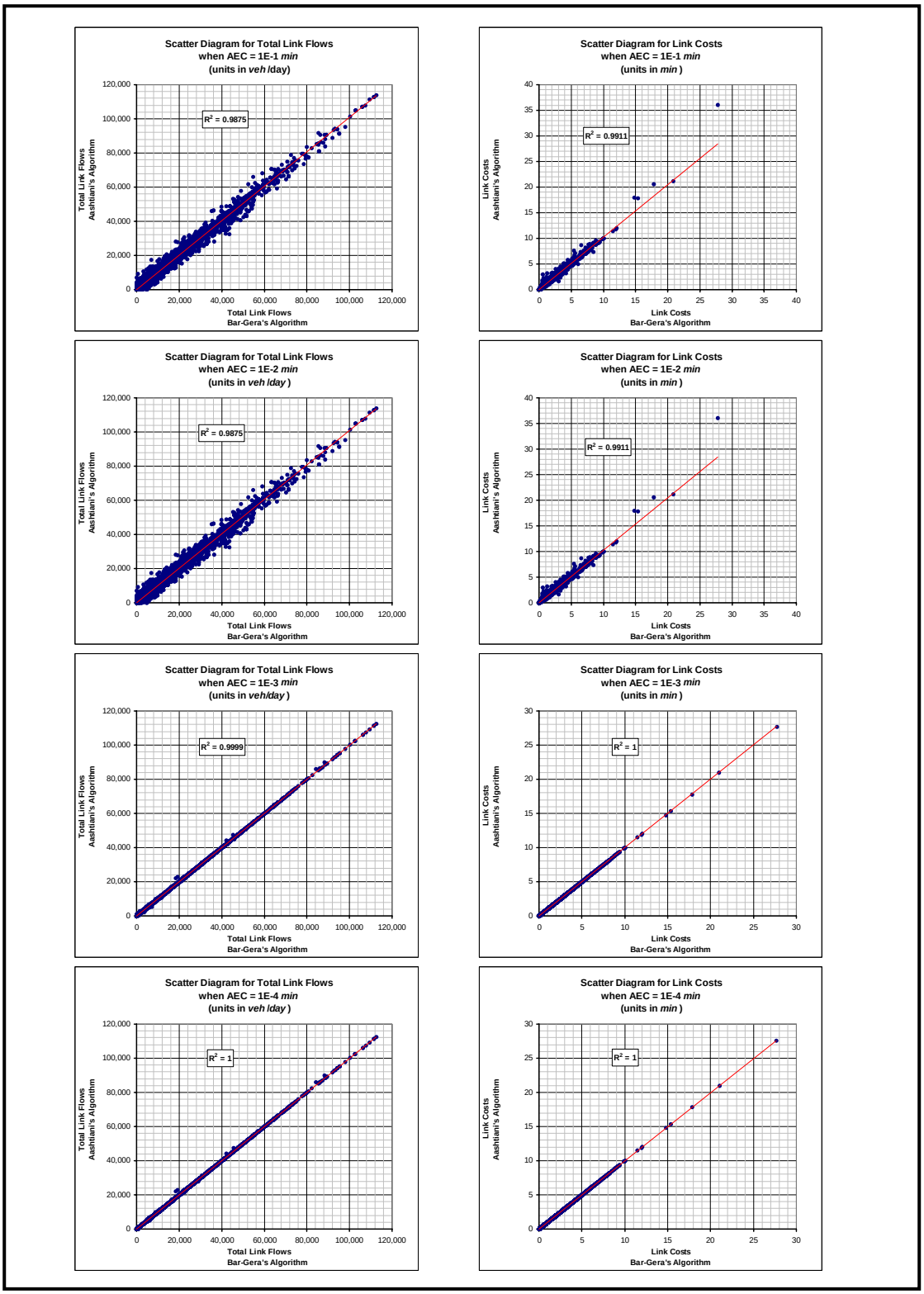


Figure 3-8c. PHILAD network: scatter diagrams and coefficients of determination that compare the total link flows and the link costs calculated by Aashtiani's and Bar-Gera's algorithms when targeting an AEC of 10^{-1}, 10^{-2}, 10^{-3} and 10^{-4} minutes.

Description of the results shown on Figure 3-8a to Figure 3-8c

Network: Philadelphia
Code: PHILAD
Number of Nodes: 13,389
Number of Zones: 1,525
Number of Arcs: 40,003
Number of OD Pairs: 1,149,795
Complexity: 45,995,249,385 arcs·OD pairs
Level of Complexity: 7 (10^{10} to 10^{11})
Total flow (total demand): 18,503,872 vehicles/day
Units for the total link flows f_a: vehicles/day
Units for the link costs t_a: minutes
Existence of tolls: Yes
Distance factor equal to zero: Yes

The units for this network are also known. Again, one could immediately conclude that an AEC of approximately 10^{-1} minutes should render a solution that is accurate enough. This statement is corroborated by the linear trend shown on the scatter diagram for a targeted AEC of 10^{-1} minutes ($R^2 = 0.9875$). An AEC of 10^{-1} minutes or less should be useful for recognizing trends in the curve described on *Figure 3-8b*.

Figure 3-8b shows a more mixed result than with the CHIC_R network. Like with the CHIC_R network, for a rather high AEC (when the AEC is close to 10^{-3} minutes), Bar-Gera's algorithm is 10% slower than Aashtiani's. But unlike the CHIC_R, Aashtiani's algorithm becomes slower than Bar-Gera's until reaching a difference of approximately 110%. And also, unlike the CHIC_R network, Bar-Gera's algorithm ends by being slower by a very small margin. The differences in time are very significant at the beginning (10% ≈ 15 minutes) and when the AEC ranges between 10^{-3} and 10^{-7} minutes (110% ≈ 2.5 hours). The values of the MEC present the same trend shown by the values of the AEC.

Like with the CHIC_R network, Bar-Gera's algorithm started with an initial solution that had a larger T and at the same time slower to compute. Nevertheless, this value quickly diminished. Also, like with the CHIC_R the computational time is very high: from 20 minutes to 4 hours.

Since the units of the link costs are known, one could conclude that the practitioner would prefer Aashtiani's algorithm for delivering faster results at acceptable low accuracies. Nevertheless, if wanting better accuracies, he or she runs the risk of spending two more hours using Aashtiani's algorithm than Bar-Gera's.

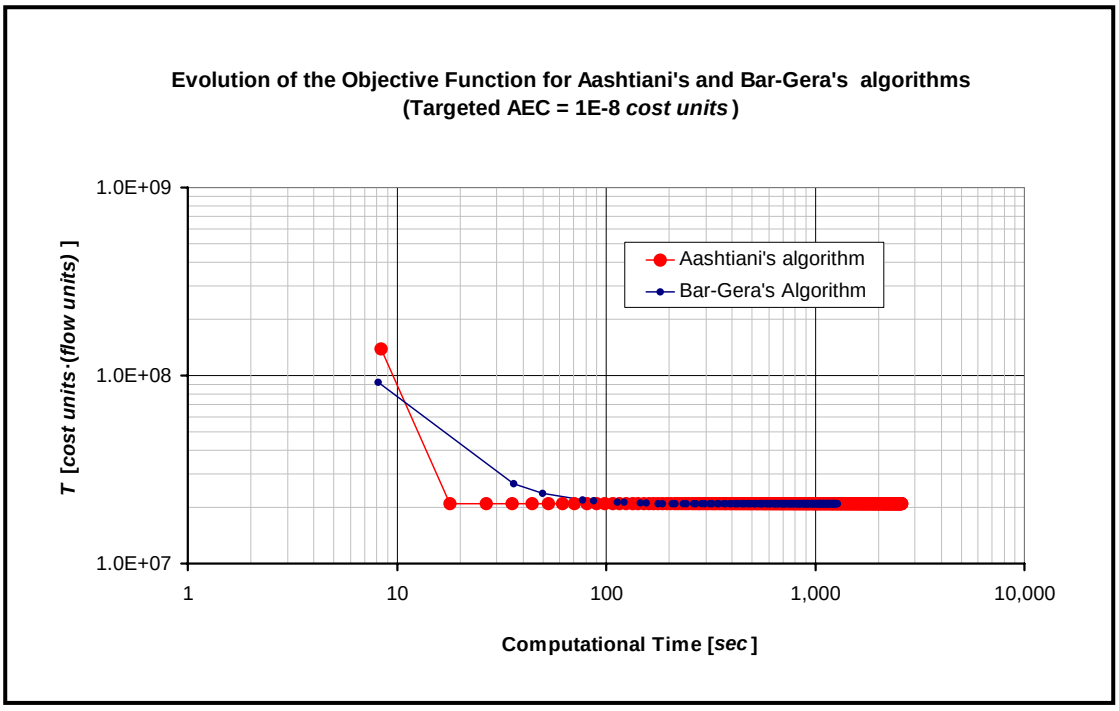


Figure 3-9a. BERL_C network: Evolution of the objective function value for Aashtiani's and Bar-Gera's algorithms.

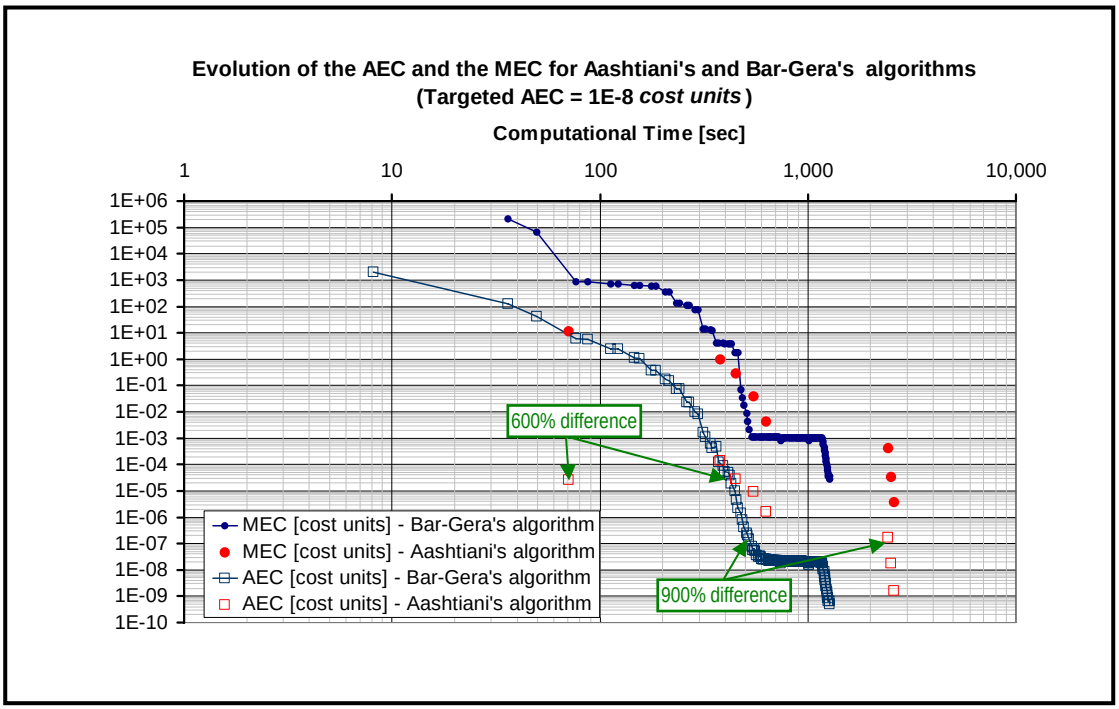


Figure 3-9b. BERL_C network: Evolution of the average excess cost and the maximum excess cost for Aashtiani's and Bar-Gera's algorithms (values in green indicate maximum differences).

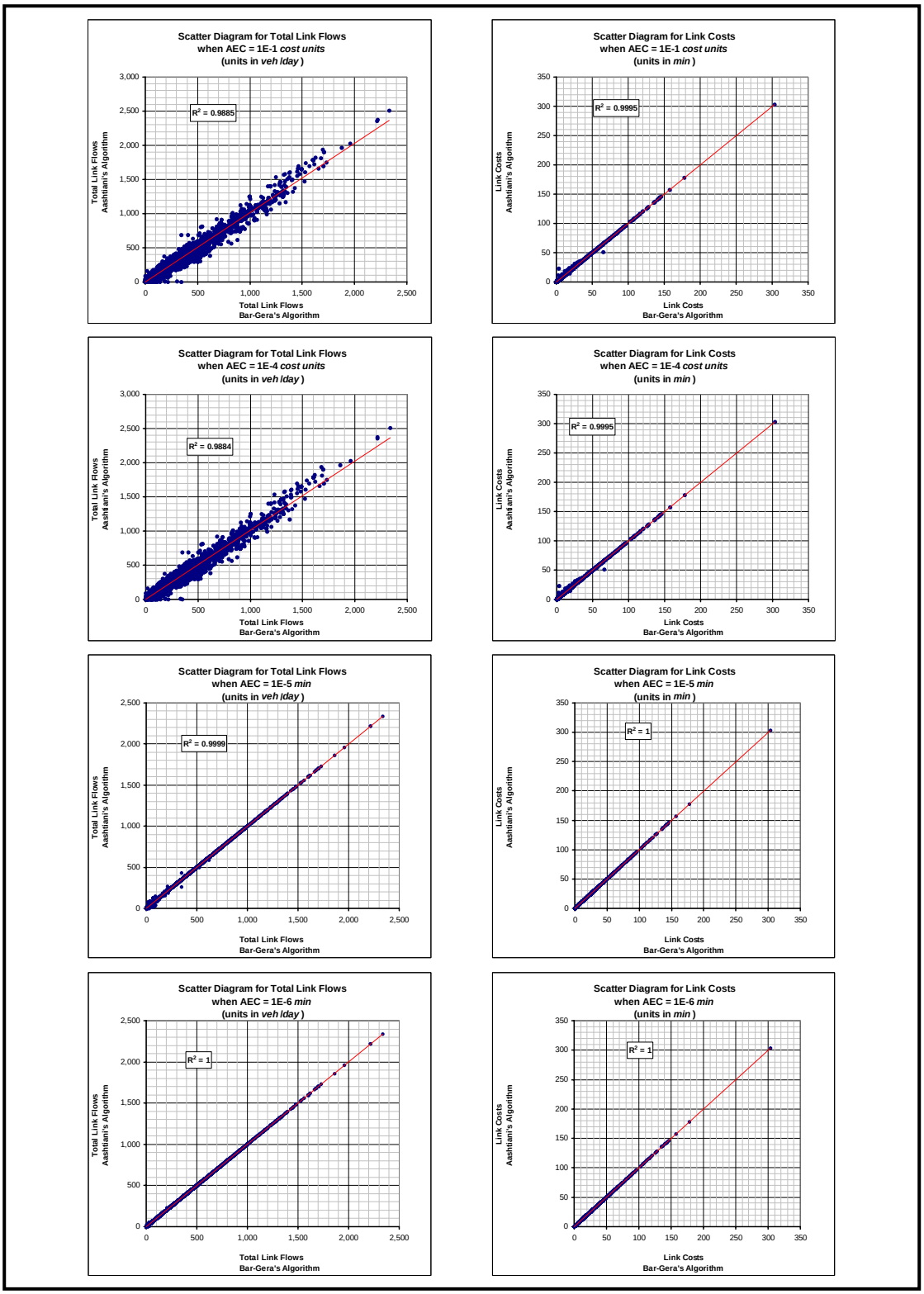


Figure 3-9c. BERL_C network: scatter diagrams and coefficients of determination that compare the total link flows and the link costs calculated by Aashtiani's and Bar-Gera's algorithms when targeting an AEC of 10^{-1}, 10^{-4}, 10^{-5} and 10^{-6} cost units.

Description of the results shown on Figure 3-9a to Figure 3-9c

Network: Berlin Center
Code: BERL_C
Number of Nodes: 12,981
Number of Zones: 865
Number of Arcs: 28,376
Number of OD Pairs: 49,688
Complexity: 1,409,946,688 arcs·OD pairs
Level of Complexity: 6 (10^9 to 10^{10})
Total flow (total demand): 168,222 *flow units*
Units for the total link flows f_a: unknown
Units for the link costs t_a: unknown
Existence of tolls: No
Distance factor equal to zero: Yes

Since the units are not known for this network, one needs to look carefully at the scatter diagrams (*Figure 3-9c*) in order to determine whether an AEC of 10^{-1} is enough for rendering an accurate solution. It seems that it is more difficult to argue that 10^{-1} in the AEC is enough than with the previous two networks. Nevertheless, as explained in the next paragraph, we will only need to consider values in the AEC when the targeted AEC is close to 10^{-5}. As shown in the corresponding scatter diagram (*Figure 3-9c*, when targeted AEC = 10^{-5}), the solutions **f** from both algorithms are close.

The trends in the AEC, as shown on *Figure 3-9b*, are less favorable to Aashtiani's algorithm than in the previous networks. At a not very low AEC of almost 10^{-5}, Bar-Gera's algorithm is almost 600% slower, but the trend quickly shifts: Bar-Gera becomes the fastest algorithm when spending more computational time. At some point, for an AEC close to 10^{-7}, Aashtiani's algorithm is 900% slower. These two extremes in computational time are very significant. As with CHIC_R, Bar-Gera's algorithm increases its speed at every subsequent iteration. It is interesting that at some level, when the AEC ranges between 10^{-4} and 10^{-5}, both algorithms seem equally fast. The values of the MEC corroborate the trends shown by the values in the AEC.

Figure 3-9b presents two additional interesting features. The trends in the MEC and in the AEC show that both algorithms struggle in finding a better solution when trying to reach an AEC close to 10^{-8} in the case of Bar-Gera's algorithm and an AEC close 10^{-7} in the case of Aashtiani's algorithm. *Figure 3-9b* also shows that while Aashtiani's algorithm has a lot of control over the MEC, Bar-Gera's algorithm has control over the AEC. This is observed at the increase in the AEC that Aashtiani's algorithm have from 10^{-5} to 10^{-4} (that is, when the computational time is between100 seconds to 350 seconds). Meanwhile, the MEC in Aashtiani's algorithm is strictly decreasing.

In this network, the computational time ranges from one to 25 minutes approximately. Contrary to the previous networks, Aashtiani's algorithm spent a little bit more time

calculating the initial solution. Also, contrary to the previous networks, the value of T for this initial solution was higher than the initial solution of Bar-Gera's network.

Overall, Bar-Gera's algorithm seems faster. Nevertheless, one cannot discard stating that Aashtiani's algorithm is faster at lower (and perhaps acceptable) levels of accuracy.

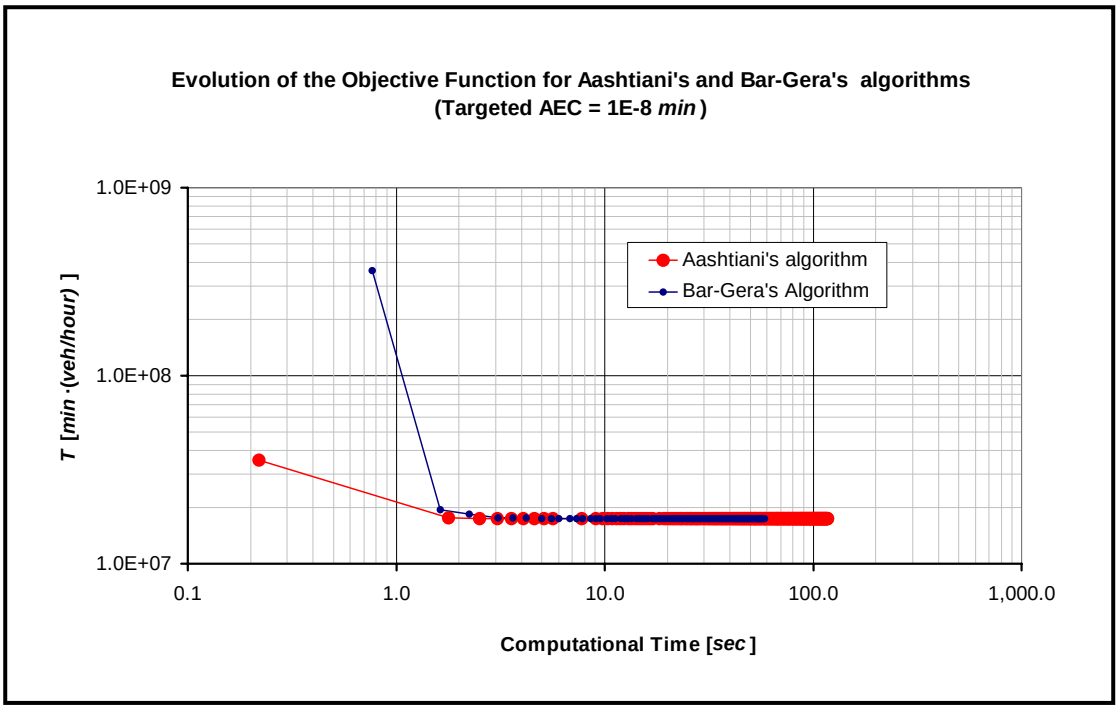


Figure 3-10a. CHIC_S network: Evolution of the objective function value for Aashtiani's and Bar-Gera's algorithms.

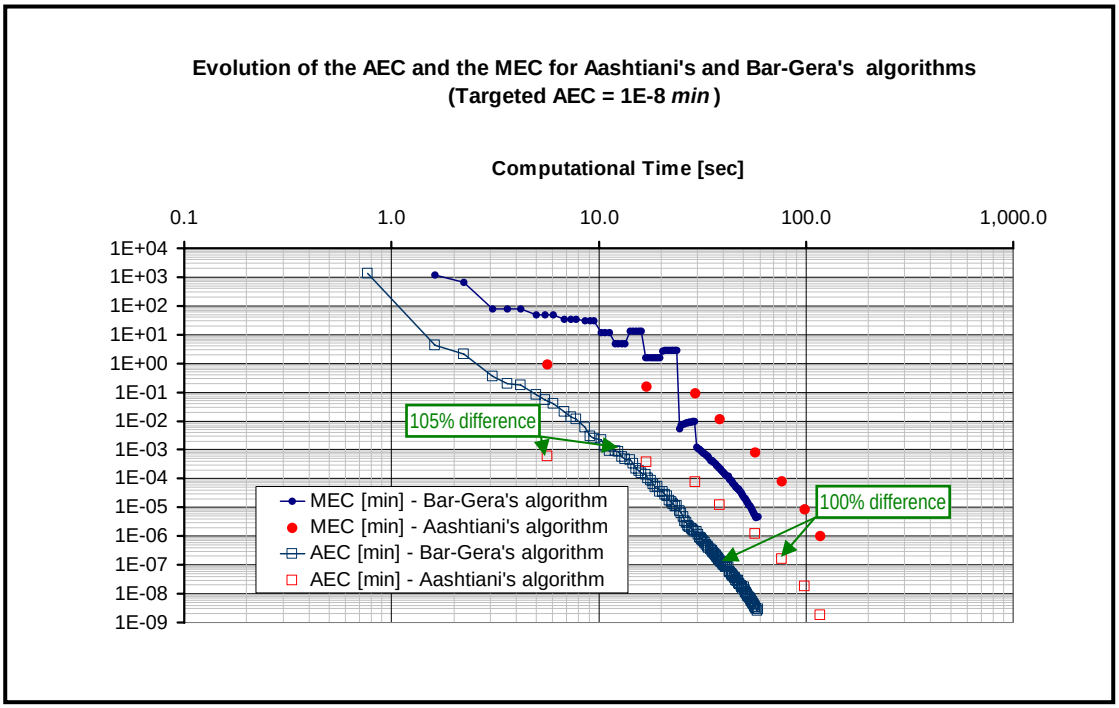


Figure 3-10b. CHIC_S network: Evolution of the average excess cost and the maximum excess cost for Aashtiani's and Bar-Gera's algorithms (values in green indicate maximum differences).

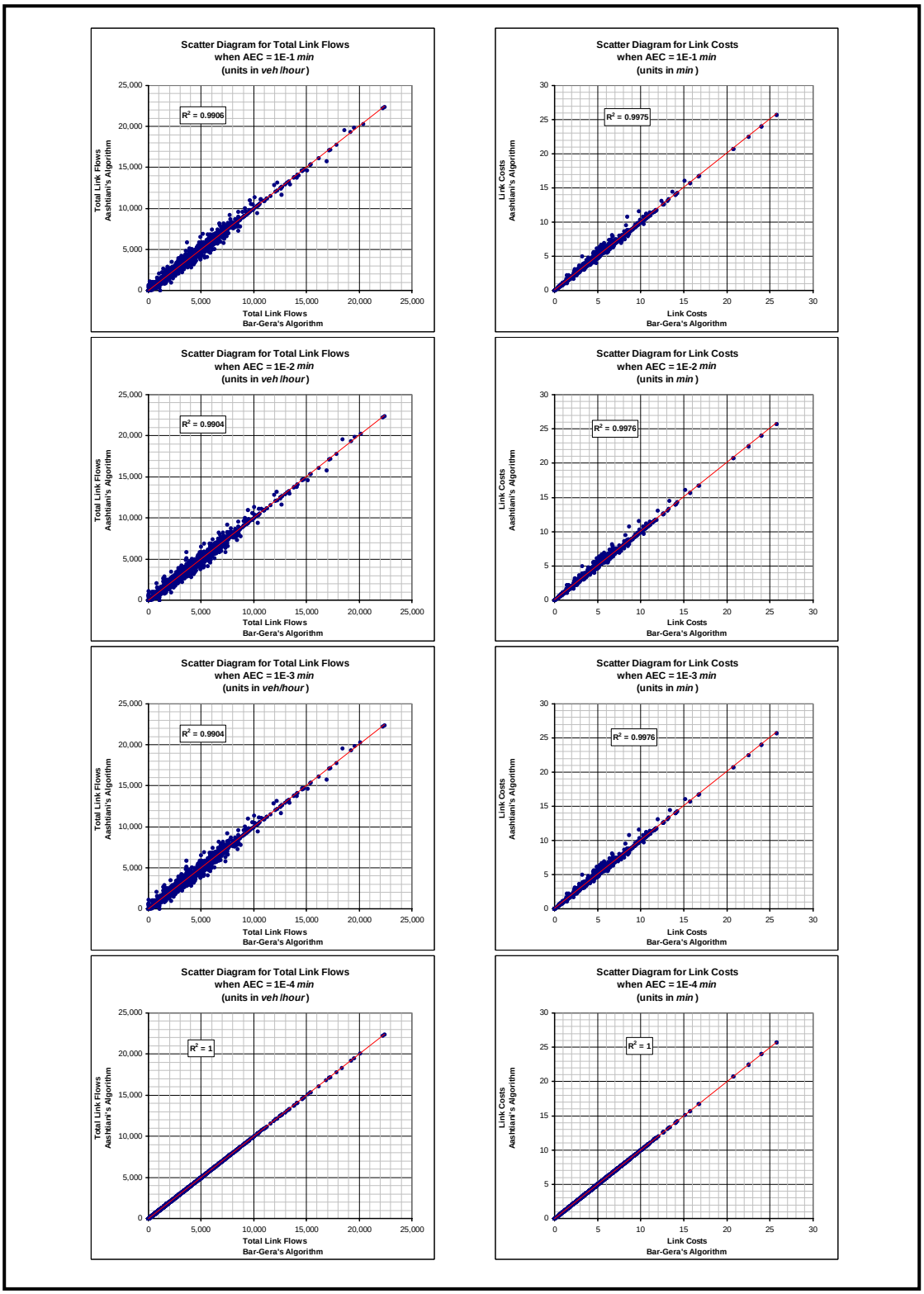


Figure 3-10c. CHIC_S network: scatter diagrams and coefficients of determination that compare the total link flows and the link costs calculated by Aashtiani's and Bar-Gera's algorithms when targeting an AEC of 10^{-1}, 10^{-2}, 10^{-3} and 10^{-4} minutes.

Description of the results shown on Figure 3-10a to Figure 3-10c

Network: Chicago Sketch
Code: CHIC_S
Number of Nodes: 933
Number of Zones: 387
Number of Arcs: 2,950
Number of OD Pairs: 142,512
Complexity: 420,410,400 arcs·OD pairs
Level of Complexity: 5 (10^8 to 10^9)
Total flow (total demand): 1,260,907 vehicles/hour
Units for the total link flows f_a: vehicles/hour
Units for the link costs t_a: minutes
Existence of tolls: No
Distance factor equal to zero: Yes

The scatter diagrams confirm that a targeted AEC of 10^{-1} minutes is enough for rendering a satisfactory solution. For example, the scatter diagram of link costs (when the targeted AEC is equal to 10^{-1} minutes) shows a difference of three minutes approximately in the worst case between costs t_a of a same link a. The coefficient of determination for the total link flows for a targeted AEC of 10^{-1} is $R^2 = 0.9906$. Therefore, values less than or equal to 10^{-1} minutes should not be discarded when looking for trends in the AEC curve.

The trend described by *Figure 3-10b* in terms of AEC is as follows. For a not very low AEC close to 10^{-3} minutes, Bar-Gera's algorithm is 105% slower than Aashtiani's. Then, Bar-Gera's algorithm increases its rate of convergence and quickly surpasses Aashtiani's. At the end, for an AEC close to 10^{-8} minutes, Aashtiani's algorithm is 100% slower than Bar-Gera's. These differences are very significant in terms of percentage values. The values of the MEC corroborate the trend seen on the AEC. Nevertheless, the trend in the MEC is more favorable to Aashtiani's algorithm for low accuracies. This feature is due to the high control that Aashtiani's algorithm over the MEC.

In sum, the results in this network are very similar to the BERL_C network. But in this case, there is more certainty when stating that the initial results have enough accuracy.

Like with most of the previous networks, Bar-Gera's initial solution had a higher value in the objective function, when compared to Aashtiani's initial solution, and it was calculated within less time. As always, by the time Aashtiani's algorithm completes its first cycle, Bar-Gera's algorithm has already generated a solution with a very similar T.

In this network, the computational time ranges from 5 seconds to 2 minutes approximately.

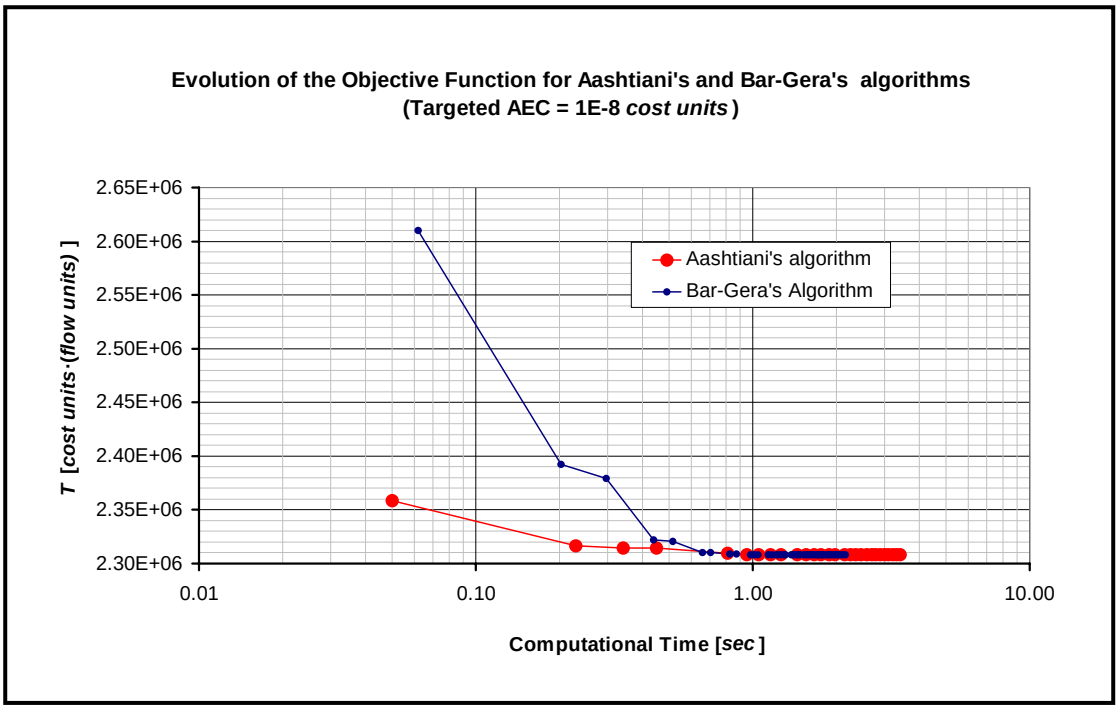


Figure 3-11a. M_P_F network: Evolution of the objective function value for Aashtiani's and Bar-Gera's algorithms.

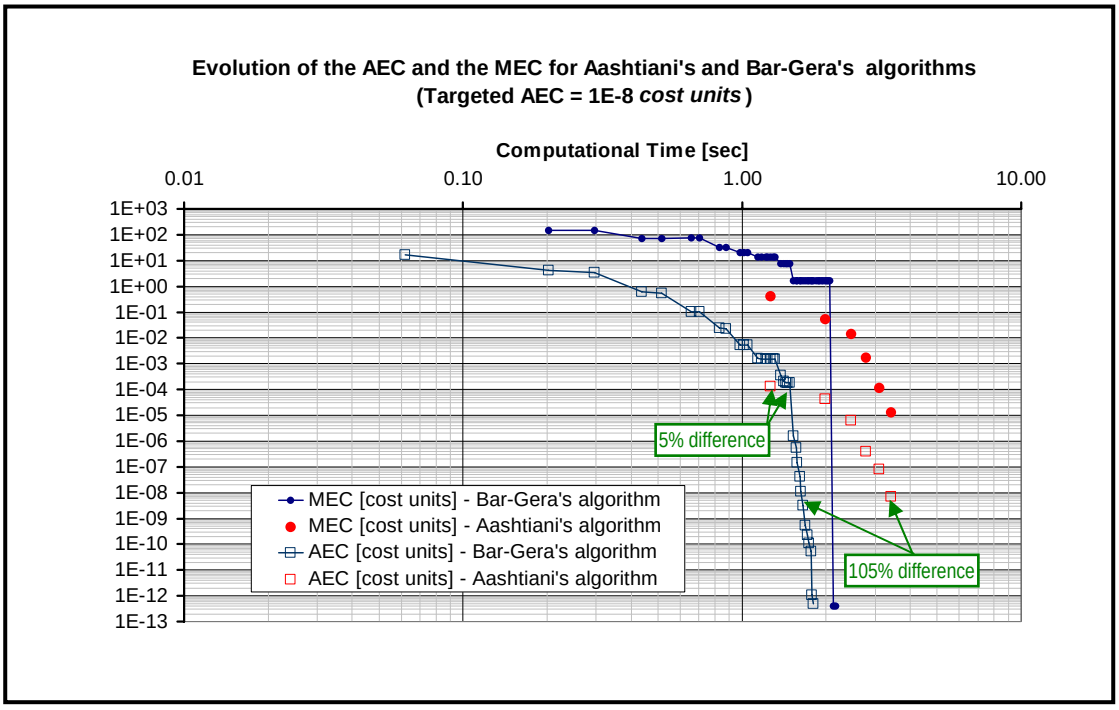


Figure 3-11b. M_P_F network: Evolution of the average excess cost and the maximum excess cost for Aashtiani's and Bar-Gera's algorithms (values in green indicate maximum differences).

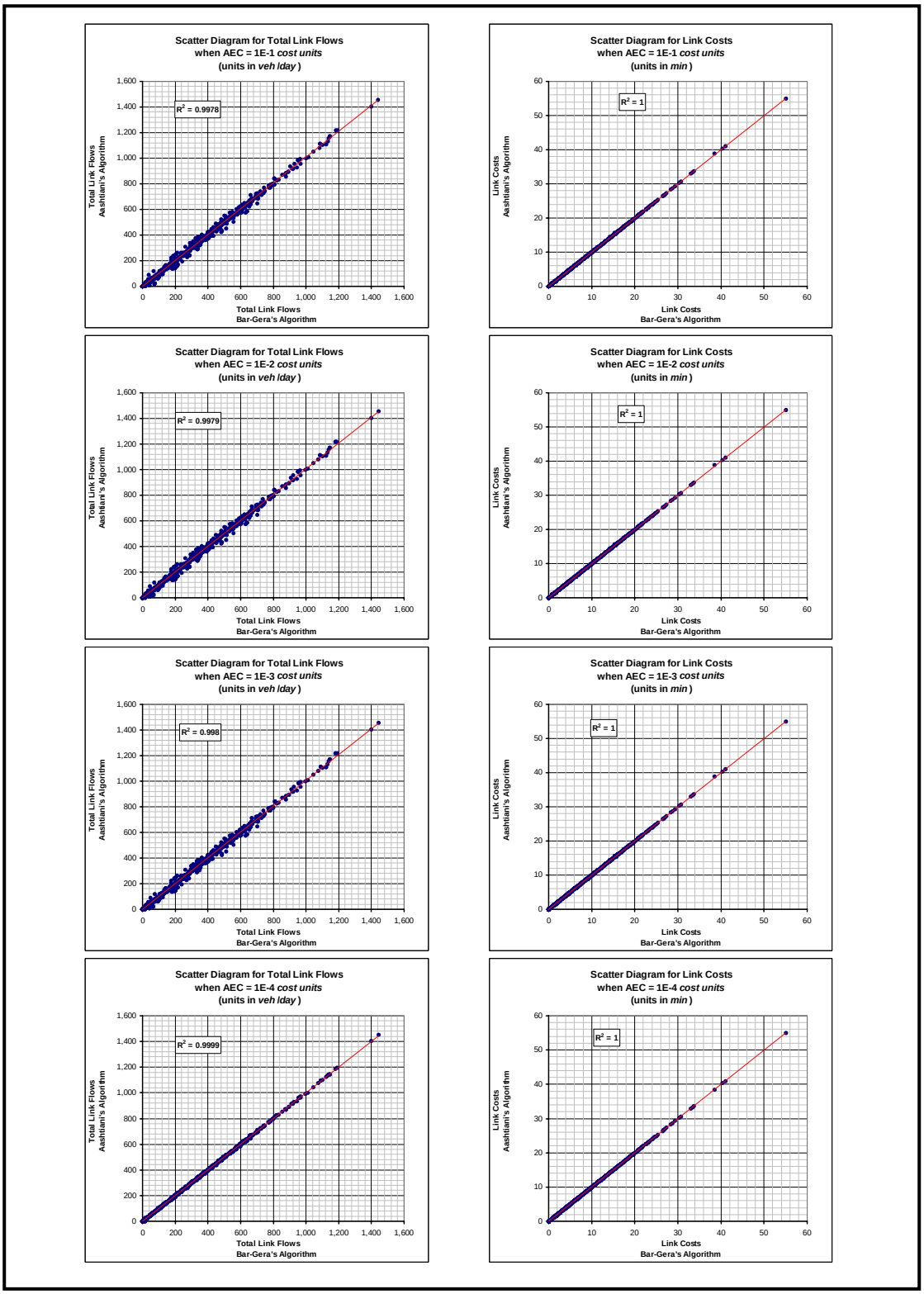


Figure 3-11c. M_P_F network: scatter diagrams and coefficients of determination that compare the total link flows and the link costs calculated by Aashtiani's and Bar-Gera's algorithms when targeting an AEC of 10^{-1}, 10^{-2}, 10^{-3} and 10^{-4} cost units.

Description of the results shown on Figure 3-11a to Figure 3-11c

Network: Mitte, Prenzlauer Berg and Friedricshain
Code: M_P_F
Number of Nodes: 974
Number of Zones: 98
Number of Arcs: 2,184
Number of OD Pairs: 9,505
Complexity: 20,758,920 arcs·OD pairs
Level of Complexity: 4 (10^8 to 10^9)
Total flow (total demand): 23,648 *flow units*
Units for the total link flows f_a: unknown
Units for the link costs t_a: unknown
Existence of tolls: No
Distance factor equal to zero: Yes

Arguably, *Figure 3-11c* reveal that a solution of an AEC of approximately equal to 10^{-1} is precise enough (R^2 = 0.9978). Nonetheless, we will only need to consider values in the AEC close to 10^{-4}. As shown in the corresponding scatter diagram (*Figure 3-9c*, when targeted AEC = 10^{-5}), the solutions are very accurate (R^2 = 0.9999).

The trends in the AEC are very similar to the two previous networks (BERL_C and CHIC_S): Aashtiani's algorithm starts by being faster (5% difference in computational time) and then Bar-Gera's becomes the fastest (105% difference). The differences are very significant. Bar-Gera's algorithm is always faster at every subsequent iteration confirming a trend that seen in all the previous networks except (perhaps) PHILAD. The curve described by the MEC corroborates the trend shown by the curve of the AEC.

According to *Figure 3-11a* and as with the previous networks (except BERL_C), Aashtiani's algorithm starts with a better initial solution but Bar-Gera's algorithm very quickly catches up.

In this network, the computational time ranges from one to three seconds approximately, a big decrease from the previous network which is one upper level of complexity.

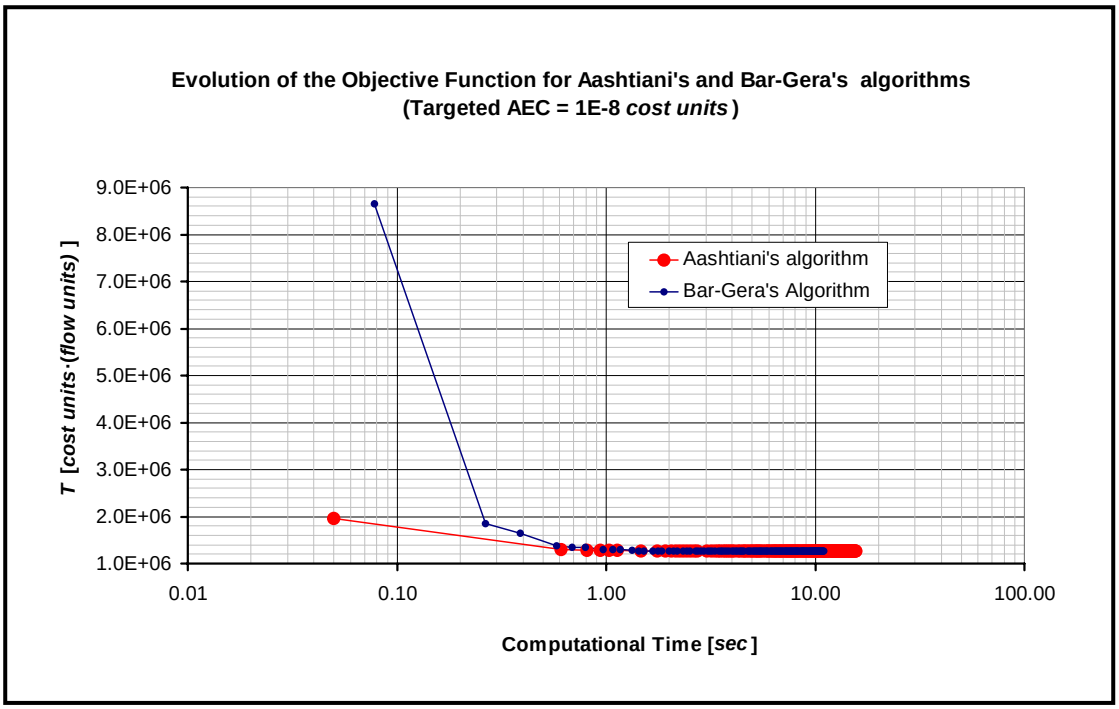


Figure 3-12a. BARCEL network: Evolution of the objective function value for Aashtiani's and Bar-Gera's algorithms.

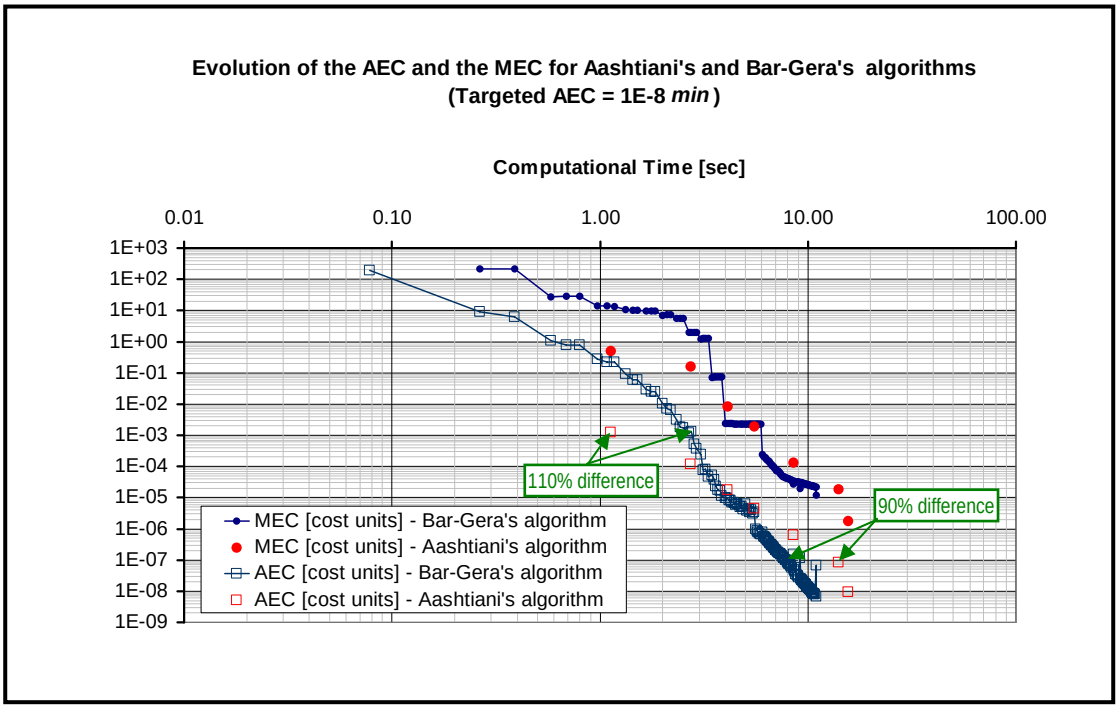


Figure 3-12b. BARCEL network: Evolution of the average excess cost and the maximum excess cost for Aashtiani's and Bar-Gera's algorithms (values in green indicate maximum differences).

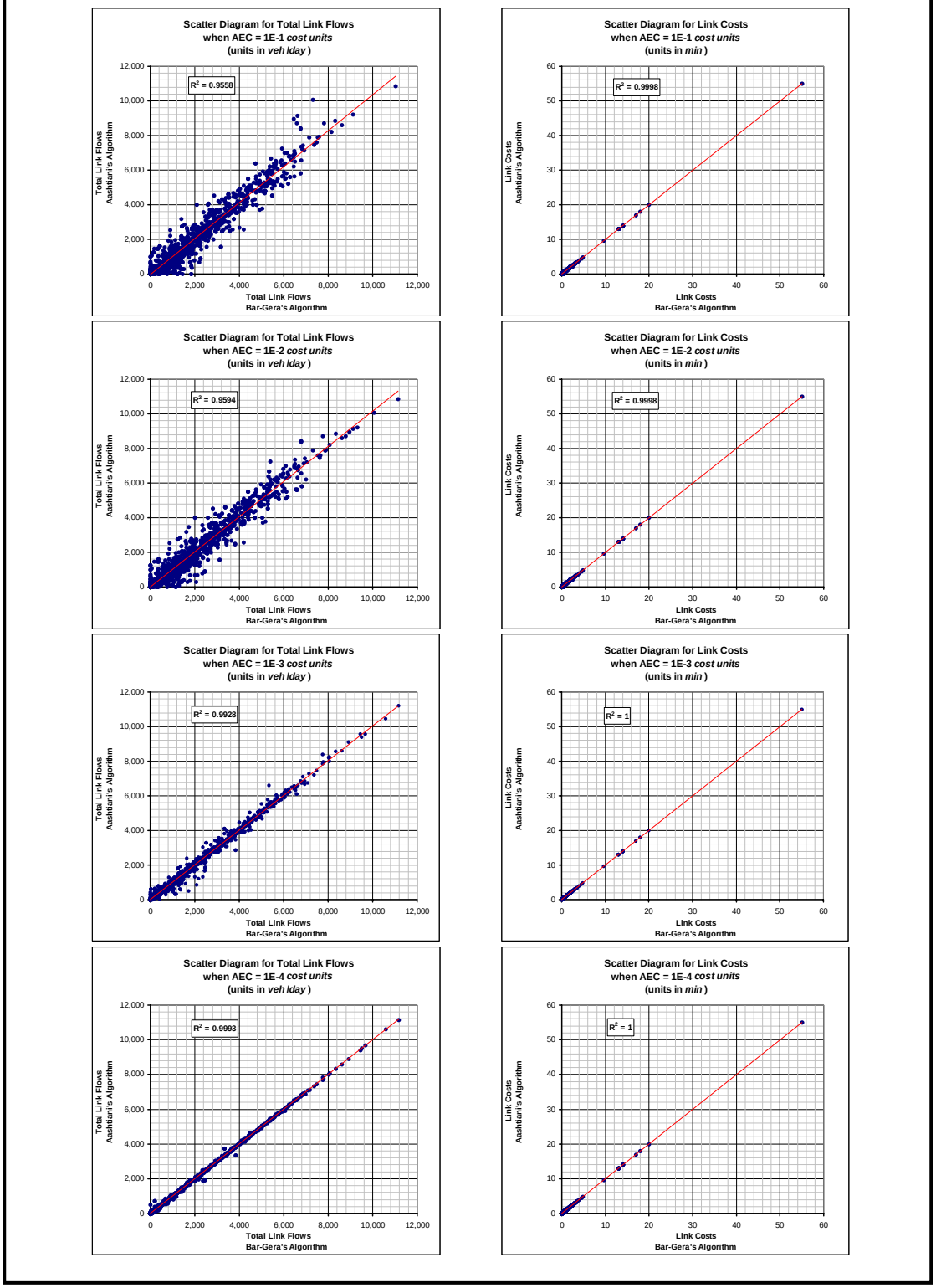


Figure 3-12c. BARCEL network: scatter diagrams and coefficients of determination that compare the total link flows and the link costs calculated by Aashtiani's and Bar-Gera's algorithms when targeting an AEC of 10^{-1}, 10^{-2}, 10^{-3} and 10^{-4} cost units.

Description of the results shown on Figure 3-12a to Figure 3-12c

Network: Barcelona
Code: BARCEL
Number of Nodes: 930
Number of Zones: 110
Number of Arcs: 2,522
Number of OD Pairs: 7,922
Complexity: 19,979,284 arcs·OD pairs
Level of Complexity: 4 (10^8 to 10^9)
Total flow (total demand): 184,679 *flow units*
Units for the total link flows f_a: unknown
Units for the link costs t_a: unknown
Existence of tolls: No
Distance factor equal to zero: Yes

The scatter diagram of the total link flows (*Figure 3-12c*) does not reveal with complete assurance that an AEC of 10^{-1} is accurate enough. At a targeted AEC of 10^{-2}, the accuracy improves ($R^2 = 0.9928$). But at a targeted AEC of 10^{-3}, the solution is very accurate ($R^2 = 0.9993$).

The trends that *Figure 3-12b* describes were already explained in the previous section (see page 34). For convenience to the reader, that analysis is restated here verbatim: For an AEC less than or equal to 10^{-3}, Aashtiani's algorithm starts by being the fastest. When the AEC is approximately 10^{-3}, Bar-Gera's algorithm is slower. No algorithm is faster than the other always; there is no strong superiority of one algorithm over the other. While Aashtiani's algorithm ends by being 95% slower than Bar-Gera's, Bar-Gera's algorithm starts by being 160% slower. Both algorithms seem equally fast for an AEC between 10^{-5} and 10^{-6}. Overall, we can observe that Bar-Gera's algorithm becomes faster as one allows more computational time. The differences are fairly significant since they represent values greater than 90%. The values of the MEC corroborate the values of the AEC. When Aashtiani's algorithm is slower in terms of the AEC, so it is in terms of the MEC.

The trends seen on this network are similar to BERL_C, CHIC_R and M_P_F. Overall, Bar-Gera's algorithm seem to be faster. Nevertheless, one cannot discard stating that Aashtiani's algorithm is faster at lower (and acceptable) accuracies.

According to *Figure 3-12a* and as with the previous networks (except BERL_C), Aashtiani's algorithm starts with a better initial solution but Bar-Gera's algorithm very quickly catches up. Aashtiani's initial solution is obtained within less computational time.

In this network, the computational time ranges from one to 12 seconds approximately: slightly different to M_P_F which had the same level of complexity.

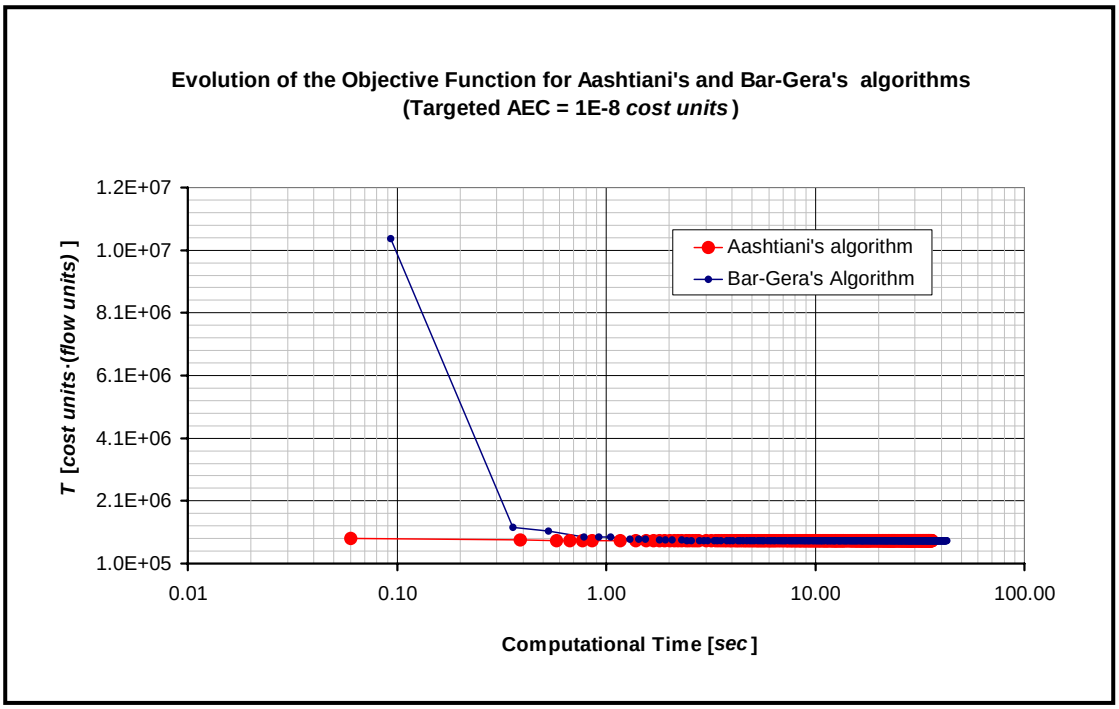


Figure 3-13a. WINNIP network: Evolution of the objective function value for Aashtiani's and Bar-Gera's algorithms.

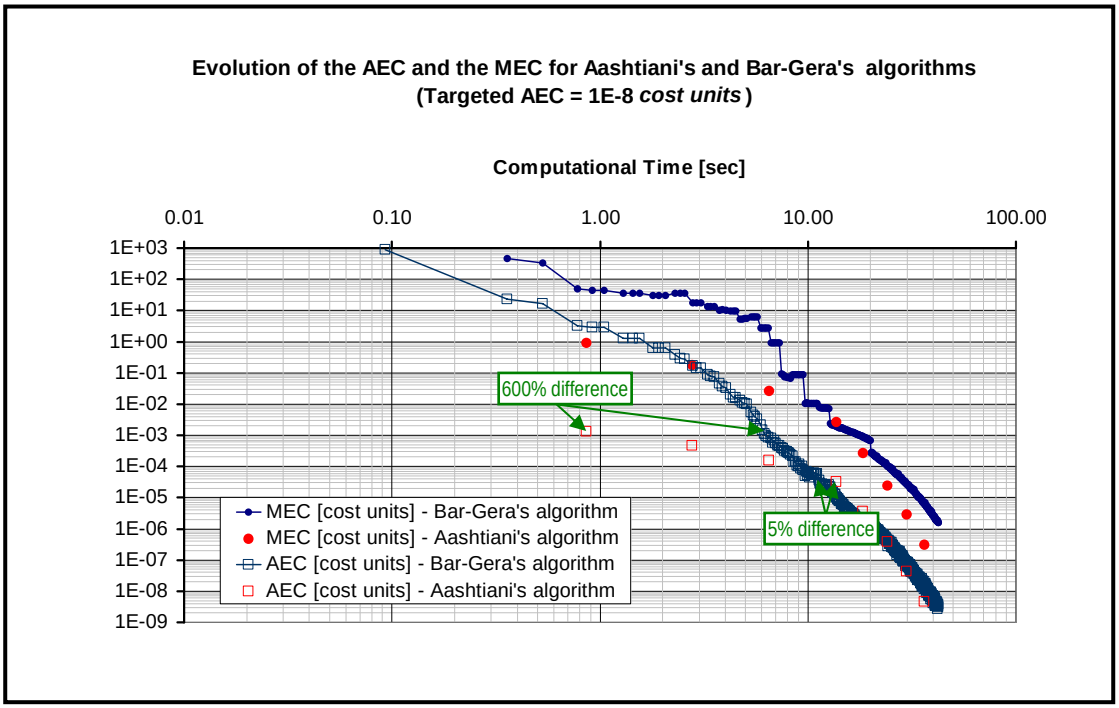


Figure 3-13b. WINNIP network: Evolution of the average excess cost and the maximum excess cost for Aashtiani's and Bar-Gera's algorithms (values in green indicate maximum differences).

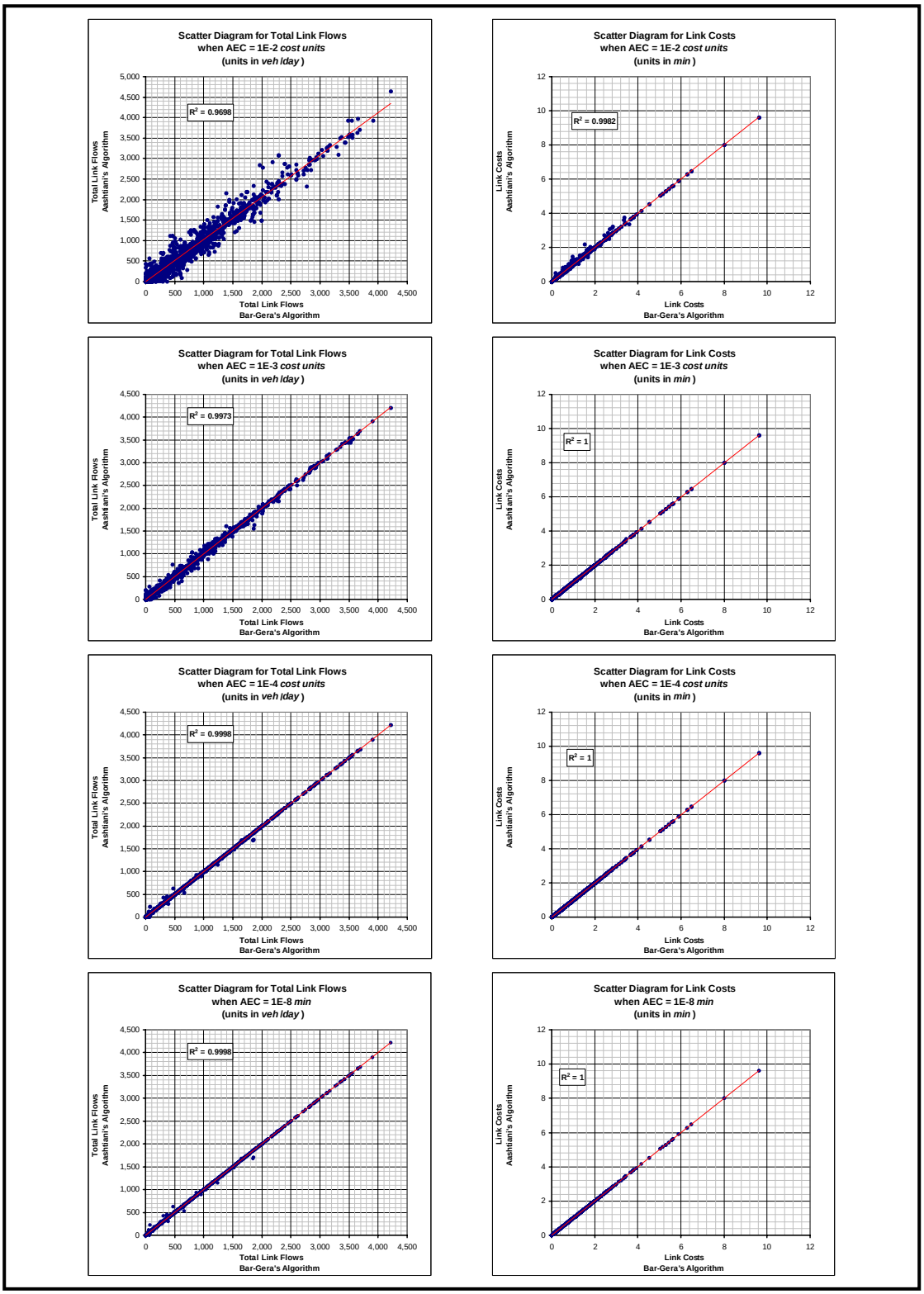


Figure 3-13c. WINNIP network: scatter diagrams and coefficients of determination that compare the total link flows and the link costs calculated by Aashtiani's and Bar-Gera's algorithms when targeting an AEC of 10^{-1}, 10^{-2}, 10^{-3} and 10^{-8} cost units.

Description of the results shown on Figure 3-13a to Figure 3-13c

Network: Winnipeg
Code: WINNIP
Number of Nodes: 1,040
Number of Zones: 154
Number of Arcs: 2,836
Number of OD Pairs: 4,344
Complexity: 12,319,584 arcs·OD pairs
Level of Complexity: 4 (10^7 to 10^8)
Total flow (total demand): 64,784 *flow units*
Units for the total link flows f_a: unknown
Units for the link costs t_a: unknown
Existence of tolls: No
Distance factor equal to zero: Yes

This network presents a special characteristic that the other networks do not have. WINNIP contains links with link performance functions equal to zero. As mentioned previously in the *Section Discussion on the Assumptions Required by Both Methods*, connectors can have null performance functions. But this network presents, besides its connectors, other links with null performance functions. This feature leads to many solutions **f**. This characteristic explains why when the targeted AEC is as extremely low as 10^{-8}, the link flows are still not completely equal. Nevertheless, the number of these links is very small to deter the algorithms from finding an almost equal solution.

Like with the BARCEL network, the scatter diagram of the total link flows do not describe a well define straight line when the targeted AEC is equal to 10^{-1}. The straight line starts to emerge at a targeted AEC of 10^{-3} ($R^2 = 0.9973$) and definitely at a targeted AEC of 10^{-4} ($R^2 = 0.9998$).

The trends described by the AECs are very similar to the PHILAD network but a little bit more favorable to Aashtian's algorithm: Aashtiani's algorithm is the fastest for high values in the AEC, then Bar-Gera's algorithm becomes the fastest and finally, at the end, Aashtiani's algorithm becomes slightly better. Regarding the maximum differences presented, for a not very low AEC of 10^{-3}, Bar-Gera's algorithm is 600% slower. For a medium AEC of approximately 10^{-5}, Aashtiani's algorithm is around 5% slower. At other values, both algorithms have almost the same performance. Unlike with the PHILAD network, the values of the MEC corroborate the trends seen on the values of the AECs. Very differently from the other networks, Bar-Gera's algorithm does not present an increasing better performance at every subsequent iteration. Even if at an AEC of 10-5 Bar-Gera's algorithm surpasses the other algorithm, Bar-Gera's algorithm becomes the slowest again at the end.

Another similarity between the PHILAD network and this one is that for a significant range of AECs (10^{-5} to 10^{-7}), Aashtiani's algorithm and Bar-Gera's algorithm are practically equally fast.

The curve of *T*, as shown on *Figure 3-12a*, is very similar to the one seen in BARCEL: the initial solution of Aashtiani's algorithm is faster and more rapidly

calculated (but by only 0.06 seconds). Again, Bar-Gera's algorithm quickly (within 0.9 seconds) generates a solution with a similar value of *T*.

In this network, the computational time for obtaining a meaningful solution ranges from one to 40 seconds approximately: similar to BARCEL which had the same level of complexity.

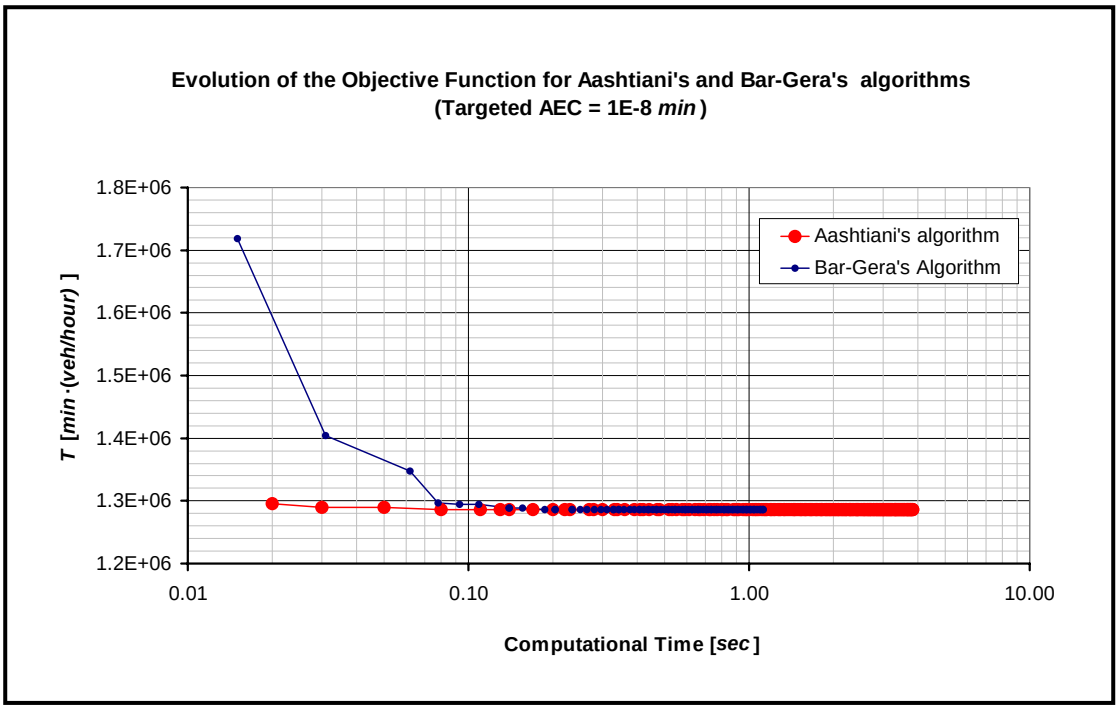


Figure 3-14a. ANAH network: Evolution of the objective function value for Aashtiani's and Bar-Gera's algorithms.

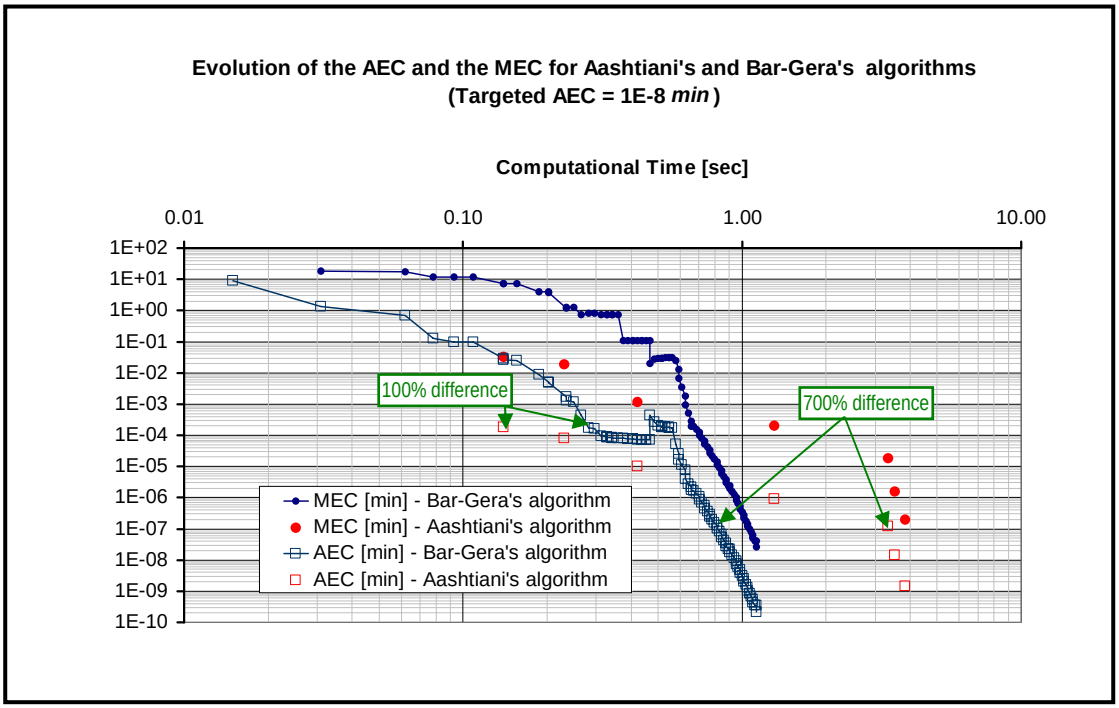


Figure 3-14b. ANAH network: Evolution of the average excess cost and the maximum excess cost for Aashtiani's and Bar-Gera's algorithms (values in green indicate maximum differences).

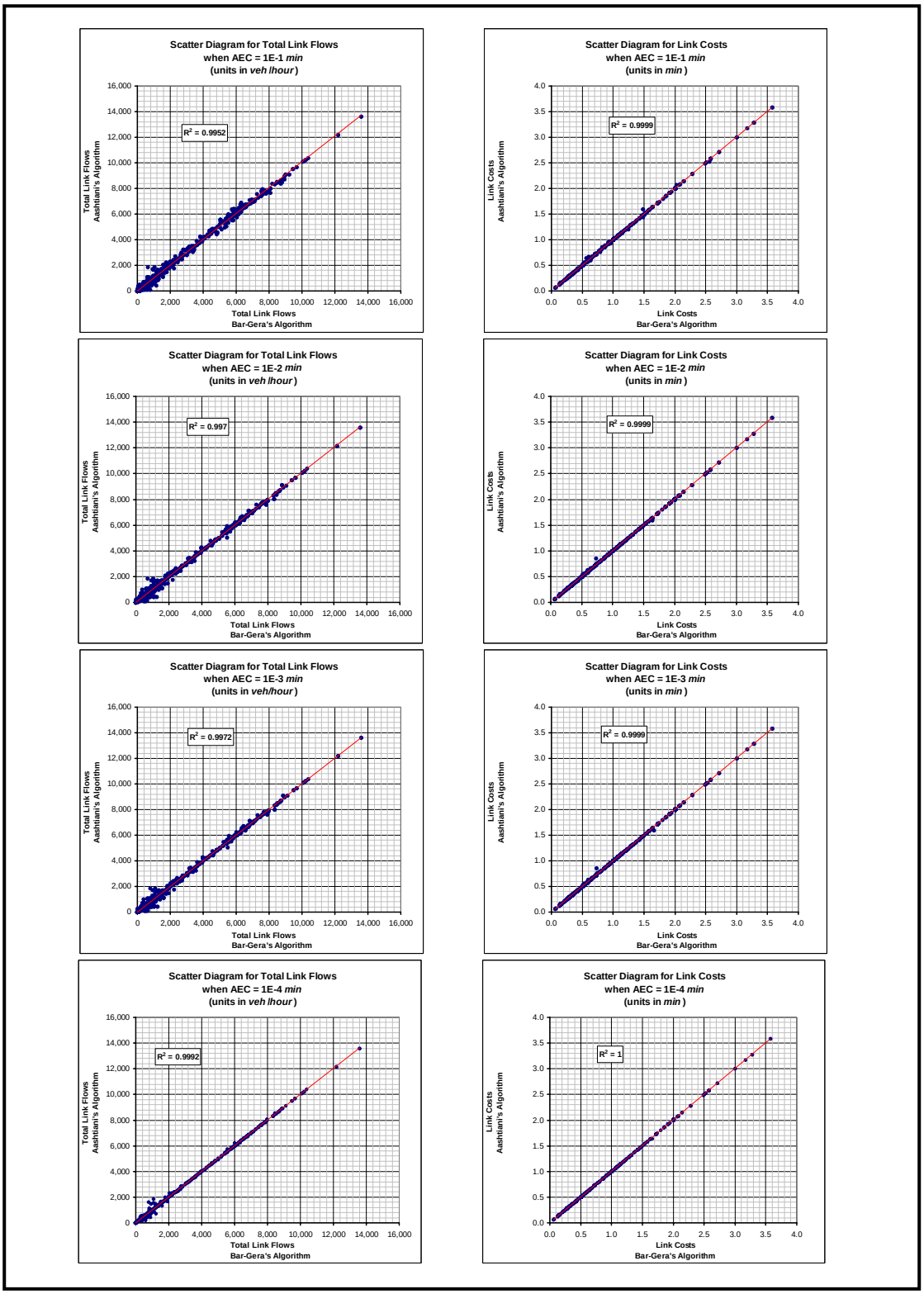


Figure 3-14c. ANAH network: scatter diagrams and coefficients of determination that compare the total link flows and the link costs calculated by Aashtiani's and Bar-Gera's algorithms when targeting an AEC of 10^{-1}, 10^{-2}, 10^{-3} and 10^{-4} minutes.

Description of the results shown on Figure 3-14a to Figure 3-14c

Network: Anaheim
Code: ANAH
Number of Nodes: 416
Number of Zones: 38
Number of Arcs: 914
Number of OD Pairs: 1,406
Complexity: 1,285,084 arcs·OD pairs
Level of Complexity: 3 (10^7 to 10^8)
Total flow (total demand): 104,694 vehicles/hour
Units for the total link flows f_a: vehicles/hour
Units for the link costs t_a: minutes
Existence of tolls: No
Distance factor equal to zero: Yes

Knowledge of the units for this network allows to state that an AEC of 10^{-1} minutes (or 10^{-2} minutes since this is a small network) would guarantee a solution that is precise enough. The scatter diagram of the total link flows for a targeted AEC of 10^{-1} minutes supports this statement (in this case, $R^2 = 0.9952$).

The trends described by the AECs are similar to those of BERL_C, CHIC_S, BARCEL and M_P_F: Aashtiani's algorithm is the fastest for high values in the AEC and then Bar-Gera's algorithm becomes the fastest. There is a 100% difference in computational time when the AEC approximates 10^{-4} minutes and towards the end, there is a 700% difference when the AEC approximates to 10^{-7} minutes. Both differences are very significant. As with the previous networks (except for Winnipeg), Bar-Gera's algorithm is faster at every subsequent iteration. The results in the MECs corroborate the trends seen in the AECs.

This is the second network, besides BERL_C, where Bar-Gera's algorithm required less time computing its initial solution. As with all the other networks where the value of T is greater in the initial solution of Bar-Gera's algorithm than in the one of Aashtini's, Bar-Gera's algorithm quickly obtains a solution with a similar T.

In this network, the computational time for obtaining a meaningful solution ranges from 0.12 seconds to 4 seconds approximately. Up to this point, computational times seem to be proportional to the level of complexity.

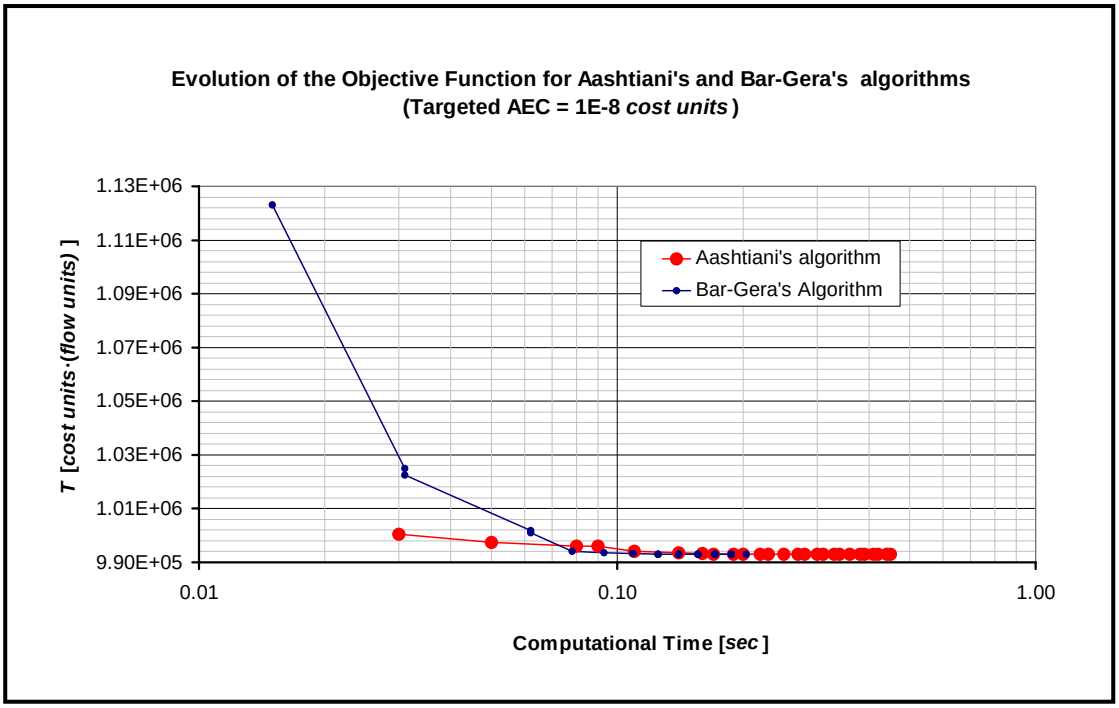


Figure 3-15a. MIT_C network: Evolution of the objective function value for Aashtiani's and Bar-Gera's algorithms.

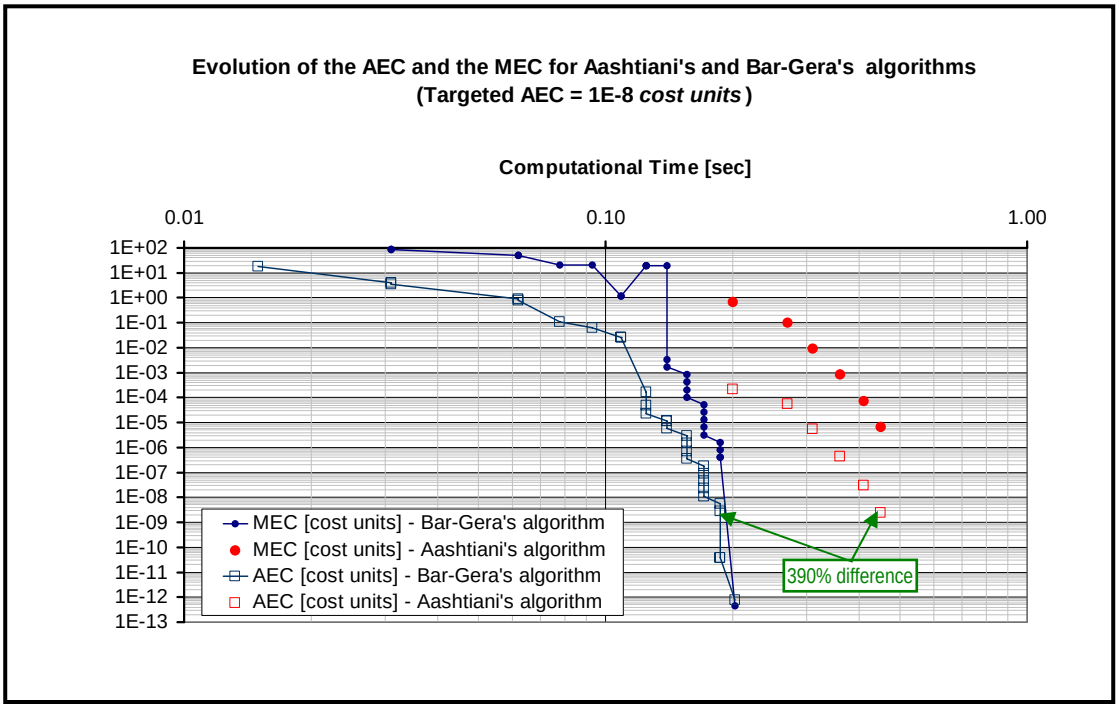


Figure 3-15b. MIT_C network: Evolution of the average excess cost and the maximum excess cost for Aashtiani's and Bar-Gera's algorithms (values in green indicate maximum differences).

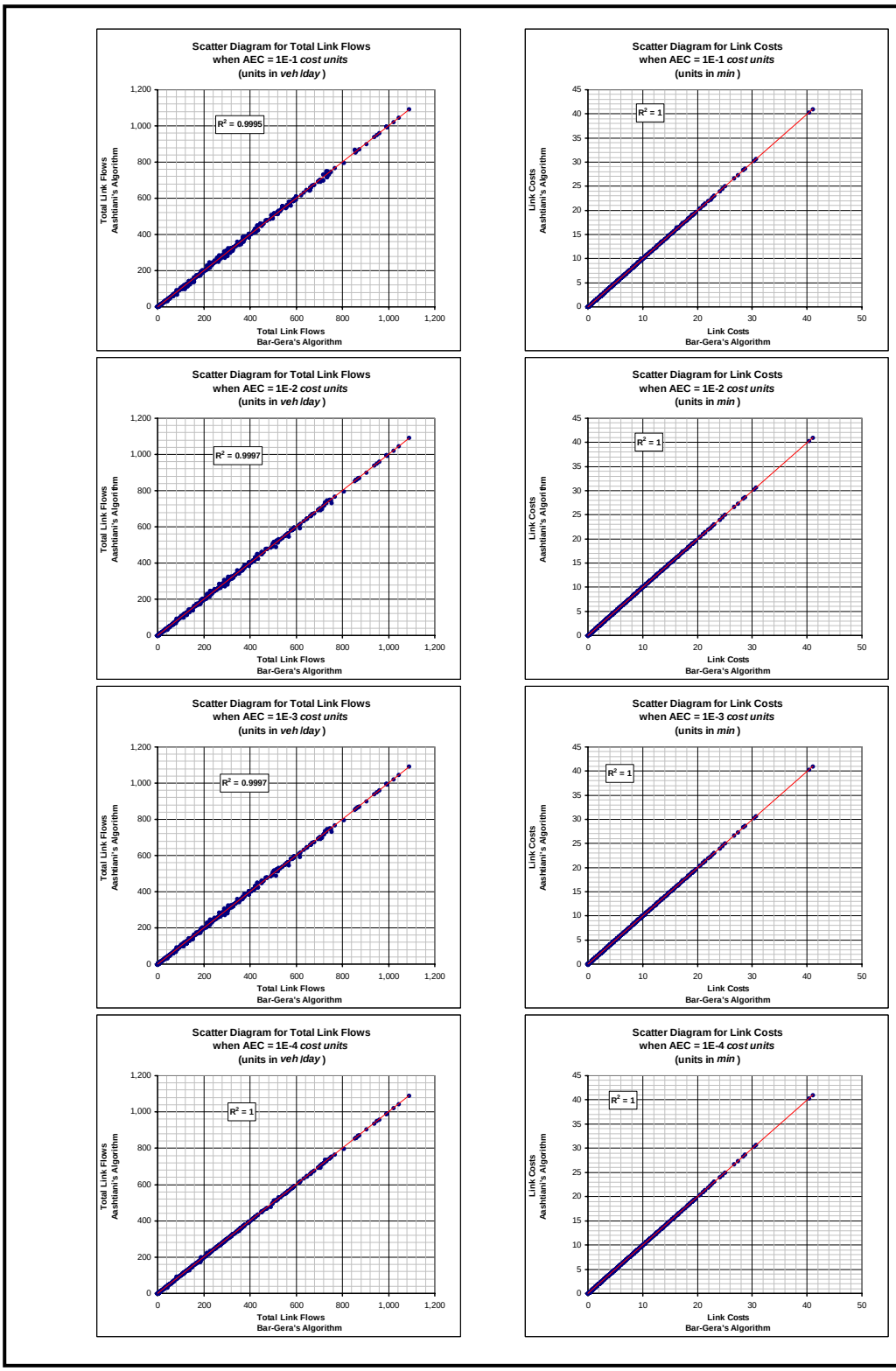


Figure 3-15c. MIT_C network: scatter diagrams and coefficients of determination that compare the total link flows and the link costs calculated by Aashtiani's and Bar-Gera's algorithms when targeting an AEC of 10^{-1}, 10^{-2}, 10^{-3} and 10^{-4} cost units.

Description of the results shown on Figure 3-15a to Figure 3-15c

Network: Mitte Center
Code: MIT_C
Number of Nodes: 397
Number of Zones: 36
Number of Arcs: 871
Number of OD Pairs: 1,260
Complexity: 1,097,460 arcs·OD pairs
Level of Complexity: 3 (10^7 to 10^8)
Total flow (total demand): 11,482 *flow units*
Units for the total link flows f_a: unknown
Units for the link costs t_a: unknown
Existence of tolls: No
Distance factor equal to zero: Yes

The scatter diagrams seem to indicate that for a targeted AEC of 10^{-1}, the solutions have enough accuracy.

Clearly in this network, Bar-Gera's algorithm is the fastest at any level of precision. And the differences are very significant: from a 60% difference to an almost 400% difference. Nonetheless, in absolute units, the differences are less than a second due to the small size of the network. The trends in the MEC corroborate the trends in the AEC.

The trends observed with this network seem somewhat different from other networks due to the clear superiority of Bar-Gera's algorithm. Still, one can observe that the speed of Bar-Gera's algorithm increases at every subsequent iteration.

Figure 3-15a shows that Bar-Gera's algorithm spent less time computing its initial solution. Nevertheless, this difference in time is small compare to the total computational time spent by both algorithms. Like in all the networks except BERL_C, the value of T in Aashtiani's algorithm starts by being smaller but Bar-Gera's algorithm quickly reaches such small values.

This is the first network where the computational time does not exceed one second. In consequence, calculating a solution **f.** in this network required more carefulness: parallel operations in the computer had the potential to affect the computational times.

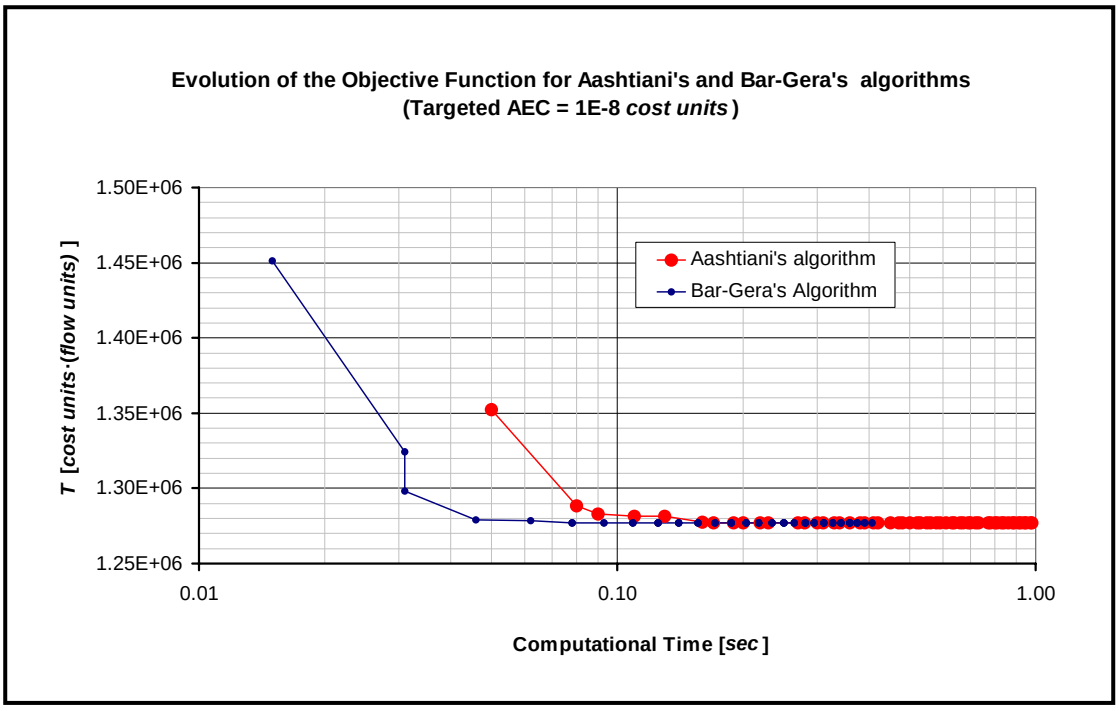


Figure 3-16a. PR_C network: Evolution of the objective function value for Aashtiani's and Bar-Gera's algorithms.

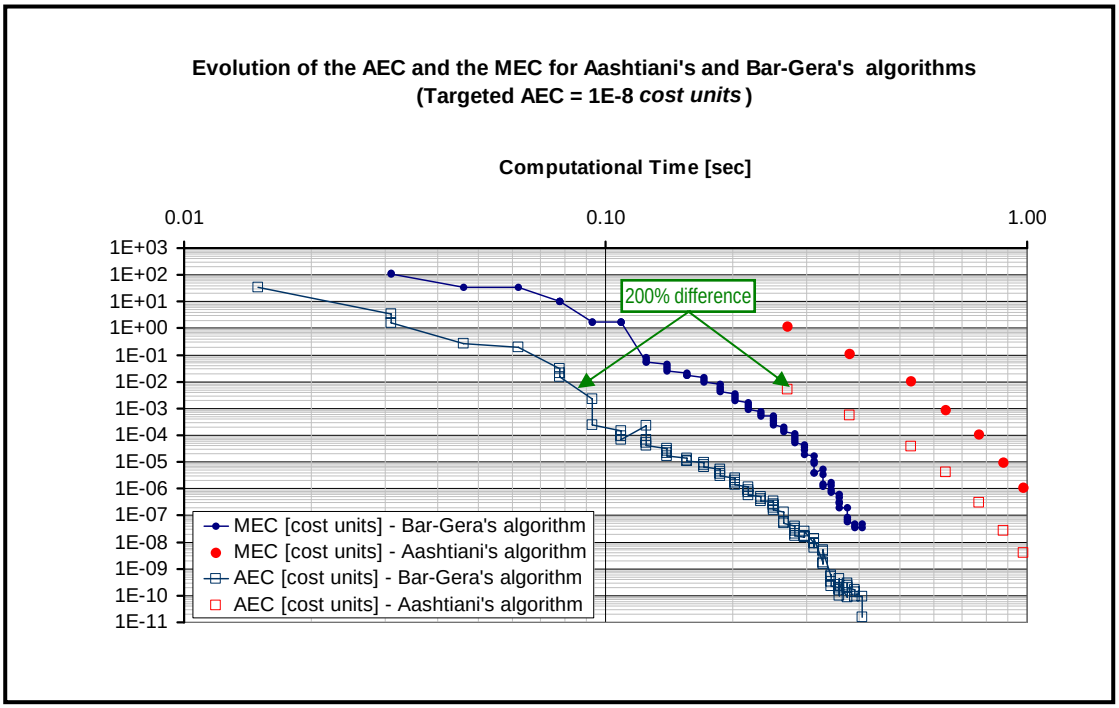


Figure 3-16b. PR_C network: Evolution of the average excess cost and the maximum excess cost for Aashtiani's and Bar-Gera's algorithms (values in green indicate maximum differences).

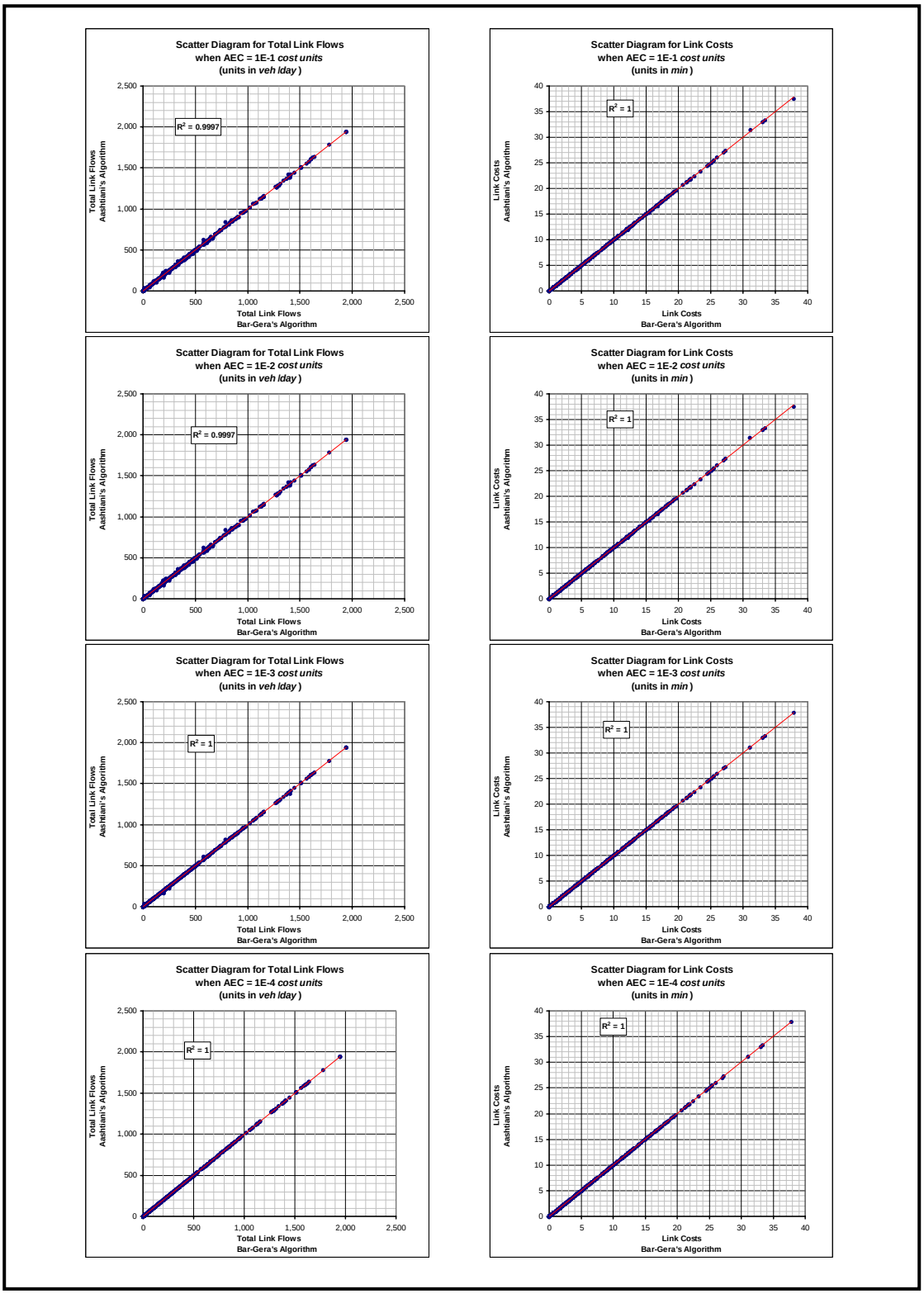


Figure 3-16c. PR_C network: scatter diagrams and coefficients of determination that compare the total link flows and the link costs calculated by Aashtiani's and Bar-Gera's algorithms when targeting an AEC of 10^{-1}, 10^{-2}, 10^{-3} and 10^{-4} cost units.

Description of the results shown on Figure 3-16a to Figure 3-16c

Network: Prenzlauer Berg Center
Code: PR_C
Number of Nodes: 749
Number of Zones: 352
Number of Arcs: 749
Number of OD Pairs: 1,406
Complexity: 1,053,094 arcs·OD pairs
Level of Complexity: 3 (10^6 to 10^7)
Total flow (total demand): 16,659 *flow units*
Units for the total link flows f_a: unknown
Units for the link costs t_a: unknown
Existence of tolls: No
Distance factor equal to zero: Yes

The results observed in this network are very similar to MIT_C. The scatter diagrams and the trends are similar to those presented by the MIT_C network. In general, Bar-Gera's algorithm outperforms Aashtiani's algorithm by a 200% difference that seems to be constant at every value in the AEC. These differences are significant but because the network is small, the differences are always less than one second.

Like with MIT_C, *Figure 3-16a* shows that the value of *T* in Aashtiani's algorithm starts by being smaller but Bar-Gera's algorithm quickly catches up. Also, like with MIT_C, Bar-Gera's algorithm spent less time in calculating the initial solution. Nonetheless, this difference in time is much less than the differences seen on *Figure 3-16b*.

The computational time does not exceed one second. As with MIT_C, measuring the computational times presented challenges.

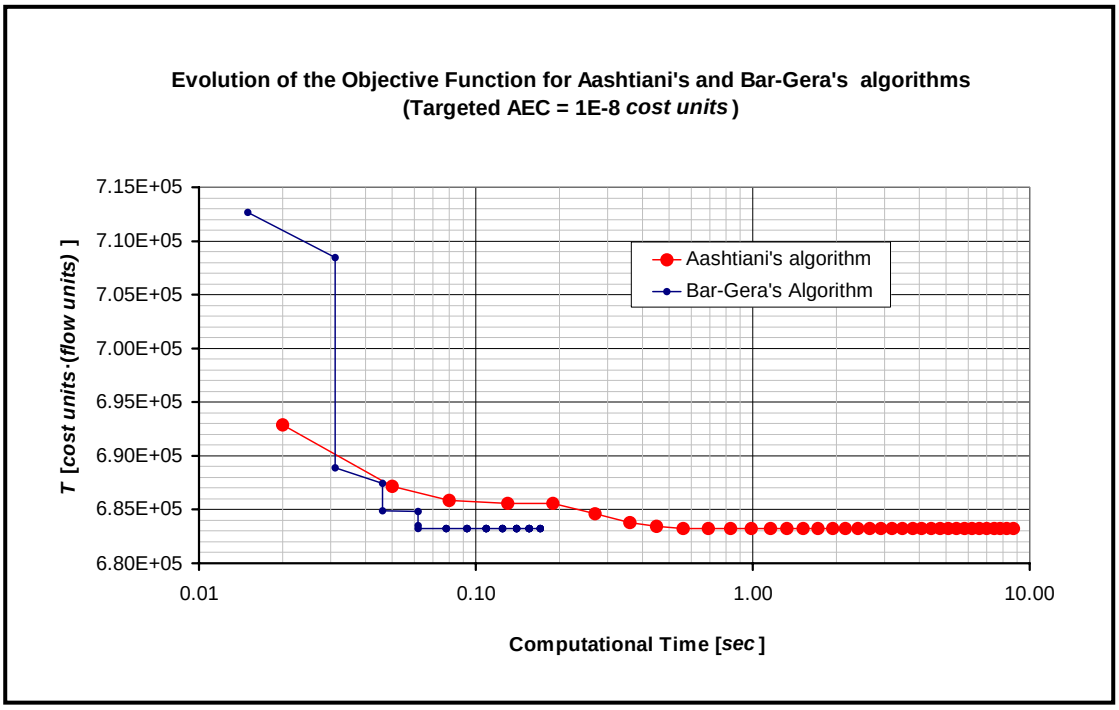


Figure 3-17a. TIEG_C network: Evolution of the objective function value for Aashtiani's and Bar-Gera's algorithms.

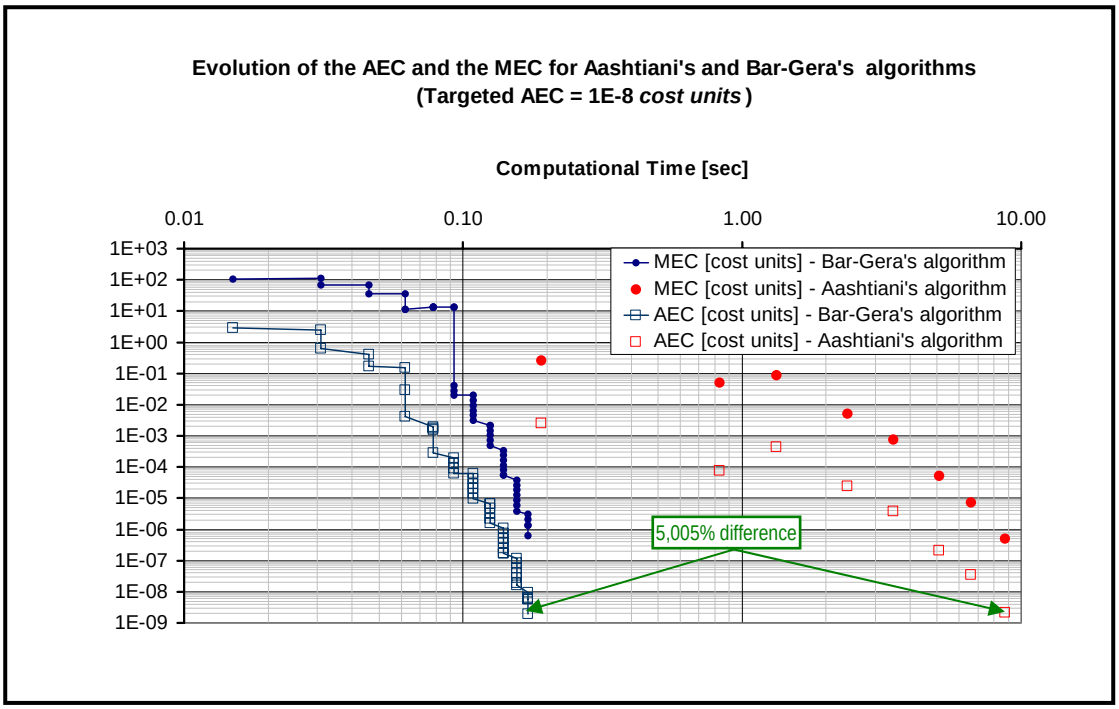


Figure 3-17b. TIEG_C network: Evolution of the average excess cost and the maximum excess cost for Aashtiani's and Bar-Gera's algorithms (values in green indicate maximum differences).

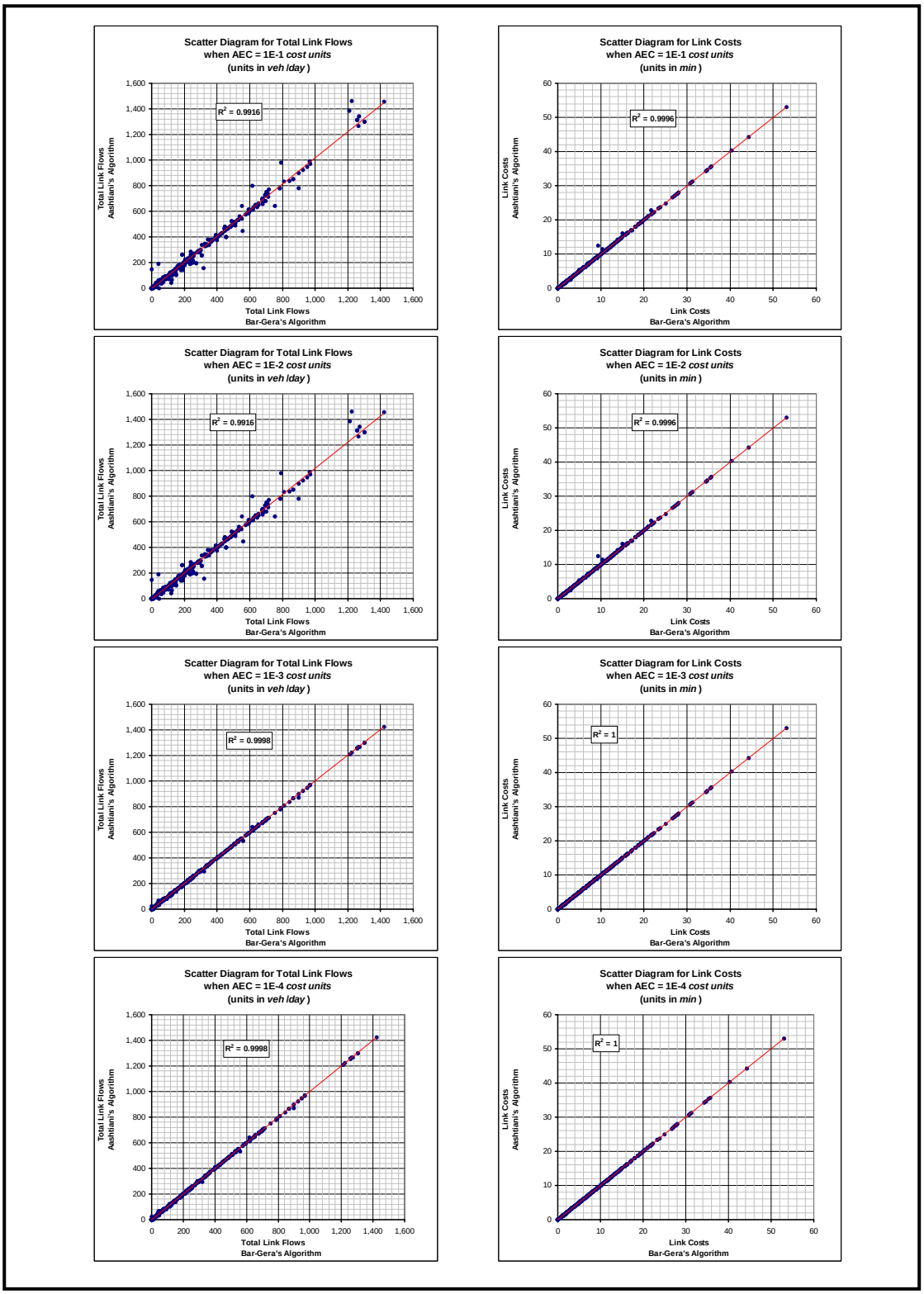


Figure 3-17c. TIEG_C network: scatter diagrams and coefficients of determination that compare the total link flows and the link costs calculated by Aashtiani's and Bar-Gera's algorithms when targeting an AEC of 10^{-1}, 10^{-2}, 10^{-3} and 10^{-4} cost units.

Description of the results shown on Figure 3-17a to Figure 3-17c

Network: Tiegarten Center
Code: TIEG_C
Number of Nodes: 359
Number of Zones: 26
Number of Arcs: 766
Number of OD Pairs: 644
Complexity: 493,304 arcs·OD pairs
Level of Complexity: 2 (10^6 to 10^7)
Total flow (total demand): 10,754 *flow units*
Units for the total link flows f_a: unknown
Units for the link costs t_a: unknown
Existence of tolls: No
Distance factor equal to zero: Yes

This network falls in the same category of MIT_C and PR_C. In general, Bar-Gera's algorithm outperforms Aashtiani's algorithm by a 5,000% difference.

Like with MIT_C and PR_C, the value of T in Aashtiani's algorithm starts by being smaller but Bar-Gera's algorithm quickly catches up. Also, like with MIT_C and PR_C, Bar-Gera's algorithm spent less time in calculating the initial solution but this difference is much smaller than the total computational times observed in *Figure 3-17b*.

The computational time does not exceed one second. As with MIT_C and PR_C, measuring the computational times presented challenges.

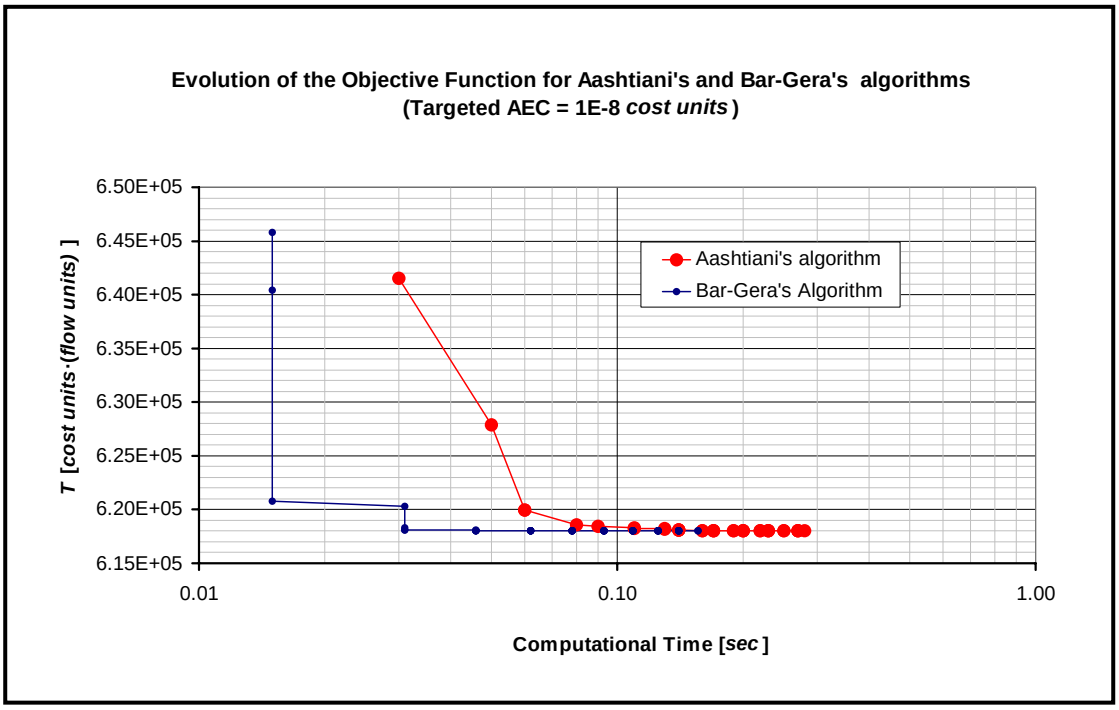


Figure 3-18a. FR_C network: Evolution of the objective function value for Aashtiani's and Bar-Gera's algorithms.

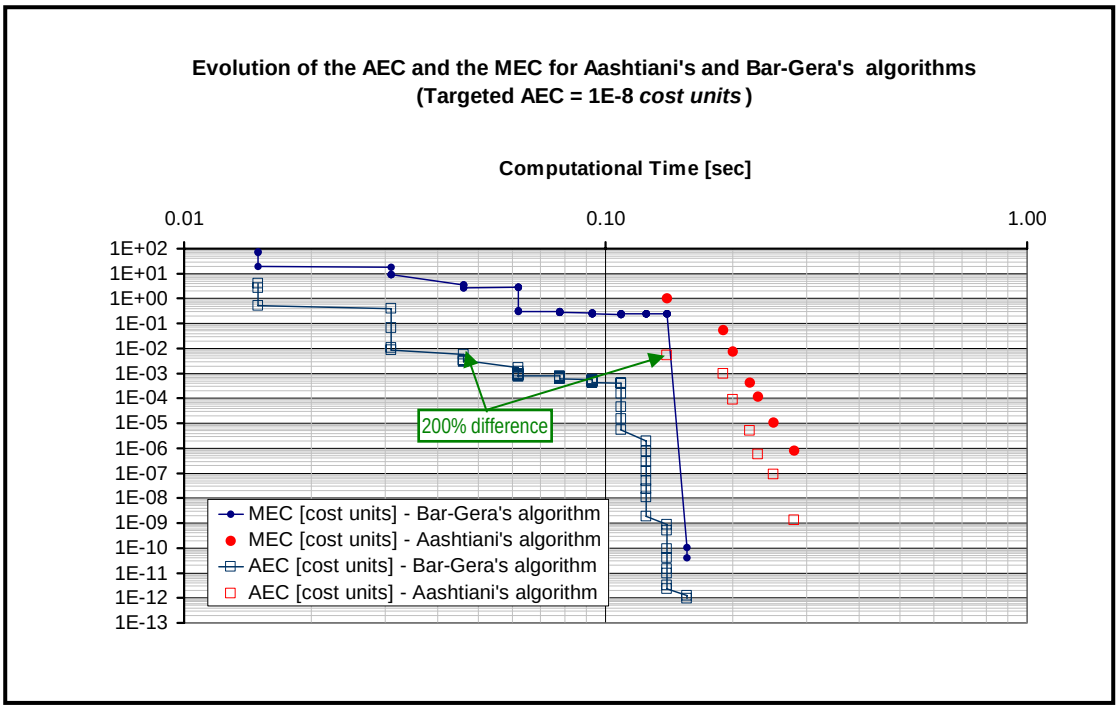


Figure 3-18b. FR_C network: Evolution of the average excess cost and the maximum excess cost for Aashtiani's and Bar-Gera's algorithms (values in green indicate maximum differences).

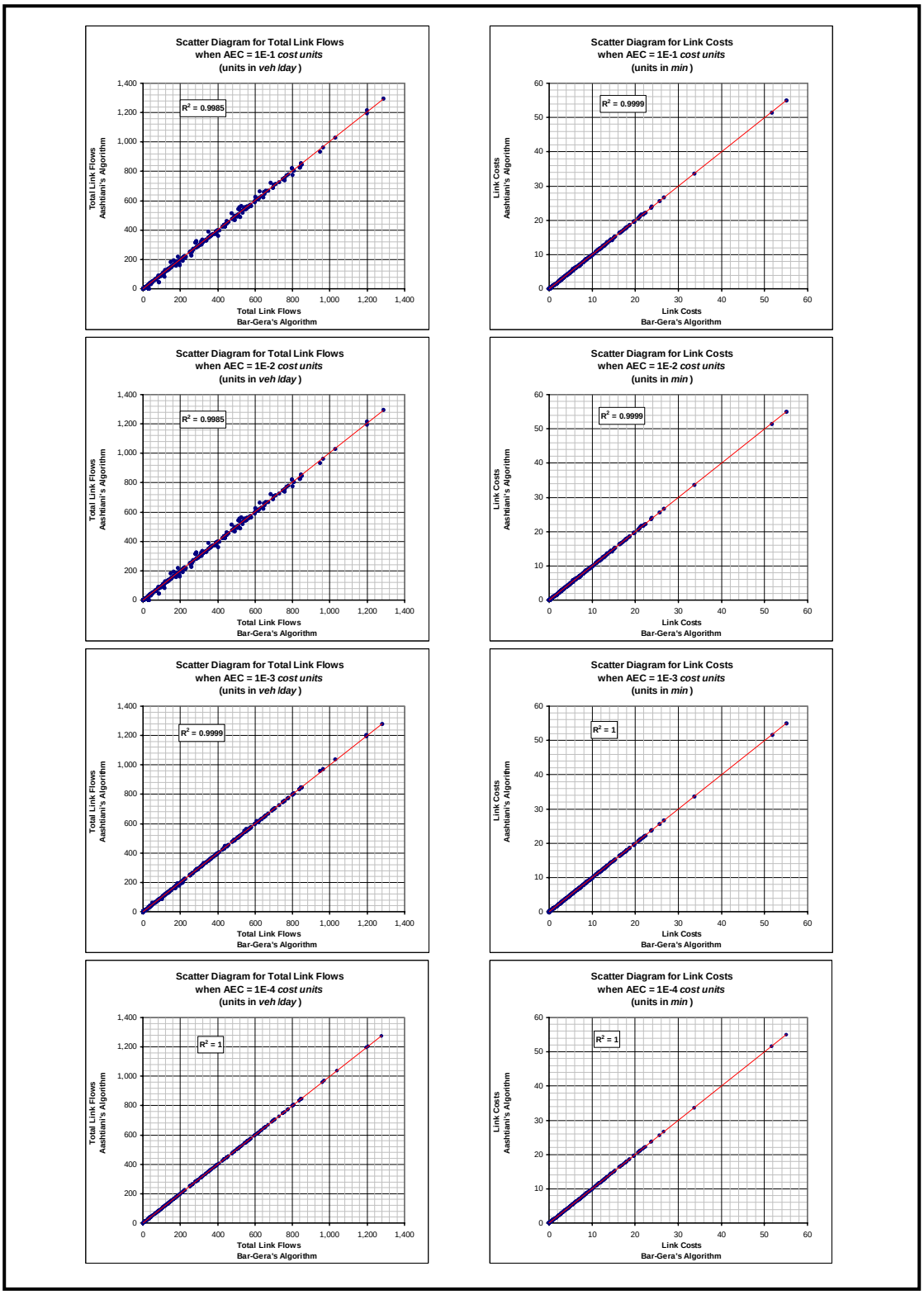


Figure 3-18c. FR_C network: scatter diagrams and coefficients of determination that compare the total link flows and the link costs calculated by Aashtiani's and Bar-Gera's algorithms when targeting an AEC of 10^{-1}, 10^{-2}, 10^{-3} and 10^{-4} cost units.

Description of the results shown on Figure 3-18a to Figure 3-18c

Network: Friedricshain Center
Code: FR_C
Number of Nodes: 224
Number of Zones: 23
Number of Arcs: 523
Number of OD Pairs: 506
Complexity: 264,638 arcs·OD pairs
Level of Complexity: 2 (10^5 to 10^6)
Total flow (total demand): 11,205 *flow units*
Units for the total link flows f_a: unknown
Units for the link costs t_a: unknown
Existence of tolls: No
Distance factor equal to zero: Yes

This network falls in the same category of the previous three networks. Bar-Gera's algorithm outperforms Aashtiani's algorithm by a difference of at least 150% always.

As shown on *Figure 3-18a*, the trends of the objective function are very similar to those shown in the previous three networks.

The computational time does not exceed one second. As with the previous two networks, measuring the computational times presented challenges.

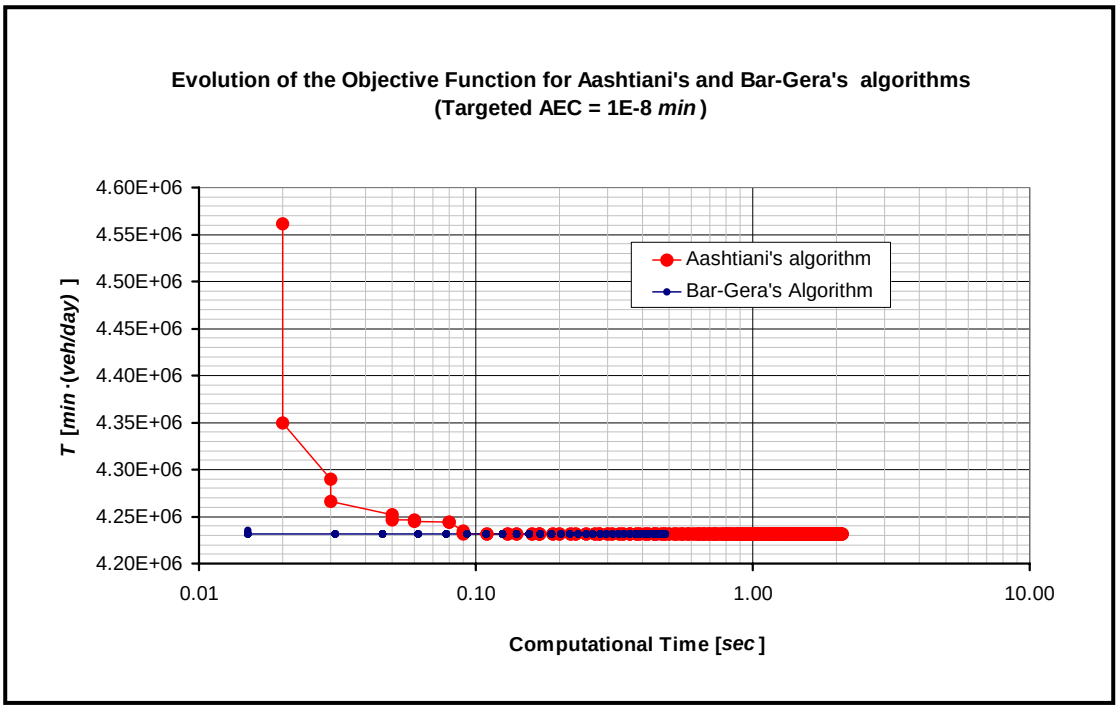


Figure 3-19a. SIOUX network: Evolution of the objective function value for Aashtiani's and Bar-Gera's algorithms.

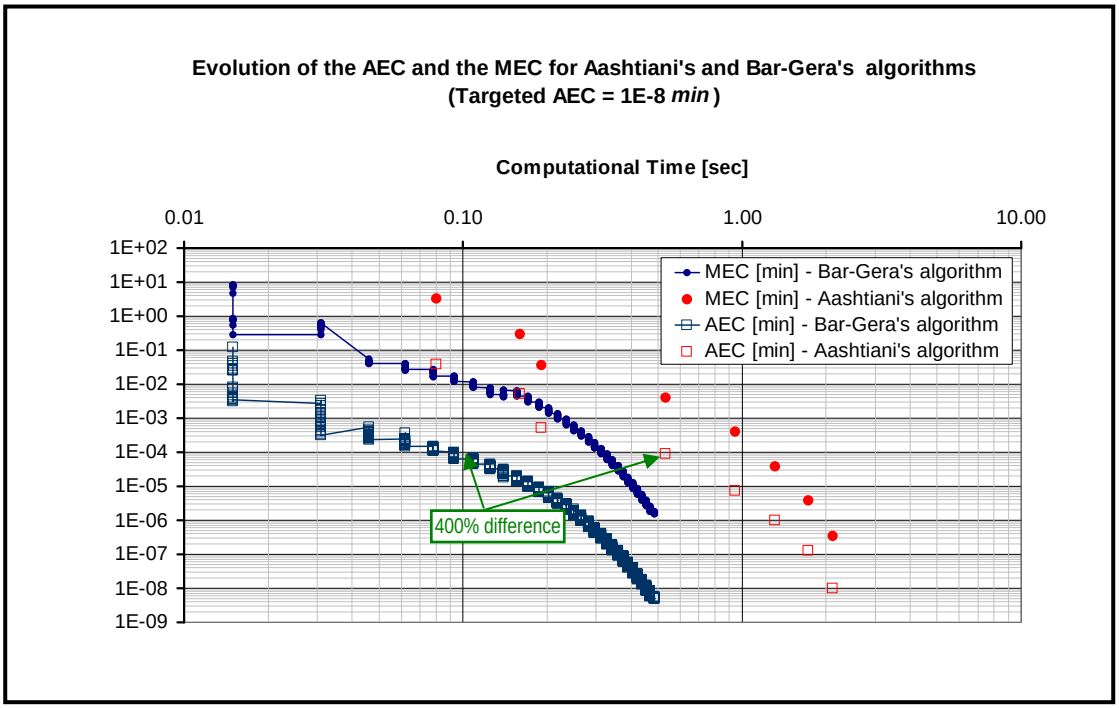


Figure 3-19b. SIOUX network: Evolution of the average excess cost and the maximum excess cost for Aashtiani's and Bar-Gera's algorithms (values in green indicate maximum differences).

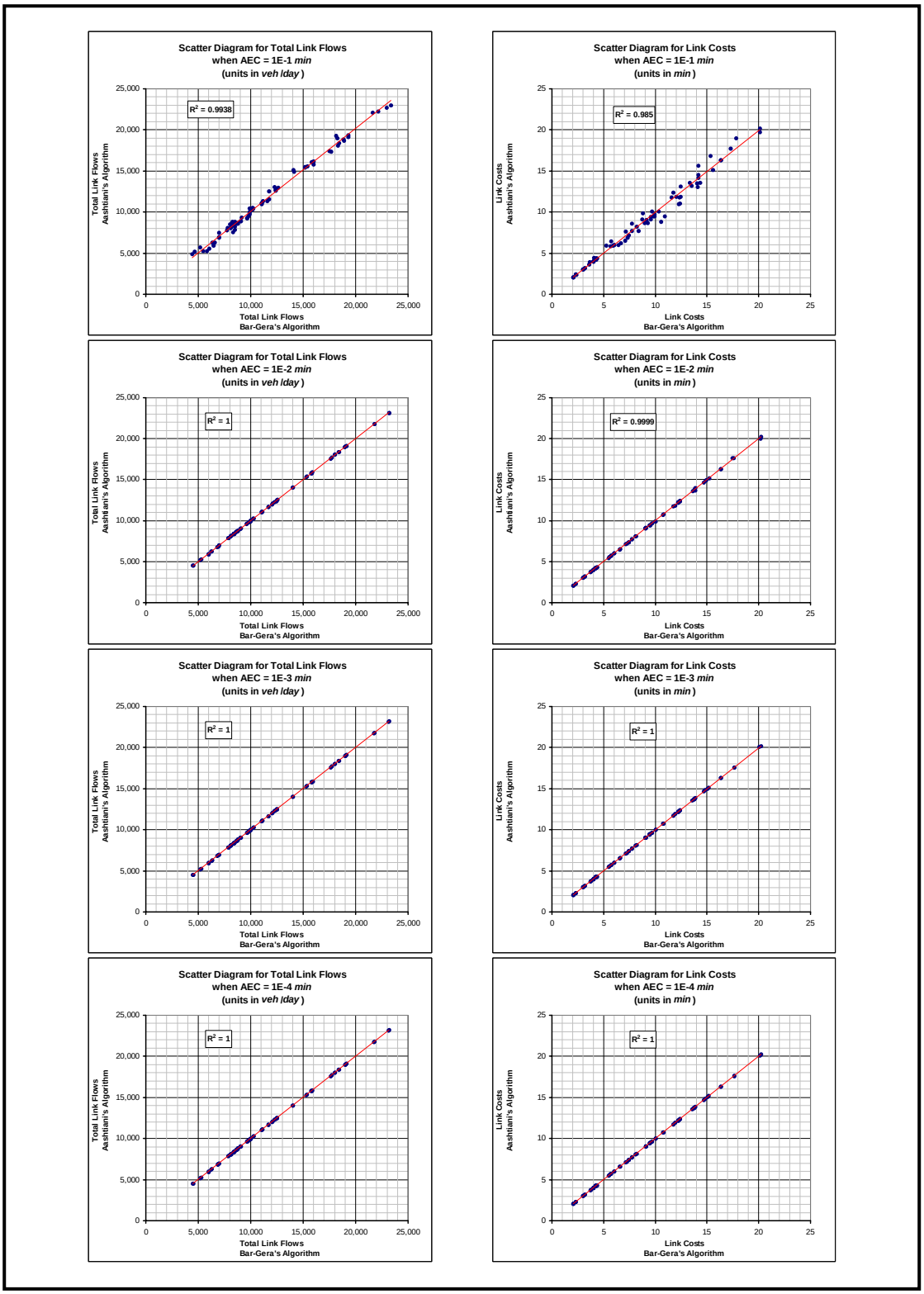


Figure 3-19c. SIOUX network: scatter diagrams and coefficients of determination that compare the total link flows and the link costs calculated by Aashtiani's and Bar-Gera's algorithms when targeting an AEC of 10^{-1}, 10^{-2}, 10^{-3} and 10^{-4} minutes.

Description of the results shown on Figure 3-19a to Figure 3-19c

Network: Sioux-Falls
Code: SIOUX
Number of Nodes: 24
Number of Zones: 24
Number of Arcs: 76
Number of OD Pairs: 552
Complexity: 41,952 arcs·OD pairs
Level of Complexity: 1 (10^4 to 10^5)
Total flow (total demand): 360,600 *flow units*
Units for the total link flows f_a: vehicles/day
Units for the link costs t_a: minutes
Existence of tolls: No
Distance factor equal to zero: Yes

The results shown on *Figure 3-19b* present the same trend shown in the past four networks: an outperformance in speed of Bar-Gera's algorithm at every level of accuracy. The only difference with those networks is that on *Figure 3-19a*, Bar-Gera's initial solution has a much smaller value of T than Aashtiani's. Like with the previous networks, Bar-Gera's algorithm is quicker in computing the initial solution but this difference is less than the difference that Bar-Gera's algorithm makes in computing the final solutions (that is, for an AEC of 10^{-3} minutes or less). And finally, like in the previous networks, the time for calculating acceptable solutions never exceeds one second in either of the two algorithms.

In terms of memory requirements, *Table 3-3* presents the maximum amount of memory used by each of the algorithms. These values, corresponding to a targeted AEC of 10^{-7} (whether in minutes or other cost units), allow to easily determine that Bar-Gera's algorithm requires much less memory.

Level of Complexity		Network	Memory required in MB to Converge to a Solution for a targeted AEC of 10^{-7}	
			Aashtiani's Method	Bar-Gera's Method
1	10^4 to 10^5	Sioux-Falls	26.44	0.01
2	10^5 to 10^6	Friedrichshain Center	40.85	0.05
		Tiegarten Center	60.62	0.07
3	10^6 to 10^7	PrenzlauerBerg Center	140.89	0.08
		Mitte Center	122.92	0.11
		Anaheim	144.74	0.20
4	10^7 to 10^8	Winnipeg	642.32	2.23
		Barcelona	1,090.06	1.97
		Mitte, PrenzlauerBerg, and Friedrichshain	1,200.98	0.76
5	10^8 to 10^9	Chicago Sketch	8,454.77	6.69
6	10^9 to 10^{10}	Berlin Center	7,299.33	93.83
7	10^{10} to 10^{11}	Philadelphia	363,540.88	344.40
8	10^{11} to 10^{12}	Chicago Regional	591,268.58	111.50

Table 3-3. Memory requirements for both algorithms when targeting an AEC of 10^{-7} (If the network is CHIC_R, PHILAD, CHIC_S, ANAH or SIOUX, the units of the AEC are in minutes. Otherwise, the units are not known.).

Analysis

To begin with, we need to verify whether the classification made to the networks on *Figure 3-1* was adequate. This classification uses the definition of complexity as recommended in previous literature (Jahn et al. 2005; Holmberg and Di Yuan 2003).

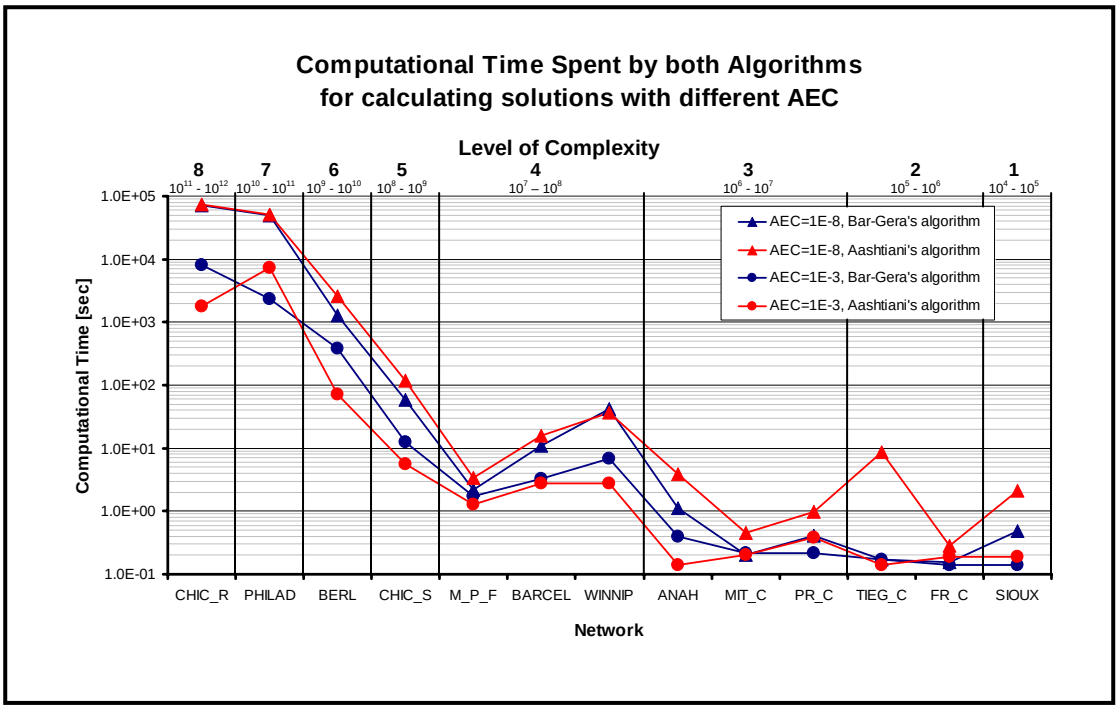


Figure 3-20. SIOUX network: Evolution of the average excess cost and the maximum excess cost for Aashtiani's and Bar-Gera's algorithms.

Figure 3-20 presents the time spent by each algorithm for an AEC of 10^{-3} and 10^{-7}. This figure shows that the level of complexity does correlate with the computational time spent. Therefore, the classification used seems to be appropriate.

Now, regarding the results shown in the previous section, a careful observation allows classifying them into four categories. The first category comprises the networks in which Aashtiani's algorithm is always the fastest regardless on how much accuracy the practitioner is seeking. The second category comprises networks in which Aashtiani's algorithm is the fastest at a high AEC and at a very low AEC. At other values in the AEC, Bar-Gera's is the fastest. The third category comprises the networks where Bar-Gera's algorithm is always the fastest except when targeting a high AEC. The fourth category comprises the networks where Bar-Gera's algorithm is the fastest independently of the AEC that the practitioner is targeting. *Table 3-4* presents the four categories.

Category	General Description	When Bar-Gera's algorithm is the slowest,		When Aashtiani's algorithm is the slowest,		Network	Level of Complexity	
		it is this % slower,	at this AEC approx[1]	it is this % slower,	at this AEC approx[1]			
1	Aashtiani's algorithm is always the fastest.	400%	1E-3 *m*	6%[2]	1E-8 *m*	CHIC_R	8	10^{11} - 10^{12}
2	Aashtiani's algorithm is always the fastest except for a medium accuracy.	10%[3]	1E-3 *m*	110%	1E-4 *m*	PHILAD	7	10^{10} - 10^{11}
		600%	1E-3 *cu*	5%	1E-5 *cu*	WINNIP	4	10^{7} - 10^{8}
3	Bar-Gera's algorithm is always the fastest except for a high (but always satisfactory) accuracy.	600%	1E-5 *cu*	900%	1E-7 *cu*	BERL_C	6	10^{9} - 10^{10}
		105%	1E-3 *m*	100%	1E-7 *m*	CHIC_S	5	10^{8} - 10^{9}
		5%	1E-4 *cu*	105%	1E-8 *cu*	M_P_F	4	10^{7} - 10^{8}
		110%	1E-3 *cu*	90%	1E-7 *cu*	BARCEL	4	10^{7} - 10^{8}
		100%	1E-4 *m*	700%	1E-7 *m*	ANAH	3	10^{6} - 10^{7}
4	Bar-Gera's algorithm is always the fastest.			390%	1E-9 *cu*	MIT_C	3	10^{6} - 10^{7}
				200%	1E-2 *cu*	PR_C	3	10^{6} - 10^{7}
				5,005%	1E-9 *cu*	TIEG_C	2	10^{5} - 10^{6}
				200%	1E-2 *cu*	FR_C	2	10^{5} - 10^{6}
				400%	1E-4 *m*	SIOUX	1	10^{4} - 10^{5}

[1] *"cu" refers to cost units which are unknown. "m" refers to minutes.*
[2] *The corresponding MEC on both algorithms suggest that Aashtiani's algorithm is faster instead of Bar-Gera's.*
[3] *Although this difference is not very large, the trend shown by the MEC suggests a larger difference.*

Table 3-4. Categories that summarize the performance of Aashtiani's and Bar-Gera's algorithms in terms of computational speed. For every category, this table shows a general description, the networks that it comprises and the extreme differences in computational time.

Four important findings are to be drawn from *Table 3-4*. First, as a network is more complex, Aashtiani's algorithm has a better performance in terms of computational speed. Second, in three out of four categories, Aashtiani's algorithm is the fastest when the accuracy is not so high. The reader should notice that *Figure 3-20* confirms this trend. The values in the AEC that appear on *Table 3-4* in which Aashtiani's algorithm is faster, although they are not very low, they do guarantee solutions with sufficient accuracy according to the scatter diagrams analyzed in the previous section.

Third, in the first three categories, although one algorithm seems to outperform the other, there are still levels of precision in which the opposite is true. For example in the CHIC_S network, overall, Bar-Gera's algorithm seems to be the fastest, but when targeting an AEC of 10^{-3} minutes, Aashtiani's algorithm could be much faster than Bar-Gera's. Therefore, although the categories reveal tendencies in the performances, the variability in the computational time is very high.

Finally, Bar-Gera's algorithm becomes faster than Aashtiani's at every subsequent iteration. The only exception was WINNIP. In other words, allocating more computational time tends to favor always Bar-Gera's algorithm and therefore, this algorithm would deliver the most accurate solution.

Since the above findings suggest that Aashtiani's algorithm is faster when the precision is not very demanding, the reader might find useful *Table 3-5*. On this table, one can observe the AEC at which Aashtiani's algorithm stops being the fastest.

Category	Network	Level of Complexity		Bar-Gera's algorithm is the slowest,		Both algorithms become equally fast, at this AEC
				by this percentage,	at this AEC	
1	CHIC_R	8	10^{11} - 10^{12}	400%	1.0 E-3 *minutes*	1.0 E-8 *minutes*
2	PHILAD	7	10^{10} - 10^{11}	10%	1.3 E-3 *minutes*	1.1 E-3 *minutes*
	WINNIP	4	10^{7} - 10^{8}	600%	1.0 E-3 *cost units*	1.7 E-5 *cost units*
3	BERL_C	6	10^{9} - 10^{10}	600%	1.2 E-5 *cost units*	1.2 E-5 *cost units*
	CHIC_S	5	10^{8} - 10^{9}	105%	1.6 E-4 *minutes*	1.3 E-4 *minutes*
	M_P_F	4	10^{7} - 10^{8}	5%	1.0 E-4 *cost units*	1.8 E-5 *cost units*
	BARCEL	4	10^{7} - 10^{8}	110%	1.0 E-3 *cost units*	1.2 E-5 *cost units*
	ANAH	3	10^{6} - 10^{7}	100%	1.1 E-4 *minutes*	1.3 E-3 *minutes*

Table 3-5. Networks for which Aashtiani's algorithm starts being faster than Bar-Gera's algorithm and then, at some AEC (as indicated on the last column), both algorithms become equally fast.

It is important to clarify some aspects concerning the evolution of the objective function *T*. One could state that Aashtiani's algorithm performs better at low levels of accuracy because in general, the initial solution that Aashtiani's algorithm generates has a lower *T*. Two arguments contradict this statement. First, as the corresponding figures suggest (*Figure 3-7a*, *Figure 3-8a* up to *Figure 3-19a*), by the second or third iteration of Bar-Gera's algorithm, its value of *T* becomes as low as the one of Aashtiani's algorithm. Second, the computational time measured during this thesis starts from the moment the algorithm starts generating the initial solution. Therefore, while Bar-Gera's initial solution has, most of the times, a higher *T*, it requires in some cases less time for its calculation. On the other hand, Aashtiani's algorithm can spend more time in calculating its initial solution.

Finally, as expected when comparing an origin-based algorithm with a route based algorithm, Bar-Gera's method requires less memory. Nevertheless, an interesting result is that Aashtiani's algorithm, although it can consider many routes for every OD pair (that is, the cardinality of the set of working paths $\boldsymbol{R}_i^w$ can be as high as $|\boldsymbol{R}_i|$), the results show that the algorithm only used four routes at the most.

CHAPTER 4: SUMMARY, CONCLUSIONS AND FURTHER RESEARCH

This thesis aimed at comparing Aashtiani's method and Bar-Gera's method for solving the static traffic assignment with fixed demand (S-TAP-F). The main focus of the comparison was to determine which method requires the least computational time when executing their corresponding algorithms. The main intention of their authors was not to create two methods that could be comparable between themselves but two methods that could solve two similar but not identical problems: each method has different assumptions about the problems to solve. For this reason, this thesis looked, in the second chapter, at their theoretical backgrounds and found out that problems, if compliant with the following assumptions, serve as a test bed for comparing both methods:

1. The TAP to be solved is static.
2. The TAP to be solved has a fixed demand.
3. The TAP to be solved is deterministic.
4. The solution does not need to be integer.
5. Every link performance function depends solely on the total link flow that passes through it and not on other total link flows.
6. Every link performance function is positive, continuous and strictly increasing.
7. The solutions are to be compared is in terms of total link flows (and not route flows).

The second chapter also looked at the parameters required by Aashtiani's algorithm and Bar-Gera's algorithm. It concluded that there are interesting parameters that regulate the ratio of iterations versus cycles (or in other words, "local search" versus "global search"). Nevertheless, Bar-Gera's software suggested a value for its parameter (that is, $m = 2$) and a value that presented satisfactory preliminary results was used for Aashtiani's algorithm (that is, $m_A = 10$).

Two additional aspects were analyzed concerning the nature of the algorithms. To begin with, Aashtiani (1979) suggested that his algorithm should use Bellman's shortest path algorithm (Bellman 1958) because it was considered the best at the time (Golden 1975). Instead, this thesis used the L-deque algorithm (Pape 1974) because according to a more recent study by Pallottino and Scuttelà (1998), it is the fastest for transportation networks. Secondly, Bar-Gera's algorithm starts with an arbitrary feasible solution. Aashtiani's algorithm starts with a different initial solution, that is, a solution in which all the total route flow for each OD pair is assigned to the path with

the least cost. Since the software used for Bar-Gera's algorithm was not open source, Aashtiani's algorithm could not be modified accurately so that it could generate the same initial solution that Bar-Gera's algorithm used.

Finally, the second chapter reviewed the data structures used by Bar-Gera and the data structures that Toobaie (1998) added to Aashtiani's algorithm.

This thesis presented on its third chapter thirteen networks (and link performance functions) that complied with the assumptions identified in the previous chapter. One of these networks, WINNIP, did not comply with the assumptions thoroughly. It contained links, other than connectors, whose performance functions were equal to zero. Nevertheless, results showed that the number of this kind of links was insufficient for affecting a meaningful comparison. The networks were classified according to the product of the number of OD pairs times the number of arcs as suggested in previous literature (Jahn et al. 2005; Holmberg and Di Yuan 2003). To every value of complexity, this thesis assigned a number (a level) from 1 to 8, where 8 would refer to the most complex network. According to this numeration, every level of complexity had at least one network (see *Figure 3-1*).

Due to (1) the absence of units in some networks, (2) the lack of a physical interpretation for the objective function T (defined in [2-2a]), and (3) the very different results obtained at every level of accuracy in the solution **f.**, the numerical comparison required crafting a careful procedure. The procedure chosen for this thesis was the following:

1. For each method, run the algorithms targeting a very low value in the AEC.

2. Plot (like with the first and second approaches) the evolution of T.

3. For various targeted AECs, plot one scatter diagram that compares the total link flows f_a and another scatter diagram that compares the the link costs t_a between the two algorithms.

4. Use the scatter diagrams to determine the maximum targeted AEC that guarantees a straight line (the reader can observe here that this verification is somewhat subjective, especially when the units are unknown. The coefficient of determination R^2 is recommended for this verification but it can be heavily influenced by outliers whose magnitudes are very large compared to the rest of points).

5. Plot (like with the second approach) the evolution of the AEC and the MEC against the computational time.

6. Look for *trends* in the evolution of the AEC and verify those trends with the MEC. Make sure that the values of the AEC are less than or equal to the maximum targeted AEC that was determined on step 4.

The main conclusions to be drawn from the results observed on this thesis are the following (most of them can be drawn from *Table 3-4* which, for convenience to the reader, appears below): (1) Aashtiani's algorithm and Bar-Gera's algorithm showed very similar performances not because their computational times were similar but because on a same network, Aashtiani's algorithm could be much faster than Bar-Gera's at certain level of accuracy and much slower at another level of accuracy. It can be affirmed from here, that the variability of the results was very high. (2) Aashtiani's algorithm seems to be the fastest algorithm when applied to the most complex networks. Clearly, it was faster than Bar-Gera's algorithm for solving the largest network. (3) In most cases, Aashtiani's algorithm was the fastest at solving the S-TAP-F at low but satisfactory levels of accuracy. The maximum accuracy at which Aashtiani's algorithm was always faster can only be stated when the units are known. Therefore, from the four largest networks whose units were known, one can conclude that Aashtiani's algorithm was faster than Bar-Gera's when the AEC was greater than or equal to 0.11 minutes (this result is drawn from *Table 3-5*). (4) On the contrary, Bar-Gera's algorithm has an increasing computational speed. As a result, at almost every lesser value in the AEC, Bar-Gera's algorithm will perform faster. Even with the largest network where Aashtiani's algorithm was the fastest, it seems that if requiring a greater (but unnecessary) precision, Bar-Gera's algorithm could become the fastest. (5) Bar-Gera's algorithm presented a clear superiority in the five least complex networks.

Category	General Description	When Bar-Gera's algorithm is the slowest,		When Aashtiani's algorithm is the slowest,		Network	Level of Complexity	
		it is this % slower,	at this AEC approx[1]	it is this % slower,	at this AEC approx[1]			
1	Aashtiani's algorithm is always the fastest.	400%	1E-3 *m*	6%[2]	1E-8 *m*	CHIC_R	8	10^{11} - 10^{12}
2	Aashtiani's algorithm is always the fastest except for a medium accuracy.	10%[3]	1E-3 *m*	110%	1E-4 *m*	PHILAD	7	10^{10} - 10^{11}
		600%	1E-3 *cu*	5%	1E-5 *cu*	WINNIP	4	10^{7} - 10^{8}
3	Bar-Gera's algorithm is always the fastest except for a high (but always satisfactory) accuracy.	600%	1E-5 *cu*	900%	1E-7 *cu*	BERL_C	6	10^{9} - 10^{10}
		105%	1E-3 *m*	100%	1E-7 *m*	CHIC_S	5	10^{8} - 10^{9}
		5%	1E-4 *cu*	105%	1E-8 *cu*	M_P_F	4	10^{7} - 10^{8}
		110%	1E-3 *cu*	90%	1E-7 *cu*	BARCEL	4	10^{7} - 10^{8}
		100%	1E-4 *m*	700%	1E-7 *m*	ANAH	3	10^{6} - 10^{7}
4	Bar-Gera's algorithm is always the fastest.			390%	1E-9 *cu*	MIT_C	3	10^{6} - 10^{7}
				200%	1E-2 *cu*	PR_C	3	10^{6} - 10^{7}
				5,005%	1E-9 *cu*	TIEG_C	2	10^{5} - 10^{6}
				200%	1E-2 *cu*	FR_C	2	10^{5} - 10^{6}
				400%	1E-4 *m*	SIOUX	1	10^{4} - 10^{5}

[1] *"cu" refers to cost units which are unknown. "m" refers to minutes.*
[2] *The corresponding MEC on both algorithms suggest that Aashtiani's algorithm is faster instead of Bar-Gera's.*
[3] *Although this difference is not very large, the trend shown by the MEC suggests a larger difference.*

Table 3-4 (repeated). Categories that summarize the performance of Aashtiani's and Bar-Gera's algorithms in terms of computational speed. For every category, this table shows a general description, the networks that it comprises and the extreme differences in computational time.

One lesson to be drawn from this study is that the utilization of non-open source software carries many disadvantages. The large number of trials and actual computations required software whose stopping criteria could not be tailored. Bar-Gera's software only offers the AEC as the stopping criterion. A second lesson to be drawn from these results is that, given the speed of current desktop and laptop computers, networks with a complexity lower than 10^{-7} have become non-recommendable for comparing convergence rates among S-TAP-F solution algorithms. Their computational times do not exceed one second.

Further study could focus on using larger networks for comparing both networks. Also, further study could analyze the impact of using different shortest path algorithms for Aashtiani's algorithm. The results obtained for this thesis are the product of using the L-deque shortest-path algorithm (Pape 1974) in Aashtiani's algorithm. This choice follows Pallottino and Scutella's (1998) recommendation which states that for networks typical of transportation models, the L-deque shortest-path algorithm is the best choice. Therefore, there is still the possibility that Aashtiani's algorithm could show a better performance if using a different shortest path algorithm.

Comparisons in the past between algorithms that solve the S-TAP have used dimensionless metrics as stopping criteria such as the *relative gap* (Boyce, Ralevic-Dekic, and Bar-Gera 2004). This dimensionless metrics offer advantages but, when comparing algorithms with very mixed results, the practitioner might find insufficient to conclude that an algorithm is faster on the other just by comparing the time in reaching an ideal solution.

Previous to this study, we already knew that (1) Aashtiani's method offered the possibility of solving different kinds of TAPs, and that (2) Bar-Gera's method, although just designed for the S-TAP-F, required less computer memory (this result was confirmed by *Table 3-3*). Now, this thesis shows that no algorithm is thoroughly faster than the other. But results do suggest that (1) as networks become more complex, Aashtiani's algorithm becomes faster than Bar-Gera's for low levels of accuracy, and (2) as networks become even more complex, Aashtiani's algorithm can be considered the fastest overall.

APPENDIX: THE TWO METHODS IN DETAIL

The following appendix describes in detail the two methods that this thesis compares. The reader can regard it as a complement to *Chapter 2*. Nevertheless, this appendix is completely self-contained. Since Bar-Gera's (1999) and Aashtiani's (1979) original works are extensive, since Aashtiani conceived his method for several types of TAPs (not just the S-TAP-F), and since both references use different notations, this appendix becomes necessary. This appendix also allows *Chapter 2* to conclude whether both algorithms are comparable for the networks used here.

This appendix comprises three sections. The first one unifies concepts used by both methods. The second section explains and compares the mathematical formulations. The third section presents the algorithms.

Definitions and Notation

The heavy use of summations and the different terms used by Bar-Gera and Aashtinai renders this section important.

Regarding the geometry of a network, this thesis uses a specific notation for *nodes*, *arcs*, *tails* and *heads*. Letter *n* will refer to any kind of node, that is, any intersection, toll booth, destination point, origin point, and so on. Letter *a* will represent any *arc* (or *link*), that is, any street segment connecting two nodes (As explained at the end of this section, there is a special kind of arcs denominated *connectors*.) The notation a_t will refer to the tail (the beginning node) of an arc and the notation a_h will refer to the head (the ending node) of an arc. Thus, this thesis considers all arcs as directed, that is, there is always one beginning node and one ending node. This thesis will also use a binary notation for representing arcs, that is, $[a_t, a_h]$. As defined in [A-1], $\boldsymbol{N}$ will represent the set of all nodes and $\boldsymbol{A}$ will represent the set of all arcs (notice the use of apostrophes instead of numeric subindices).

$$\boldsymbol{N} = \{n, n', n'', \ldots\} \qquad \text{[A-1a]}$$

$$\boldsymbol{A} = \{a, a', a'', \ldots\} \qquad \text{[A-1b]}$$

The OD matrix is one of the inputs needed for solving the TAP. This matrix presents the total *trips* or (the *demand*) that users need to make. Bar-Gera's and Aashtiani's methods make use of the following concepts that take into account that trip information: *zones*, *origin nodes*, *destination nodes*, *OD pairs,* and *routes*. A *zone* refers to any node where a trip starts or a trip ends. Therefore, a zone can be an *origin node p* or a *destination node q*. $\boldsymbol{N}_o$ will represent the set of origin nodes and $\boldsymbol{N}_d$ will represent the set of destination nodes. An *OD pair* is simply a pair of nodes composed by an origin *p* and a destination *q*. For this thesis, an OD pair will always assume that some users will indeed need to travel from that OD pair's origin to that OD pair's destination (in other words, this thesis does not take into account OD pairs with zero demand). Bar-Gera uses a binary notation for referring to an OD pair while Aashtiani

finds more convenience in using a unary notation because his algorithm relies heavily on the manipulation of OD pairs. Therefore, this thesis will use both notations, that is, a binary notation (p, q) and a unary notation i. Letter $\boldsymbol{I}$ will represent the set of all OD pairs given in the OD matrix. $\boldsymbol{N}_d(p)$ will represent the set of destination nodes q whose OD pairs have the same origin p. For a *route* (or *path*), that is, any set of non-repeated adjacent nodes, this thesis will use the notation [n, n', n'', …]. Letter $\boldsymbol{R}$ will represent the set of all possible routes. Nevertheless, most of the times, this thesis will refer to *routes that connect OD pairs* as defined in the given OD matrix. The notation $\boldsymbol{R}_{(p,\,q)}$ (or $\boldsymbol{R}_i$) will represent the subset of routes that connect an origin node p with a destination node q (or an OD pair i). This thesis will use three different notations for referring to routes that connect OD pairs: (1) [p, n, n', …, q], (2) $r_{(p,\,q)}$, or (3) r_i. Bar-Gera prefers using the first and second notations while Aashtiani prefers the third. The following expressions show the definitions and the relationships existing among the above sets.

$$\boldsymbol{N}_o = \{p,\, p',\, p'', \ldots\} \qquad \text{[A-2a]}$$

$$\boldsymbol{N}_o \subseteq \boldsymbol{N} \qquad \text{[A-2b]}$$

$$\boldsymbol{N}_d = \{q,\, q',\, q'', \ldots\} \qquad \text{[A-2c]}$$

$$\boldsymbol{N}_d(p) \subseteq \boldsymbol{N}_d \subseteq \boldsymbol{N} \qquad \text{[A-2d]}$$

$$\boldsymbol{I} = \{i,\, i',\, i'', \ldots\} = \{(p,\, q),\, (p,\, q'), \ldots, (p',\, q),\, (p',\, q'), \ldots\} \qquad \text{[A-2e]}$$

$$\boldsymbol{I} \subseteq \boldsymbol{N}_o \times \boldsymbol{N}_d \qquad \text{[A-2f]}$$

$$\boldsymbol{R}_i = \{r_i,\, r_i',\, r_i'', \ldots\} = \boldsymbol{R}_{(p,\,q)} = \{r_{(p,\,q)},\, r_{(p,\,q)}',\, r_{(p,\,q)}'', \ldots\} \qquad \text{[A-2g]}$$

$$\boldsymbol{R} = \{r_1,\, r_1',\, r_1'', \ldots,\, r_2,\, r_2',\, r_2'', \ldots\, r_3,\, r_3',\, r_3'', \ldots\} \qquad \text{[A-2h]}$$

$$\boldsymbol{R} = \{r_{(p,\,q)},\, r_{(p,\,q)}', \ldots, r_{(p,\,q')},\, r_{(p,\,q')}', \ldots,\, r_{(p',\,q)},\, r_{(p',\,q)}', \ldots, r_{(p',\,q')},\, r_{(p',\,q')}', \ldots\} \qquad \text{[A-2i]}$$

$$\boldsymbol{R} = \bigcup_{\forall i \in \boldsymbol{I}} \boldsymbol{R}_i = \bigcup_{\forall (p,\,q) \in \boldsymbol{I}} \boldsymbol{R}_{(p,\,q)} \qquad \text{[A-2j]}$$

d_i (or $d_{(p,\,q)}$) will denote the demand corresponding to an OD pair i (or the demand of users that start their trip on origin node p and end on destination node q). To indicate that a link a is part of a route $r_{(p,\,q)}$, this thesis will follow Bar-Gera's suggestion by using the following notation: $a \subseteq r_{(p,\,q)}$.

Taking into account the concept of flow, comparing Bar-Gera and Aashtiani's methods requires specifying a notation for the following terms: *route flow*, *origin-based link flow* and *total link flow*. The route flow h_{r_i} is the flow that passes along a

route r_i corresponding to an OD pair *i*. Therefore, a feasible solution to the TAP should guarantee that the sum of all route flows h_{r_i}, where $r_i \in \boldsymbol{R}_i$, is equal to the demand d_i. This thesis has chosen the notation h_{r_i} instead of h_{rpq} (as suggested by Bar-Gera 1999) or h_r^{pq} (as suggested by Sheffi 1985), because the latter two notations do not consider giving a separate numeration to routes belonging to different OD pairs. This thesis will also use, in a less frequent manner, the notation $h_{r_{(p,q)}}$ to indicate the origin and destination nodes, and the notation $h_{[n,n',n'',\ldots]}$ to indicate the nodes of a specific route. The notations h_{r_i} and $h_{r_{(p,q)}}$ also help better specify the summations that involve route flows. The notation **h** represents the vector containing all route flows. In Aashtiani's method, it is helpful to assume that the route flows in the vector **h** are grouped by OD pairs as shown in *[9a]*:

$$\mathbf{h} = \left[h_{1_1}\ h_{2_1} \ldots h_{|\boldsymbol{R}_1|_1}\ \ h_{1_2}\ h_{2_2} \ldots h_{|\boldsymbol{R}_2|_2}\ \ldots h_{1_{|I|}}\ h_{2_{|I|}} \ldots h_{|\boldsymbol{R}_{|I|}|_{|I|}} \right] \qquad \text{[A-3a]}$$

In Aashtiani's method, the concept of *route flow sub-vector* is also useful as defined below. This definition will allow decomposing the *route flow vector* [A-3a] into route flow sub-vectors.

$$\mathbf{h}_i = \left\lfloor h_{1_i} \quad h_{2_i} \quad \ldots \quad h_{|\boldsymbol{R}_i|_i} \right\rfloor \qquad \text{[A-3b]}$$

The mathematical definitions of the *origin-based link flow* and *total link flow* are the following:

$$f_{p,a}(\mathbf{h}) = \sum_{\forall r_{(p,q)} \in \boldsymbol{R}_{(p,q)},\ \forall q \in \boldsymbol{N}_d(p):\ a \subseteq r_{(p,q)}} h_{r_{(p,q)}} \qquad \text{[A-4]}$$

$$f_{\bullet a}(\mathbf{h}) = \sum_{\forall r_i \in \boldsymbol{R}_i, \forall i \in \boldsymbol{I}:\ a \in r_i} h_{r_i} = \sum_{\forall p \in \boldsymbol{N}_q} f_{p,a}(\mathbf{h}) \qquad \text{[A-5a]}$$

Sometimes, it is useful to use the *arc-route incidence value* $\delta_{a\,r_i}$, a binary variable which takes the value of one when a link belongs to a route and a value of zero otherwise. Using this variable, [A-5a] is equivalent to [A-5b].

$$f_{\bullet a}(\mathbf{h}) = \sum_{\forall r_i \in \boldsymbol{R}_i, \forall i \in \boldsymbol{I}} h_{r_i} \cdot \delta_{a\,r_i} \qquad \text{[A-5b]}$$

The above variables allow the definition of the following vectors and matrices: the *origin-based link flow vector for origin p* or $\mathbf{f}_p$, the *origin-based link flow matrix* or $\mathbf{f}$, and the *total link flow vector* or $\mathbf{f}_\bullet$:

$$\mathbf{f}_p(\mathbf{h}) = \left\lfloor f_{p,1}(\mathbf{h}) \quad f_{p,2}(\mathbf{h}) \quad \ldots \quad f_{p,|\boldsymbol{A}|}(\mathbf{h}) \right\rfloor \qquad \text{[A-5c]}$$

$$\mathbf{f}(\mathbf{h}) = \begin{bmatrix} f_{1,1}(\mathbf{h}) & f_{1,2}(\mathbf{h}) & \dots & f_{1,|\boldsymbol{A}|}(\mathbf{h}) \\ f_{2,1}(\mathbf{h}) & f_{2,2}(\mathbf{h}) & \dots & f_{2,|\boldsymbol{A}|}(\mathbf{h}) \\ \dots & \dots & \dots & \dots \\ f_{|\boldsymbol{N}_o|,1}(\mathbf{h}) & f_{|\boldsymbol{N}_o|,2}(\mathbf{h}) & \dots & f_{\boldsymbol{N}_o,|\boldsymbol{A}|}(\mathbf{h}) \end{bmatrix} \quad \text{[A-5d]}$$

$$\mathbf{f}_{\bullet}(\mathbf{h}) = \begin{bmatrix} f_{\bullet 1}(\mathbf{h}) & f_{\bullet 2}(\mathbf{h}) & \dots & f_{\bullet|\boldsymbol{A}|}(\mathbf{h}) \end{bmatrix} \quad \text{[A-5e]}$$

Taking into account the concept of cost, both methods refer to the following concepts: *link cost*, *route cost* and *minimum route cost*. Usually, as in this thesis, the units used for costs are units of time. The link cost, denoted as t_a, is the cost or time that it takes for a user to pass through arc *a* depending on the total link flow $f_{\bullet a}$ or on the whole vector of total link flows $\mathbf{f}_{\bullet}$. The link cost is in other words, the numeric result of the performance function. A typical performance function, where the t_a depends only on $f_{\bullet a}$, is the commonly known BPR function (Bureau of Public Roads) which this thesis uses for its results The route cost, denoted as c_{r_i}, is the sum of all the link costs that involve passing through a particular route r_i. Its mathematical definition is as follows:

$$c_{r_i}(\mathbf{h}) = \sum_{\forall a \in \boldsymbol{A}: a \in r_i} t_a[\mathbf{f}_{\bullet}(\mathbf{h})] = \sum_{\forall a \in \boldsymbol{A}} \left\{ t_a[\mathbf{f}_{\bullet}(\mathbf{h})] \cdot \delta_{a\, r_i} \right\} \quad \text{[A-6]}$$

Especially used by Aashtiani is the u_i, variable to which he refers sometimes as the *accessibility variable.* This variable, preferred to be named for this thesis as *minimum route cost*, refers to the minimum cost found among all the routes that connect a particular OD pair. Its mathematical definition is as follows:

$$u_i(\mathbf{h}) = \min_{\forall r_i \in \boldsymbol{R}_i} \left[c_{r_i}(\mathbf{h}) \right] \quad \text{[A-7]}$$

For the route costs, it is useful to define a vector $\mathbf{c}_i$. Likewise, it is useful, especially for Aashtiani's method, to define a vector **u** for the minimum route costs:

$$\mathbf{c}_i(\mathbf{h}) = \left[c_{1_i}(\mathbf{h})\, c_{2_i}(\mathbf{h}) \dots c_{|\boldsymbol{R}_i|_i}(\mathbf{h}) \right] \quad \text{[A-8a]}$$

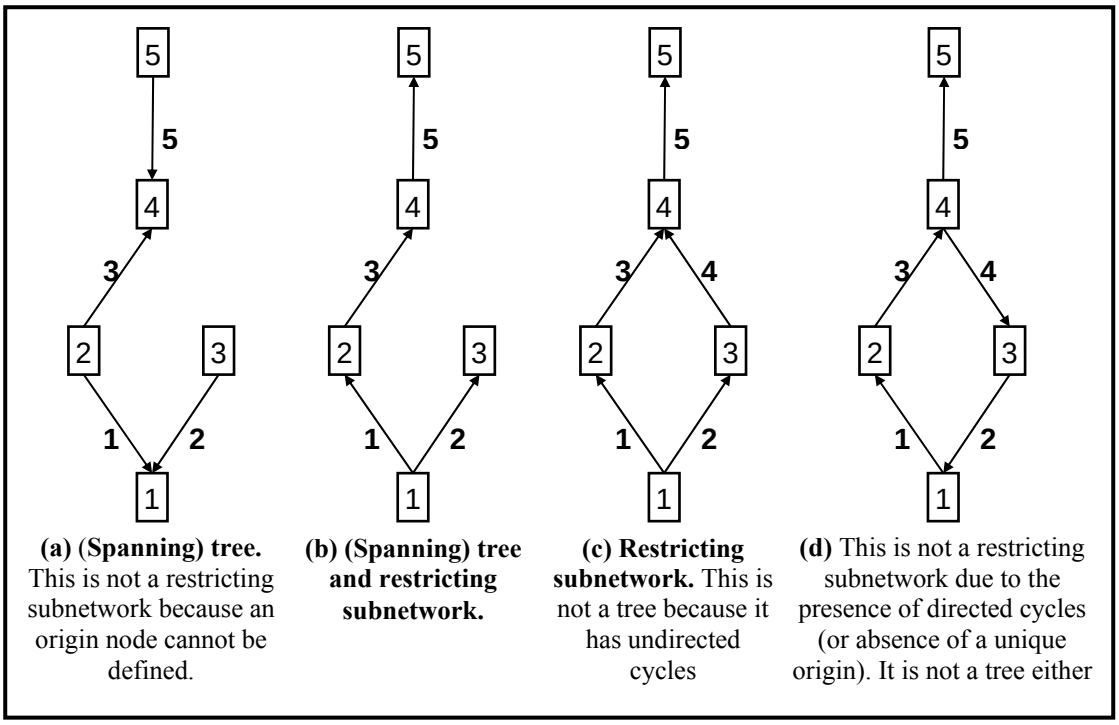
$$\mathbf{u}(\mathbf{h}) = \left[u_1(\mathbf{h})\, u_2(\mathbf{h}) \dots u_{|I|}(\mathbf{h}) \right] \quad \text{[A-8b]}$$

The reader should be aware of the following relationship which shows how the minimum route costs u_i and the route costs c_i are functions of **h.** This relationship is useful in the next section for understanding whether the solution to the S-TAP-F is unique or not.

$$u_i(\mathbf{h}) = u_i(\, \mathbf{c}_i\{\mathbf{h}\}\,) = u_i(\, \mathbf{c}_i\{\, \mathbf{t}[\, \mathbf{f}_{\bullet}(\mathbf{h})\,]\, \}\,) \quad \text{[A-9]}$$

Bar-Gera's algorithm is rich in concepts that allow explaining it more succinctly. These concepts include *restricting subnetworks*, *topological orders*, *maximum costs to a node*, *last common nodes*, *approach proportions*, *average approach costs*, *origin-based node flows*, *basic approaches* and *nonbasic approaches* among others. They assume important roles within the algorithm and therefore, their definitions will be introduced in the *Section "Bar-Gera's algorithm"* of this appendix. Nevertheless, there is one concept that requires important attention and which is used all along Bar-Gera's algorithm. This concept is the *restricting subnetwork* which he denotes as $\boldsymbol{A}_p$ because it is in essence, a subset of arcs with only one node acting as the origin. More formally, a restricting subnetwork $\boldsymbol{A}_p$ is the composition of the whole set of nodes $\boldsymbol{N}$ and a subset of the set of arcs $\boldsymbol{A}$ such that (1) there is at least one route from a specific node, named the *root*, to the rest of the nodes, and such that (2) the arcs do not form directed cycles. This definition forces any restricting subnetwork to have only one origin node p: the *root*. Although a restricting subnetwork is similar to other concepts used in graph theory such as *tree* and *spanning tree*, *Figure A-1* shows that there are differences. The specific node from which all routes start would be called *root* in graph theory but in this context, it is also an origin node p that belongs to the set $\boldsymbol{N}_o$.

The inclusion of *connectors* and *thru nodes* in a network is not specifically addressed by either Bar-Gera or Aashtiani. Nevertheless, the data used by this thesis requires an accurate definition of these two concepts. As it is often the case, including in the graphs used in this thesis, a zone does not represent an actual point in the network. Most of the times, a zone refers to a *centroid*, that is, an approximation to where the real starting (or ending) point of a group of trips really is. Therefore, the arcs that connect these zones to the rest of the network are virtual arcs, called *connectors*, whose link cost t_a does not depend on the level of congestion. The use of connectors is an approximation to reality which aims to simplify the model. Connectors should only constitute the first or the last arc of any route $r_{(p,\,q)}$. Since the cost of a connector is fixed, algorithms tend to choose routes that include connectors not only in the first or in the last arcs, and in consequence, algorithms choose invalid routes. For this reason, the data used for this thesis requires indicating which nodes are *thru nodes*. *Thru nodes* are nodes that, in a particular network, do not need to be at the beginning or at the end of a route. All thru nodes can be zones but not all zones can be thru nodes.



(a) (Spanning) tree. This is not a restricting subnetwork because an origin node cannot be defined.

(b) (Spanning) tree and restricting subnetwork.

(c) Restricting subnetwork. This is not a tree because it has undirected cycles

(d) This is not a restricting subnetwork due to the presence of directed cycles (or absence of a unique origin). It is not a tree either

Figure A-1. Examples that show the differences of a restricting subnetwork with a tree and a spanning tree.

Formulations

Although very different, the mathematical formulations used by Bar-Gera and Aashtiani share conditions that a solution should meet in order to solve the S-TAP-F. These conditions include Wardrop's first principle, conditions of conservation of flow and nonnegativity of the route flows. Due to the existence of these common conditions, one could lay out (as did Sheffi 1985, p. 65) the following formulation for the S-TAP-F which serves as a starting point for understanding what Bar-Gera and Aashtiani aim to accomplish with their formulations.

Find a vector **h** such that:

$$h_{r_i} \cdot \left[c_{r_i}(\mathbf{h}) - u_i(\mathbf{h}) \right] = 0 \qquad \forall r_i \in \boldsymbol{R}_i, \forall i \in \boldsymbol{I} \text{ [A-10a]}$$

$$c_{r_i}(\mathbf{h}) - u_i(\mathbf{h}) \geq 0 \qquad \forall r_i \in \boldsymbol{R}_i, \forall i \in \boldsymbol{I} \text{ [A-10b]}$$

$$\left(\sum_{\forall r_i \in \boldsymbol{R}_i} h_{r_i} \right) - d_i = 0 \qquad \forall i \in \boldsymbol{I} \text{ [A-10c]}$$

$$h_{r_i} \geq 0 \qquad \forall r_i \in \boldsymbol{R}_i, \forall i \in \boldsymbol{I} \text{ [A-10d]}$$

The expressions above describe mathematically the necessary conditions that turn a vector **h** into the optimal solution of the S-TAP-F. Condition [A-10a] is perhaps the

most important one. It translates Wardrop's first principle into mathematical notation. It states that if the cost of using path r_i is not equal to the minimum route cost u_i, then the flow h_{r_i} should be equal to zero. Alternatively, if the cost of using path r_i is equal to u_i, then users as, with the shortest path, will also use path r_i. Expression [A-10b] serves as a definition of u_i, since it establishes that no path should be less costly than u_i. Expression [A-10c] establishes that the sum of all the route flows should be equal to the demand. This condition is commonly referred as a constraint of conservation of flow.

The above formulation does not state any assumptions regarding the nature of the performance functions.

A very important aspect to take into account in the above formulation is that there are several optimal solutions **h** but just one optimal solution $\mathbf{f}_\bullet$ (recall how c_{r_i}, u_i, $\mathbf{f}_\bullet$ and **h** relate to each other as expressed in [A-6a]). Since c_{r_i} and u_i are functions of **t** but every element t_a is a function of multiple route flows h_{r_i}, then there are several solutions **h** to the S-TAP-F as formulated in [A-10]. Sheffi (1985, p. 67-68) and Aashtiani (1979, pp. 52-53) showed through simple examples the lack of a unique solution **h** to the S-TAP-F. In consequence, the following two formulations also do not have a unique solution **h** but do have a unique solution $\mathbf{f}_\bullet$.

Beckman's transformation

Bar-Gera uses Beckmann's transformation as the model for his method. Beckmann's transformation, a mathematical programming problem with linear constraints and a nonlinear objective function, is as follows.

Find a vector **h** such that

$$\text{minimizes } T[\mathbf{f}_\bullet(\mathbf{h})] = \sum_{\forall a \in \boldsymbol{A}} \int_0^{f_{\bullet a}(\mathbf{h})} t_a(x)\,\mathrm{d}x \qquad \text{[A-11a]}$$

subject to

$$\sum_{\forall r_i \in \boldsymbol{R}_i} h_{r_i} - d_i = 0 \qquad \forall i \in \boldsymbol{I} \qquad \text{[A-11b]}$$

$$h_{r_i} \geq 0 \qquad \forall r_i \in \boldsymbol{R}_i; \forall i \in \boldsymbol{I} \qquad \text{[A-11c]}$$

T represents the objective function and it is directly defined in terms of total link flows $\mathbf{f}_\bullet$. But every total link flow $f_{a\bullet}$, by definition (see [A-4]), is a function of route flows h_{r_i}. This formulation is an artificial optimization problem because T does not have a physical interpretation. Nevertheless, its optimal solution **h** complies with the conditions shown in [A-10] and in this way, it becomes a solution to the S-TAP-F (for a demonstration, see Bar-Gera 1999, pp. 6-7; or Sheffi 1985, pp. 63-65). For

conditions [A-10] to hold and for the optimal solution $\mathbf{f}_{\bullet}$ to be unique, the model makes four important assumptions regarding the performance functions. The second to last s*ection* of *Chapter 2* explains these assumptions.

Aashtiani's formulation

Aashtiani reframes the basic formulation shown in [A-10] by expanding the solution vector **h** with the vector **u**. Therefore, every u_i becomes a new unknown variable and not simply a function of **h**. His formulation is as follows:

Find a vector $\mathbf{h} \mid \mathbf{u} =$

$$\left[h_{1_1}\ h_{2_1} \ldots h_{|\boldsymbol{R}_1|_1}\ h_{1_2}\ h_{2_2} \ldots h_{|\boldsymbol{R}_2|_2}\ \ldots h_{1_{|\boldsymbol{I}|}}\ h_{2_{|\boldsymbol{I}|}} \ldots h_{|\boldsymbol{R}_{|\boldsymbol{I}|}|_{|\boldsymbol{I}|}}\ u_1\ u_2 \ldots u_{|\boldsymbol{I}|} \right] \text{such that}$$

$$h_{r_i} \cdot \left\lfloor c_{r_i}(\mathbf{h}) - u_i \right\rfloor = 0 \qquad \forall r_i \in \boldsymbol{R}_i, \forall i \in \boldsymbol{I} \quad \text{[A-12a]}$$

$$c_{r_i}(\mathbf{h}) - u_i \geq 0 \qquad \forall r_i \in \boldsymbol{R}_i, \forall i \in \boldsymbol{I} \quad \text{[A-12b]}$$

$$\left(\sum_{\forall r_i \in \boldsymbol{R}_i} h_{r_i} \right) - d_i = 0 \qquad \forall i \in \boldsymbol{I} \quad \text{[A-12c]}$$

$$h_{r_i} \geq 0 \qquad \forall r_i \in \boldsymbol{R}_i, \forall i \in \boldsymbol{I} \quad \text{[A-12d]}$$

$$u_i \geq 0 \qquad \forall i \in \boldsymbol{I} \quad \text{[A-12e]}$$

Aashtiani proved that as long as the performance function t_a is positive (the second to last *Section* of *Chapter 2* discusses these assumptions in detail), the formulation [A-12] is equivalent to the following one:

Find a vector $\mathbf{h} \mid \mathbf{u} =$

$$\left[h_{1_1}\ h_{2_1} \ldots h_{|\boldsymbol{R}_1|_1}\ h_{1_2}\ h_{2_2} \ldots h_{|\boldsymbol{R}_2|_2}\ \ldots h_{1_{|\boldsymbol{I}|}}\ h_{2_{|\boldsymbol{I}|}} \ldots h_{|\boldsymbol{R}_{|\boldsymbol{I}|}|_{|\boldsymbol{I}|}}\ u_1\ u_2 \ldots u_{|\boldsymbol{I}|} \right]$$ such that

$$h_{r_i} \cdot \left\lfloor c_{r_i}(\mathbf{h}) - u_i \right\rfloor = 0 \qquad \forall r_i \in \boldsymbol{R}_i,\ \forall i \in \boldsymbol{I} \quad \text{[A-13a]}$$

$$c_{r_i}(\mathbf{h}) - u_i \geq 0 \qquad \forall r_i \in \boldsymbol{R}_i,\ \forall i \in \boldsymbol{I} \quad \text{[A-13b]}$$

$$u_i \cdot \left[\sum_{\forall r_i \in \boldsymbol{R}_i} h_{r_i} - d_i \right] = 0 \qquad \forall i \in \boldsymbol{I} \quad \text{[A-13c]}$$

$$\left(\sum_{\forall r_i \in \boldsymbol{R}_i} h_{r_i} \right) - d_i \geq 0 \qquad \forall i \in \boldsymbol{I} \quad \text{[A-13d]}$$

$$h_{r_i} \geq 0 \qquad \forall r_i \in \boldsymbol{R}_i,\ \forall i \in \boldsymbol{I} \quad \text{[A-13e]}$$

$$u_i \geq 0 \qquad \forall i \in \boldsymbol{I} \quad \text{[A-13f]}$$

where $\begin{bmatrix} c_{1_1}(\mathbf{h}) - u_1 \\ \cdots \\ c_{|\boldsymbol{R}_1|_1}(\mathbf{h}) - u_1 \\ \cdots \\ c_{1_{|\boldsymbol{I}|}}(\mathbf{h}) - u_{|\boldsymbol{I}|} \\ \cdots \\ c_{|\boldsymbol{R}_{|\boldsymbol{I}|}|_{|\boldsymbol{I}|}}(\mathbf{h}) - u_{|\boldsymbol{I}|} \\ \sum_{\forall r_1 \in \boldsymbol{R}_1} h_{r_i} - d_1 \\ \cdots \\ \sum_{\forall r_{|\boldsymbol{I}|} \in \boldsymbol{R}_{|\boldsymbol{I}|}} h_{r_{|\boldsymbol{I}|}} - d_{|\boldsymbol{I}|} \end{bmatrix}^T$ is a nonlinear function of $\mathbf{h} \mid \mathbf{u}$.

The reader can observe that this formulation simply contains conditions [A-13c] and [A-13d] instead of [A-12c]. Now, formulation [A-13] is a nonlinear complementarity problem since it presents the following structure:

Find a vector $\mathbf{x} = [x_1\ x_2\ \ldots\ x_n]$ such that

$$\mathbf{x}\ \mathbf{y}(\mathbf{x})^{\mathrm{T}} = 0 \qquad \text{[A-14a]}$$

$$\mathbf{y}(\mathbf{x}) \geq 0 \qquad \text{[A-14b]}$$

$$\mathbf{x} \geq 0 \qquad \text{[A-14c]}$$

where $\mathbf{y}(\mathbf{x}) = [y_1(\mathbf{x})\ y_2(\mathbf{x}) \ldots\ y_n(\mathbf{x})]$ is a nonlinear function of **x**.

In consequence, formulation [A-12] is a nonlinear complementarity problem. Following the formulation shown in [A-14], and to simplify future notation, this thesis will represent by an **x,** the solution of any complementarity problem, and by a **y**, the vector function of **x**.

Algorithms

Using the above formulations, Bar-Gera (1999) and Aashtiani (1979) proposed the following algorithms. While Bar-Gera proved that his algorithm converges, Aashtiani showed its convergence through a range of examples. Nevertheless, as *Chapter 3* will show, Aashtiani's algorithm always converged.

Bar-Gera's Algorithm

The description presented in this subsection summarizes what Bar-Gera presented in his doctoral dissertation (1999) and to a lesser extent in a following publication (2002). The main characteristics of Bar-Gera's algorithm are the following: (1) it is an iterative algorithm, (2) it obtains a solution in terms of origin-based link flows f_{pa}, (3) it carries out a Newton-type search procedure, and (4) it does not manipulate the whole network but a restricting subnetwork $\boldsymbol{A}_p$ for each origin *p*. In other words, Bar-Gera's algorithm decomposes the problem by origins.

Figure A-2 shows a simplified version of Bar-Gera's algorithm. The complete algorithm contains a small addition within the cycle shown later on *Figure A-5*. Bar-Gera's algorithm works as follows. It starts with an initial origin-based link flow $\mathbf{f}_p$ for every origin *p* (the sum of all these origin-based link flows is equal to the solution of the problem, that is, the vector of total link flows $\mathbf{f}_\bullet = \sum_{\forall p \in \boldsymbol{N}_q} \mathbf{f}_p$). Every initial $\mathbf{f}_p$ contains only a subset of links (with positive flow) which define a subnetwork $\boldsymbol{A}_p$. Having now an initial $\mathbf{f}_p$ and an initial $\boldsymbol{A}_p$ for every origin *p*, the algorithm starts a series of iterations. At every *cycle* (the most external loop), for each origin *p*, the algorithm finds a new (and better) feasible solution by answering two questions: (1) Which links should be removed or included? In other words, how to update $\boldsymbol{A}_p$? (2) How much flow should be assigned to the links? In other words, how to update $\mathbf{f}_p$? To

answer the first question, the algorithm executes a sub-algorithm that modifies the existing restricting subnetwork $\boldsymbol{A}_p$. To answer the second question, the algorithm executes a second sub-algorithm that shifts existing origin-based link flows among the links of the subnetwork $\boldsymbol{A}_p$. In order to carry out this type of shifts, the sub-algorithm uses a *Newton-type procedure* that due to its particular features, Bar-Gera denominates it *boundary search*. After each iteration, the algorithm evaluates expression [A-11a] with the new solution $\mathbf{f}_\bullet$ and checks if it generates a satisfactory minimum value of *T*. When the algorithm no longer finds a lower value of *T*, it terminates. Since the algorithm does not obtain route flows h_{r_i}, it cannot evaluate condition [A-11b] directly. Simply, the algorithm guarantees that its procedure does not violate condition [A-11b].

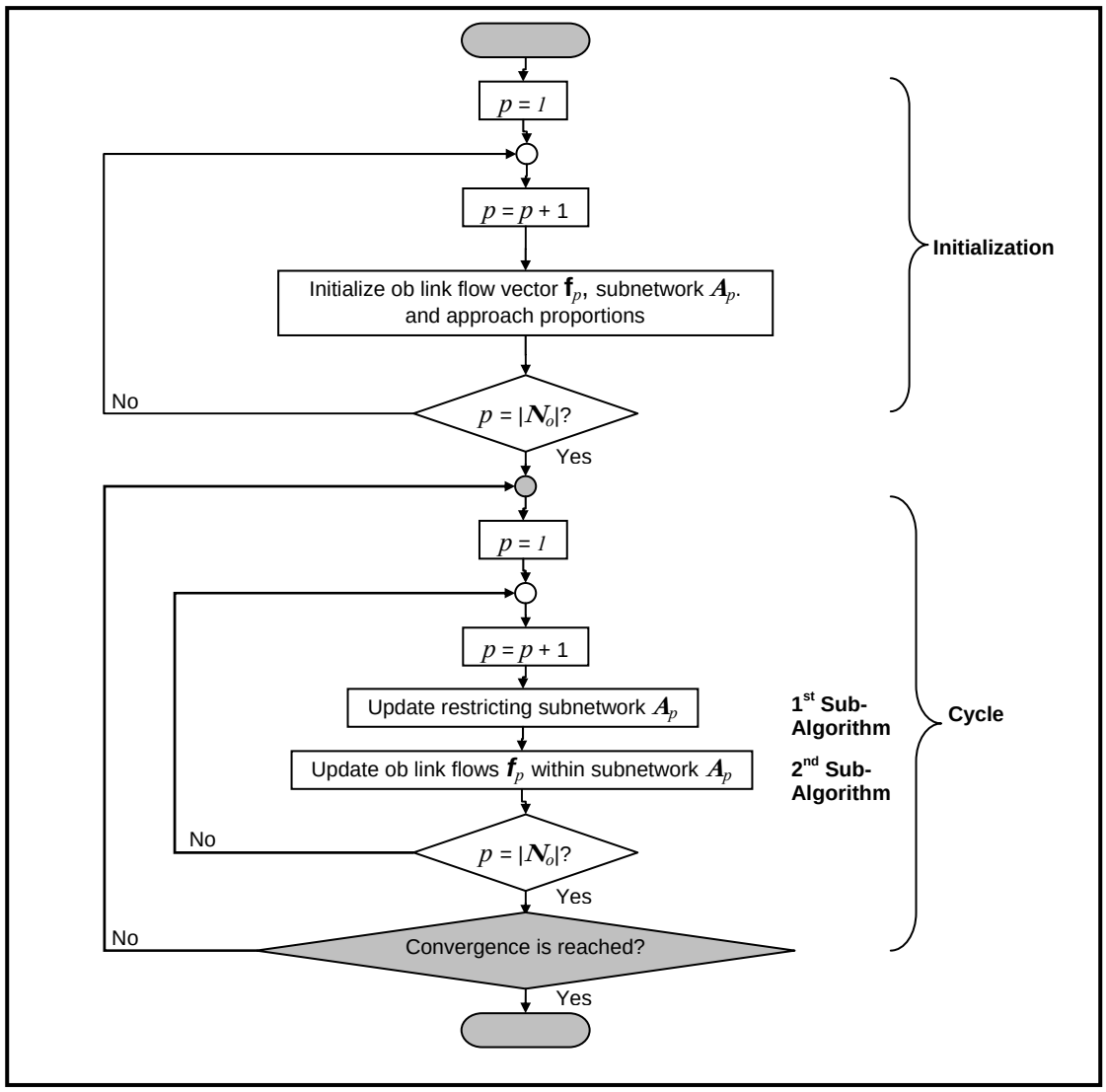
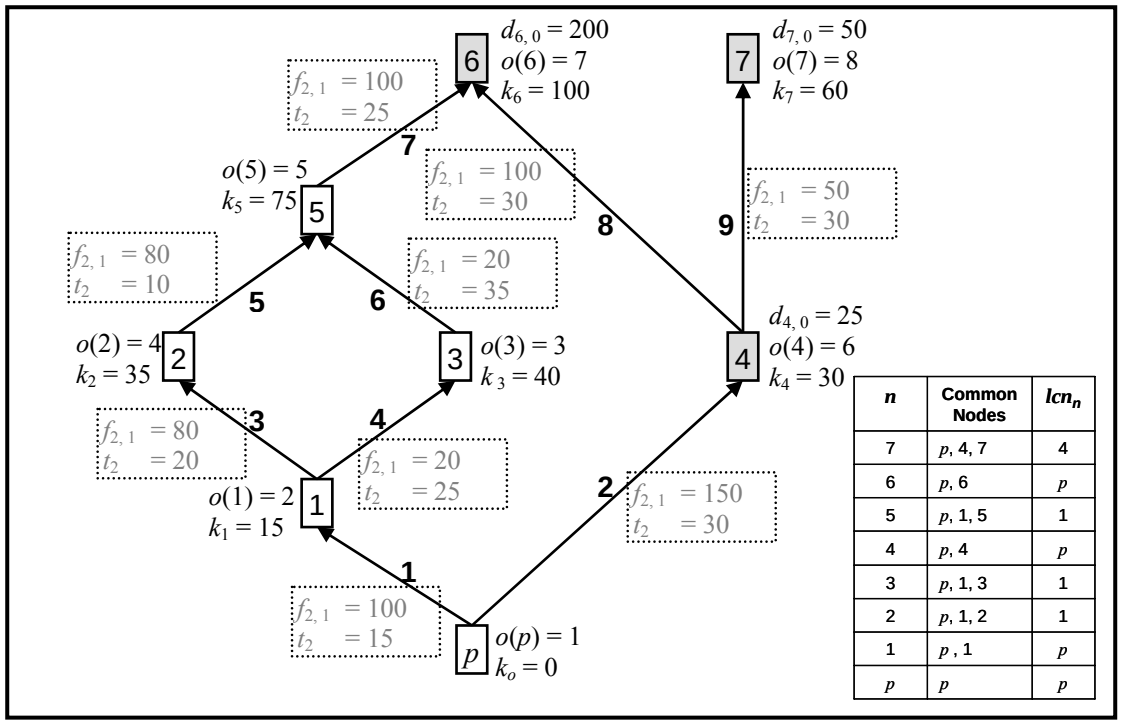


Figure A-2. Simplified version of Bar-Gera's origin-based algorithm. "ob" stands for "origin-based".

The initialization consists mainly in obtaining two important outputs: an initial feasible solution $\mathbf{f}_p$ and an initial *restricting subnetwork* $\boldsymbol{A}_p$ for every origin node *p*. In other words, every link *a* within each restricting subnetwork $\boldsymbol{A}_p$ has a link flow (actually, an origin based link flow $f_{p,a}$) assigned to it. Bar-Gera's algorithm does not require any specific procedure for the initialization phase. Therefore, the well-known "all-or-nothing assignment" (Sheffi 1985, p. 111) is one of several procedures for obtaining the above two outputs. Another output of this phase, which will become important for the first sub-algorithm, are the approach proportions, defined later in [A-21].

The first sub-algorithm addresses the question of which links should contain non-zero flow. Bar-Gera demonstrated that the optimal solution **h*** for the S-TAP-F is "acyclic by origin" (2002, p. 401) meaning that if **h*** is the optimal solution to the S-TAP-F, then the links that contain positive flow, as dictated by each resulting origin-based flow vector $\mathbf{f}_p = \mathbf{f}_p(\mathbf{h}^*)$, should not describe any directed cycles. Consequently, he chose to solve the S-TAP-F by decomposing it by origins and for each origin *p*, assign positive origin-based link flow $f_{p,a}$ only to links that constitute a restricting subnetwork $\boldsymbol{A}_p$.

The use of restricting subnetworks $\boldsymbol{A}_p$ allows the definition of three useful concepts needed in the algorithm: *topological order*, *maximum cost to a node*, and *last common node*. Given a restricting subnetwork $\boldsymbol{A}_p$, *topological order* refers to a label (a number from 1 to $|\boldsymbol{A}_p|$) that every node *i* receives indicating whether it precedes or proceeds another node *j* if they belong to the same route $r_{(p,q)}$. The topological order of a node *n* is denoted by *o*(*n*). Given a restricting subnetwork $\boldsymbol{A}_p$, the *maximum cost to node n* refers to the largest cost among the routes that connect origin *p* with node *n*. It is denoted by k_n. The *common nodes* of a node *n* are all the nodes shared by all routes $r_{(p,n)} \in \boldsymbol{R}_{(p,n)}$. The *last common node of node n* is denoted by lcn_n and it refers to the node with the largest topological order found in the set of common nodes of *n* minus *n*. *Figure A-3* shows an example of how to calculate the topological order, the maximum cost and the last common node of every node within a restricting subnetwork $\boldsymbol{A}_p$.



n	Common Nodes	lcn_n
7	*p*, 4, 7	4
6	*p*, 6	*p*
5	*p*, 1, 5	1
4	*p*, 4	*p*
3	*p*, 1, 3	1
2	*p*, 1, 2	1
1	*p* , 1	*p*
p	*p*	*p*

Figure A-3. Different elements within a subnetwork $\boldsymbol{A}_p$: last common nodes (lcn_n), maximum costs (k_n), and topological orders ($o(n)$). Origin-based link flows ($f_{p,\,a}$), link travel costs ($t_a(\mathbf{f}._{\,a})$) and demands ($d_{(p,\,q)}$) are also shown.

With the initial $\boldsymbol{A}_p$ obtained from the initialization, the first sub-algorithm executes three steps: it removes the unused links ($\forall a \in \boldsymbol{A}_p : f_{p,a} = 0$); it calculates the maximum costs k_n and it adds every link a not in the subnetwork such that $k_{a_t} \le k_{a_h}$ .These steps guarantee the construction of a new subnetwork $\boldsymbol{A}_p$. in which the algorithm will start seeking a new solution $f_{p,a}$. *Figure A-4* shows the complete description of the first sub-algorithm. The sub-algorithm ends by calculating the new topological orders, by calculating the new last common nodes and by updating the data structures that the algorithm uses to store the network (for more details on these data structures, the reader can refer to *Section "Discussion on the Data Structures Recommended for the Implementation", Chapter 2*). Bar-Gera's algorithm does not recommend any particular procedure for calculating the topological ordering. As shown later, calculating the topological ordering and the last common nodes is important for the execution of the second sub-algorithm.

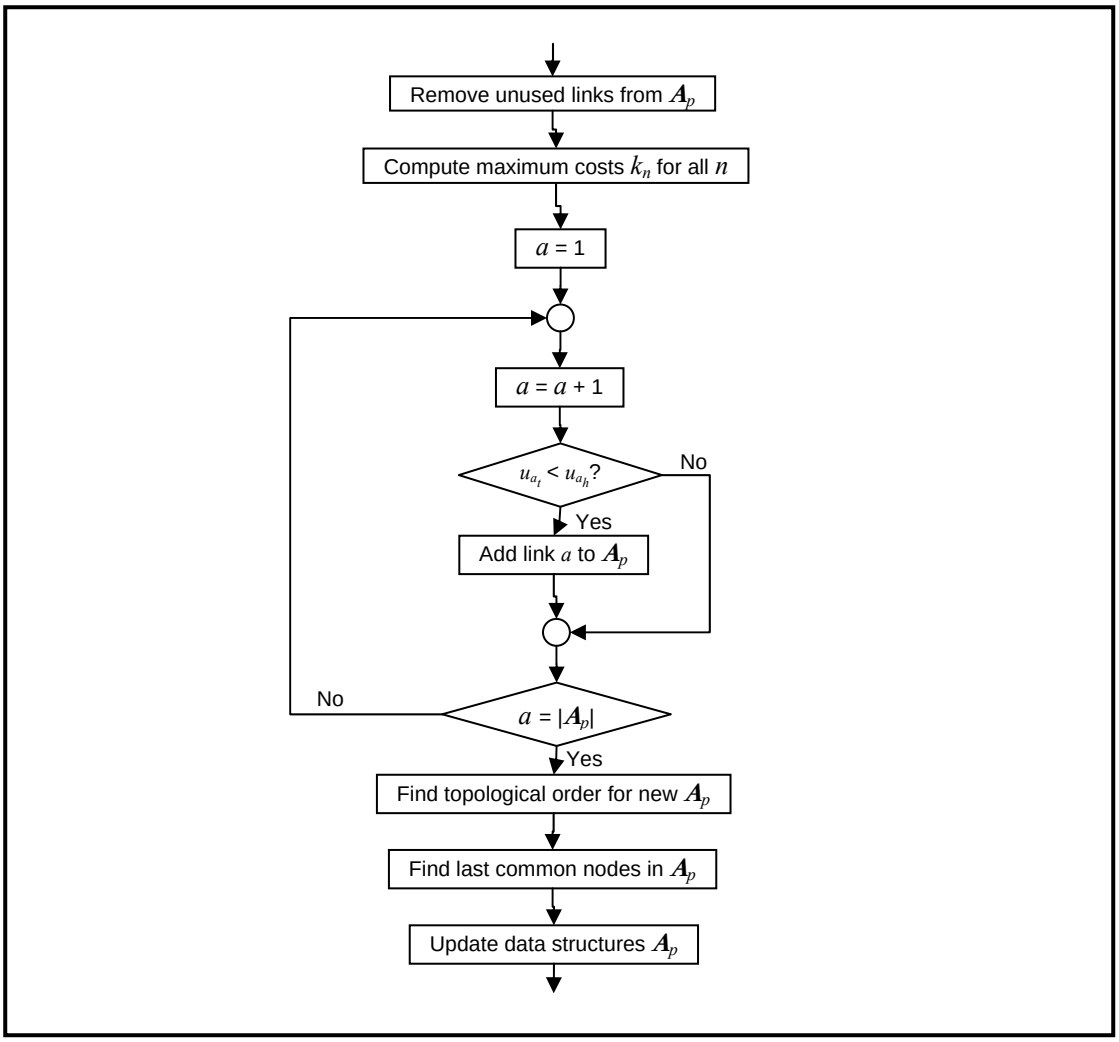


Figure A-4. First sub-algorithm in Bar-Gera's method.

The second sub-algorithm addresses the question of, given a restricting subnetwork $\boldsymbol{A}_p$, how much flow to assign to its links. Actually, since the subnetworks $\boldsymbol{A}_p$ obtained from the initialization have already an assigned flow $\mathbf{f}_p$ and since the algorithm should guarantee the conservation of flow (that is, constraints [A-10c] or [A-11b] must always hold), the question becomes "how to shift the existing flows $f_{p,\,a}$ within $\boldsymbol{A}_p$?" Bar-Gera addresses this question by using a Newton-type procedure denominated *boundary search.* This procedure consists in finding a flow shift that leads to a new origin-based flow vector $\mathbf{f}_p$ (and $\mathbf{f}$) such that T decreases in value. As with any other Newton-type procedure, the flow shift is the result of first calculating the *Newton step* (an initial vector $\Delta\mathbf{f}$ that points to the new solution $\mathbf{f}$). Then, contrary to regular convex search procedures, the sub-algorithm multiplies this step by a factor $\lambda \in [0, 1]$ and then modifies it by taking into account boundary constraints (non-negativity constraints). Due to the boundary constraints applied at the end, different values of λ generate different flow shifts. Therefore, the sub-algorithm needs finding the best λ. For this reason, the sub-algorithm is iterative by starting with a value of $\lambda = 1$ and

decreasing it at every iteration until the following directional derivative becomes less than zero:

$$\Delta\mathbf{f}\cdot\nabla T\left(\mathbf{f}+\lambda\cdot\Delta\mathbf{f}\right)=\Delta\mathbf{f}\cdot\mathbf{t}\left(\mathbf{f}+\lambda\cdot\Delta\mathbf{f}\right) \qquad \text{[A-15]}$$

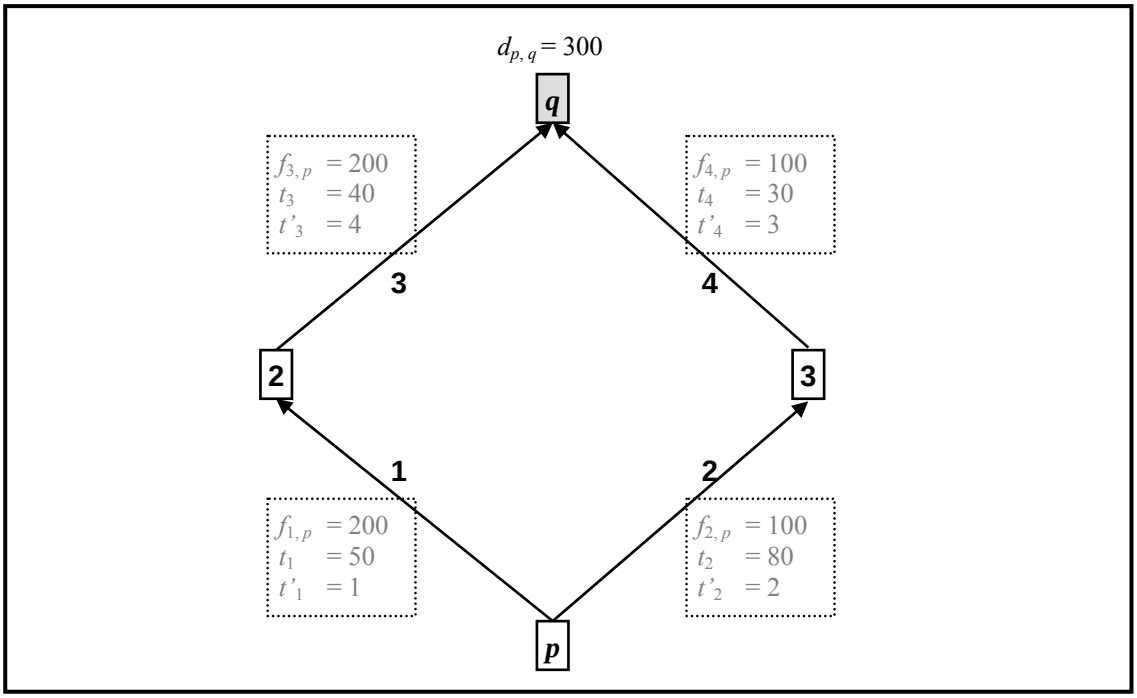


Figure A-5. Example of a two-route, one destination subnetwork $\boldsymbol{A}_p$ where an initial flow has been assigned to it. Variables shown in boxes refer to initial origin-based link flows values $f_{p,a}$, and link costs t_a and link cost derivatives t'_a evaluated with the initial flow values $f_{p,a}$. The left route $r_{p,q} = 1_{p,q}$ comprises links $a = 1$ and $a = 3$. The right route $r_{(p,q)} = 2_{(p,q)}$ comprises links $a = 2$ and $a = 4$.

The second sub-algorithm is straightforward but it involves many new terms. Defining this new terminology requires the aid of figures in order to explain them. Otherwise, the terms can seem abstract. In consequence, the reader will now be presented with a simple example that explains some of these new terms and that will serve as the basis for understanding the whole second sub-algorithm. *Figure A-5* presents this first simple example.

Figure A-5 presents an example of a subnetwork $\boldsymbol{A}_p$ with only two routes and one destination node. Throughout this example, the origin-based link flows belonging to other subnetworks will remain constant no matter what changes the subnetwork $\boldsymbol{A}_p$ experiences. The costs t_a are the result of applying a given performance function to the given origin-based link flows $f_{p,a}$ shown in the figure and to the origin-based link flows of the other subnetworks. The derivatives of the performance functions t'_a are evaluated in the same manner. The left route $r_{(p,q)} = 1_{(p,q)}$ has a lower cost than the right route $r_{(p,q)} = 2_{(p,q)}$, that is, $c_{1_{(p,q)}} < c_{2_{(p,q)}}$. In other words, the flow that this subnetwork has assigned to it does not comply with Wardrop's first principle. To

make it compliant with that principle, probably the most reasonable alternative is to shift flow, say $\Delta h_{r_{(p,q)}}$, from the most costly route $2_{(p,q)}$ to the least costly route $1_{(p,q)}$so that both routes have the same travel cost $c_{r_{(p,q)}}$. This alternative would require finding the value of $\Delta h_{r_{(p,q)}}$ by following these three steps: (1) defining the costs $c_{1_{(p,q)}}$ and $c_{2_{(p,q)}}$ in terms of $\Delta h_{r_{(p,q)}}$, (2) equalizing both costs $c_{1_{(p,q)}}$ and $c_{2_{(p,q)}}$, and (3) solving for $\Delta h_{r_{(p,q)}}$. Bar-Gera's algorithm follows this alternative and it is the basis of the second sub-algorithm. For the first step, it uses a linear approximation to determine $c_{r_{(p,q)}}\left(\Delta h_{r_{(p,q)}}\right)$ as shown below:

$$c_{r_{(p,q)}}\left(\Delta h_{r_{(p,q)}}\right)=\sum_{\forall a\in \boldsymbol{A}_p : a\subseteq r_{(p,q)}}\left[t_a\left(\Delta h_{r_{(p,q)}}\right)\right] \approx \sum_{\forall a\in \boldsymbol{A}_p : a\subseteq r_{(p,q)}}\left(t'^0_a\cdot\Delta h_{r_{(p,q)}}+t^0_a\right) \qquad \text{[A-16a]}$$

where

$$\Delta h_{r_{(p,q)}}=\begin{cases}\Delta h_{r_{(p,q)}} & \text{if } r_{(p,q)}=1_{(p,q)} \quad \text{(left route)}\\ -\Delta h_{r_{(p,q)}} & \text{if } r_{(p,q)}=2_{(p,q)} \quad \text{(right route)}\end{cases} \qquad \text{[A-16b]}$$

and t^0_a and t'^0_a are initial values of the link cost and its derivative.

The above expressions lead to the following calculations:

$$t_1\left(\Delta h_{r_{(p,q)}}\right)\approx t'^0_1\cdot\Delta h_{r_{(p,q)}}+t^0_1=1\cdot\Delta h_{r_{(p,q)}}+50 \qquad \text{[A-17a]}$$

$$t_3\left(\Delta h_{r_{(p,q)}}\right)\approx t'^0_3\cdot\Delta h_{r_{(p,q)}}+t^0_3=4\cdot\Delta h_{r_{(p,q)}}+40 \qquad \text{[A-17b]}$$

$$t_2\left(-\Delta h_{r_{(p,q)}}\right)\approx t'^0_2\cdot\left(-\Delta h_{r_{(p,q)}}\right)+t^0_2=-2\cdot\Delta h_{r_{(p,q)}}+80 \qquad \text{[A-17c]}$$

$$t_4\left(-\Delta h_{r_{(p,q)}}\right)\approx t'^0_4\cdot\left(-\Delta h_{r_{(p,q)}}\right)+t^0_4=-3\cdot\Delta h_{r_{(p,q)}}+30 \qquad \text{[A-17d]}$$

$$c_{1_{(p,q)}}\left(\Delta h_{r_{(p,q)}}\right)\approx\left(1\cdot\Delta h_{r_{(p,q)}}+50\right)+\left(4\cdot\Delta h_{r_{(p,q)}}+40\right) \qquad \text{[A-17e]}$$

$$c_{2_{(p,q)}}\left(-\Delta h_{r_{(p,q)}}\right)\approx\left(-2\cdot\Delta h_{r_{(p,q)}}+80\right)+\left(-3\cdot\Delta h_{r_{(p,q)}}+30\right) \qquad \text{[A-17f]}$$

Executing the second and third steps would be as follows:

$$c_{1_{(p,q)}}\left(\Delta h_{r_{(p,q)}}\right) \approx c_{2_{(p,q)}}\left(-\Delta h_{r_{(p,q)}}\right) \quad \text{[A-18a]}$$

$$\left(t'^{0}_{1}\cdot\Delta h_{r_{(p,q)}}+t^{0}_{1}\right)+\left(t'^{0}_{3}\cdot\Delta h_{r_{(p,q)}}+t^{0}_{3}\right)\approx\left(-t'^{0}_{2}\cdot\Delta h_{r_{(p,q)}}+t^{0}_{2}\right)+\left(-t'^{0}_{4}\cdot\Delta h_{r_{(p,q)}}+t^{0}_{4}\right) \quad \text{[A-18b]}$$

$$\Delta h_{r_{(p,q)}} \approx -\frac{\left(t^{0}_{1}+t^{0}_{3}\right)-\left(t^{0}_{2}+t^{0}_{4}\right)}{\left(t'^{0}_{1}+t'^{0}_{3}\right)+\left(t'^{0}_{2}+t'^{0}_{4}\right)} \quad \text{[A-18c]}$$

$$\Delta h_{r_{(p,q)}} \approx 2 \quad \text{[A-18d]}$$

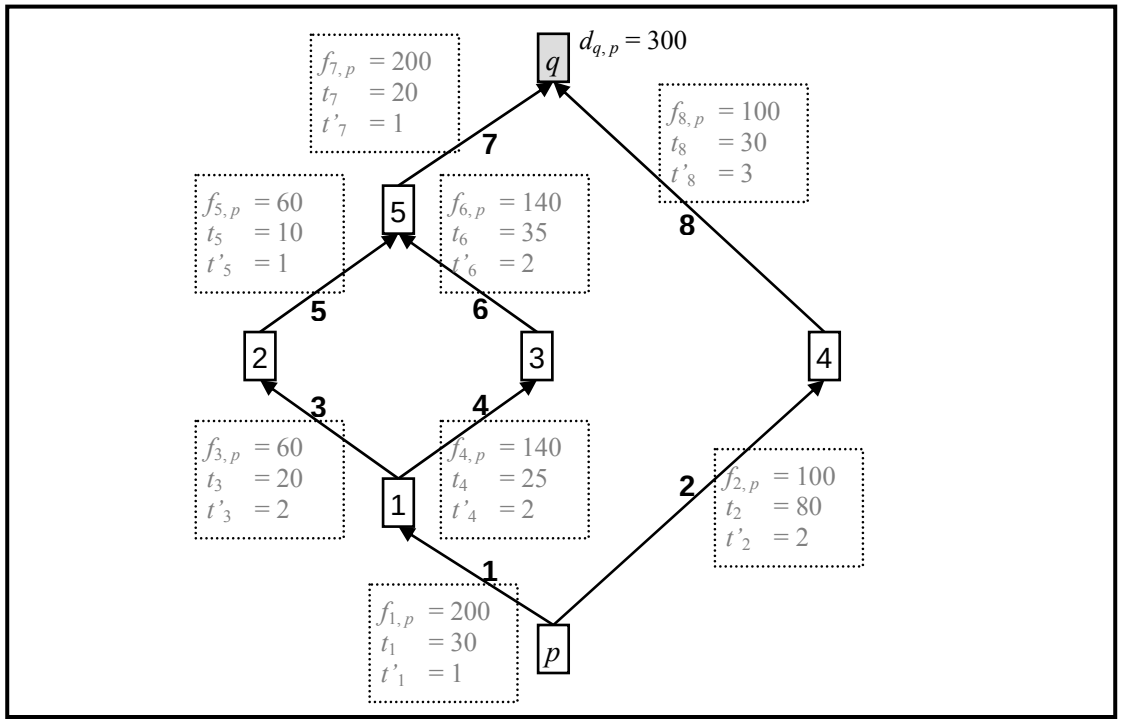


Figure A-6. Example of a three-route, one destination subnetwork $\boldsymbol{A}_p$ where an initial flow has been assigned to it. Variables shown in boxes refer to initial link flows values $f_{a,p}$, and link costs t_a and link cost derivatives t'_a evaluated at the initial link flow values. On the left, there are two routes: $r_{p,q} = 1_{p,q} = [p, 1, 2, 5, q]$ and $r_{p,q} = 3_{p,q} = [p, 1, 3, 5, q]$. On the right, there is one route: $r_{p,q} = 3_{p,q} = [p, 4, q]$.

The resulting formula [A-18c] follows the pattern of a Newton's method since the numerator is the first derivative and the denominator is the second derivative of the objective function *T*. Following Newton' method's jargon, this thesis will refer to [A-18c] as a *Newton step*. Following the same three steps, formula [A-18c] becomes much more complicated with a simple inclusion of an additional route as shown in *Figure 28*.

Contrary to the subnetwork in *Figure A-5*, the subnetwork in *Figure A-6* does not suggest that a simple shift of flow from one route to another would generate a flow assignment that complies with Wardrop's first principle. Nevertheless, if one focuses

on the flow that arrives by link a = 7 with the flow that arrives by link a =8, one could conclude that a simple shift of flow from the former link to the latter, or vice versa, guarantees that the travel cost of arriving by both links would be equal. The challenge is now how to define the cost that arrives by link 7. Bar-Gera proposes evaluating it based on the percentage of flow that is currently passing by $[p,1,2,5,q]$ and by $[p,1,3,5,q]$. Following the simple calculation made in the first simple example, the execution of the first step would be equal except for expression *[25i]*, where Bar-Gera suggested an original idea (for simplicity the notation Δh replaces the notation $\Delta h_{r_{(p,q)}}$):

$$t_1(\Delta h) \approx t'^0_1 \cdot \Delta h + t^0_1 = 1 \cdot \Delta h + 30 \quad \text{[A-19a]}$$

$$t_3(0.3 \cdot \Delta h) \approx t'^0_3 \cdot (0.3 \cdot \Delta h) + t^0_3 = 0.6 \cdot \Delta h + 20 \quad \text{[A-19b]}$$

$$t_5(0.3 \cdot \Delta h) \approx t'^0_5 \cdot (0.3 \cdot \Delta h) + t^0_5 = 0.3 \cdot \Delta h + 10 \quad \text{[A-19c]}$$

$$t_7(\Delta h) \approx t'^0_7 \cdot \Delta h + t^0_7 = 1 \cdot \Delta h + 20 \quad \text{[A-19d]}$$

$$t_4(0.7 \cdot \Delta h) \approx t'^0_4 \cdot (0.7 \cdot \Delta h) + t^0_4 = 1.4 \cdot \Delta h + 25 \quad \text{[A-19e]}$$

$$t_6(0.7 \cdot \Delta h) \approx t'^0_6 \cdot (0.7 \cdot \Delta h) + t^0_6 = 1.4 \cdot \Delta h + 35 \quad \text{[A-19f]}$$

$$t_2(-\Delta h) \approx t'^0_2 \cdot (-\Delta h) + t^0_2 = -2 \cdot \Delta h + 80 \quad \text{[A-19g]}$$

$$t_8(-\Delta h) \approx t'^0_8 \cdot (-\Delta h) + t^0_8 = -3 \cdot \Delta h + 30 \quad \text{[A-19h]}$$

$$c_{[p,1,*,5,q]}(\Delta h) \approx (1 \cdot \Delta h + 30) + 0.3 \cdot [(0.6 \cdot \Delta h + 20) + (0.3 \cdot \Delta h + 10)] + 0.7 \cdot [(1.4 \cdot \Delta h + 25) + (1.4 \cdot \Delta h + 35)] + (1 \cdot \Delta h + 20) \quad \text{[A-19i]}$$

$$c_{2_{(p,q)}}(-\Delta h) \approx (-2 \cdot \Delta h + 80) + (-3 \cdot \Delta h + 30) \quad \text{[A-19j]}$$

The expression [A-19i] represents an innovative idea of how to define the cost of arriving by a link. As later this thesis will show, this idea is the basis of what Bar-Gera defines as *average approach cost*. Following the calculations made in the first simple example, the execution of the second and third steps would be as follows:

$$c_{[p,1,*,5,q]}(\Delta h) \approx c_{2_{(p,q)}}(-\Delta h) \qquad \text{[A-20a]}$$

$$\begin{aligned} &\left(t'^{0}_{1}\cdot\Delta h + t^{0}_{1}\right)+ \\ &0.3\cdot\left[\left(0.3\cdot t'^{0}_{3}\cdot\Delta h + t^{0}_{3}\right)+\left(0.3\cdot t'^{0}_{5}\cdot\Delta h + t^{0}_{5}\right)\right]+ \\ &0.7\cdot\left[\left(0.7\cdot t'^{0}_{4}\cdot\Delta h + t^{0}_{4}\right)+\left(0.7\cdot t'^{0}_{6}\cdot\Delta h + t^{0}_{6}\right)\right]+ \\ &\left(t'^{0}_{7}\cdot\Delta h + t^{0}_{7}\right) \approx \left(-t'^{0}_{2}\cdot\Delta h + t^{0}_{2}\right)+ \\ &\left(-t'^{0}_{8}\cdot\Delta h + t^{0}_{8}\right) \end{aligned} \qquad \text{[A-20b]}$$

$$\Delta h \approx -\frac{\left\{t^{0}_{1}+\left[0.3\cdot\left(t^{0}_{3}+t^{0}_{5}\right)+0.7\cdot\left(t^{0}_{4}+t^{0}_{6}\right)\right]+t^{0}_{7}\right\}-\left(t^{0}_{2}+t^{0}_{8}\right)}{\left\{t'^{0}_{1}+\left[0.3^{2}\cdot\left(t'^{0}_{3}+t'^{0}_{5}\right)+0.7^{2}\cdot\left(t'^{0}_{4}+t'^{0}_{6}\right)\right]+t'^{0}_{7}\right\}+\left(t'^{0}_{2}+t'^{0}_{8}\right)} \qquad \text{[A-20c]}$$

$$\Delta h \approx -\frac{\left\{t^{0}_{1}+\left[\alpha\cdot\left(t^{0}_{3}+t^{0}_{5}\right)+(1-\alpha)\cdot\left(t^{0}_{4}+t^{0}_{6}\right)\right]+t^{0}_{7}\right\}-\left(t^{0}_{2}+t^{0}_{8}\right)}{\left\{t'^{0}_{1}+\left[\alpha^{2}\cdot\left(t'^{0}_{3}+t'^{0}_{5}\right)+(1-\alpha)^{2}\cdot\left(t'^{0}_{4}+t'^{0}_{6}\right)\right]+t'^{0}_{7}\right\}+\left(t'^{0}_{2}+t'^{0}_{8}\right)} \qquad \text{[A-20d]}$$

$$\Delta h \approx 0.98 \qquad \text{[A-20e]}$$

As with expression [A-18c], expression [A-20d] is also a Newton step. Nevertheless, expression [A-20d] is more complicated, especially its denominator which corresponds to the diagonal values of a Hessian matrix. The examples in *Figure A-5* and *Figure A-6* show that as a subnetwork $\boldsymbol{A}_p$ has more routes, the complexity of [A-20d] increases and so does its computation intensity. Due to this increasing complexity and due to the necessity of replacing [A-17e], [A-17f], [A-19i] and [A-19j] by formulas that can automatically be extracted regardless of the complexity of the subnetworks, the second sub-algorithm recurs to three interesting ideas: *approach proportions*, *average approach costs* and Hessian approximations.

An approach proportion, denoted by α_a, is the ratio of the origin-based link flow passing through link a to all the origin-based flow that enters to the tail of link a. It can be easily calculated in a descending topological order by using the following formulas:

$$g_p = \sum_{\forall a\in\boldsymbol{A}_p : a_t = p} f_{p,a} \qquad \text{[A-21a]}$$

$$\alpha_a = \frac{f_{p,a}}{g_{a_h}} \qquad \forall a \in \boldsymbol{A}_p \qquad \text{[A-21b]}$$

$$g_n = d_{(p,n)} + \sum_{\forall a\in\boldsymbol{A} : a_t = n} f_{p,a} \qquad \forall n \in \boldsymbol{N} - \{p\} \qquad \text{[A-21c]}$$

Bar-Gera denominates g_n as *origin-based node flow*. The above formulas guarantee that the following relationships hold.

$$\sum_{\forall a \in \boldsymbol{A}_p : a_h = n} \alpha_a = 1 \qquad \forall n \in \boldsymbol{N} - \{p\} \qquad \text{[A-22a]}$$

$$0 \le \alpha_a \le 1 \qquad \forall a \in \boldsymbol{A}_p \qquad \text{[A-22b]}$$

Now, instead of defining average costs as in [A-17e], [A-17f], [A-19i] or [A-19j], Bar-Gera uses the concept *average cost to node n*, denoted by σ_n. This concept is a function of approach proportions and *average approach cost*, denoted by μ_a. As with approach proportions, the average cost to every node in the restricting subnetwork can be calculated in a topological order as follows:

$$\sigma_p = 0 \qquad \text{[A-23a]}$$

$$\mu_a = \sigma_n + t_a \qquad \forall a \in \boldsymbol{A}_p \qquad \text{[A-23b]}$$

$$\sigma_n = \sum_{\forall a \in \boldsymbol{A}_p : a_t = n} (\alpha_a \cdot \mu_a) \qquad \forall n \in \boldsymbol{N} - \{p\} \qquad \text{[A-23c]}$$

Hessian approximations refer to two auxiliary variables (analogous to μ_a and σ_n), that the algorithm uses to simplify the calculation of the denominator of the Newton step. These two variables are ν_a (approximated derivative of μ_a cost with respect to f_a) and ρ_n (approximated derivative of σ_j with respect to g_j). The following three expressions show how to calculate them in ascending topological order:

$$\rho_p = 0 \qquad \text{[A-24a]}$$

$$\nu_a = \rho_n + t'_a \qquad \forall a \in \boldsymbol{A}_p \qquad \text{[A-24b]}$$

$$\rho_n = \sum_{\forall a \in \boldsymbol{A}_p : a_t = n} (\alpha_a^{\,2} \cdot \nu_a) \qquad \forall n \in \boldsymbol{N} - \{p\} \qquad \text{[A-24c]}$$

Since the denominator of the Newton step could be equal to zero, the second sub-algorithm uses a small value, called ε_v, in order to replace those possible null values and in this way, avoid any division by zero.

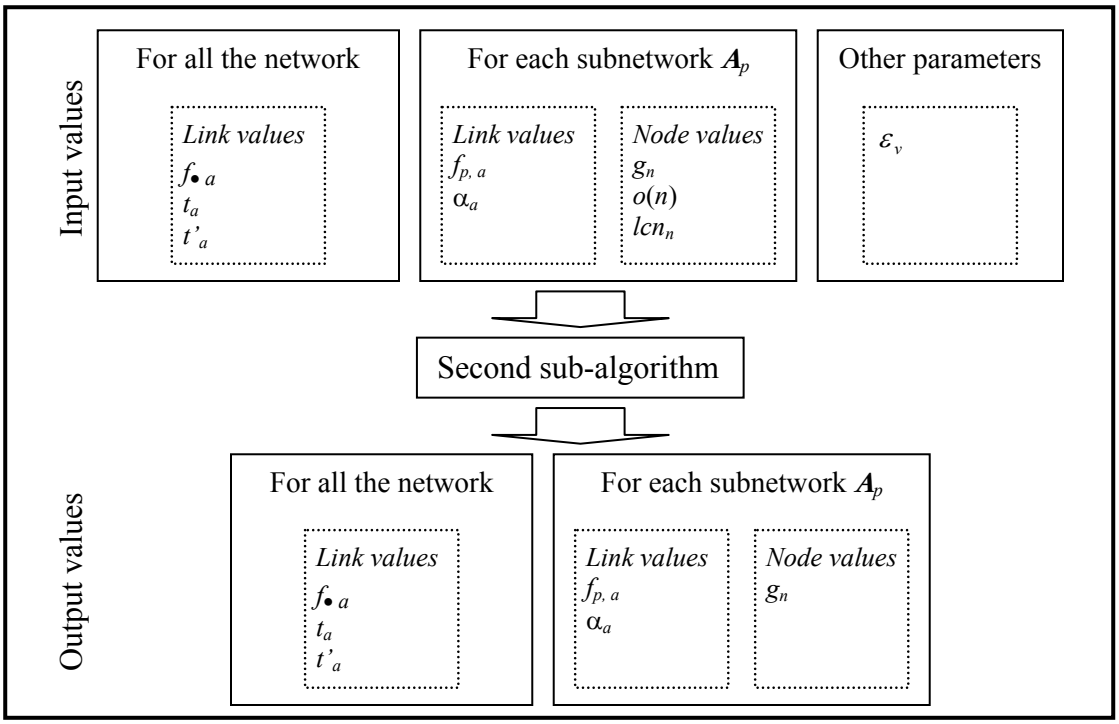


Figure A-7. Input and output values in the second sub-algorithm. Variables regarding the structure of the networks are not shown.

Up to this point, this section has explained most of the variables needed to understand the calculations involved in the second sub-algorithm. *Figure A-7* summarizes the input and output values that the second sub-algorithm uses. One of the input values is the approach proportion α_a of each link *a*. For the first run of the sub-algorithm, the initialization of the whole algorithm has previously assigned an initial value to all the approach proportions included on each subnetwork $\boldsymbol{A}_p$. If the initialization used an all-or-nothing assignment, then this initial value is equal to one. Also, the addition of links to each subnetwork $\boldsymbol{A}_p$ during the first sub-algorithm generate additional variables α_a to consider, but these new inclusions have values equal to zero. *Figure A-8* summarizes how the second sub-algorithm works.

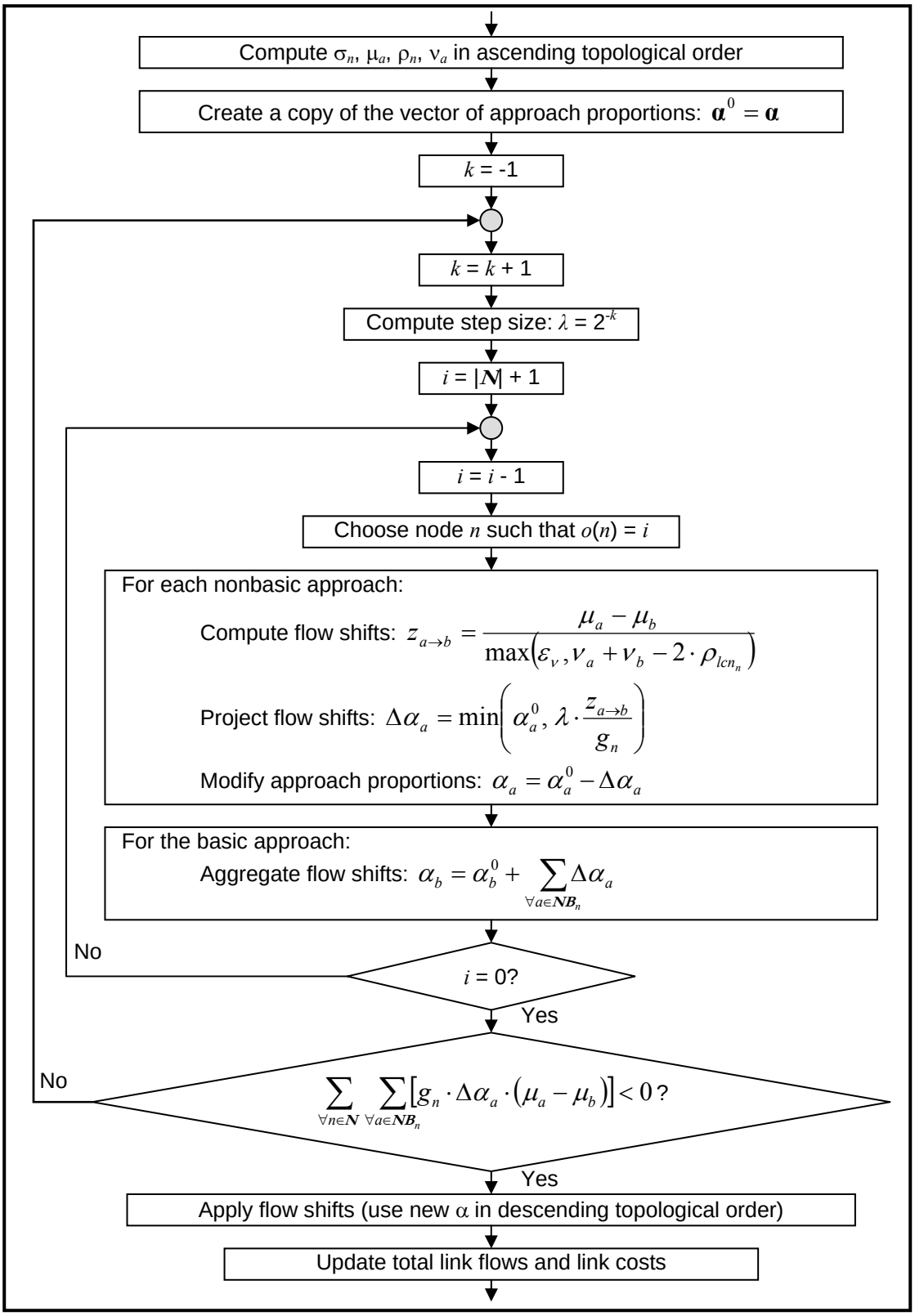


Figure A-8. Second sub-algorithm in Bar-Gera's method.

The following description of the second sub-algorithm has many variations that Bar-Gera leaves to the designer of the code to decide. One of the most important variations has to do to with the order in which the sub-algorithm calculates different variables. For example, *Figure A-8* shows a first step in which the sub-algorithm follows an ascending topological order, then within the loop, a descending topological order and finally another descending topological order. Bar-Gera indicates that for some of these calculations, the sub-algorithm could change from ascending to descending calculations and vice versa, or even simultaneous calculations. The following description corresponds to the specific sub-algorithm shown on *Figure A-8*.

Given a subnetwork $\boldsymbol{A}_p$ and the input values shown on *Figure A-7,* the second sub-algorithm starts by calculating, in a ascending topological order, the average costs and the average approach costs using [A-23] and the Hessian approximations using [A-24]. Then, the sub-algorithm starts the boundary search in which it tries several values of λ in order to find the greatest possible that renders a directional derivative, as defined in [A-15], less than zero. Using the new terminology, this condition becomes as follows:

$$\sum_{\forall n \in \boldsymbol{N}} \sum_{\forall a \in \boldsymbol{NB}_n} \left[g_n \cdot \Delta\alpha_a \cdot \left(\mu_a - \mu_b \right) \right] < 0 \qquad \text{[A-25]}$$

The above expression includes a link *b* and a set $\boldsymbol{NB}_n$. Therefore, understanding [A-25] requires understanding the steps within the loop.

Within each loop, the sub-algorithm scans all the nodes in descending topological order and at each node *n*. For each node *n*, the sub-algorithm scans its incoming links (that is, $\forall a \in \boldsymbol{A}_p : a_h = n$) and it determines the link with the minimum average cost μ_a. This link, Bar-Gera denotes it by *b* and refers to it as the *basic approach to n*. The other links to *n*, that is, the *nonbasic approaches to n*, define the set $\boldsymbol{NB}_n$. For each nonbasic approach to *n,* the sub-algorithm calculates the flow shift $z_{a \to b}$ that needs to be subtracted from the non-basic approach and added to the basic approach *b*. The formula for $z_{a \to b}$ is analogous to expressions [A-18c] and [A-20d] but, unlike those expressions, this formula has the advantage of not changing with the complexity of the subnetwork $\boldsymbol{A}_p$.

$$z_{a \to b} = \frac{\mu_a - \mu_b}{\max\left(\varepsilon_v, \nu_a + \nu_b - 2 \cdot \rho_{lcn_n} \right)} \qquad \text{[A-26]}$$

By dividing $z_{a \to b}$ by g_n, the sub-algorithm obtains the actual Newton step that Bar-Gera's algorithm needs. $z_{a \to b} / g_n$ is dimensionless. A Newton step indicates to the sub-algorithm which direction to take so that it can find a lower value of the objective function *T*. Bar-Gera proposes multiplying this step by the factor λ. For this reason, this factor also receives the name of *step size*. Before assigning $\lambda \cdot \left(z_{a \to b} / g_n \right)$ as the

value to be subtracted from α_a, the algorithm verifies that $\lambda \cdot (z_{a\to b}/g_n)$ is not greater than α_a^0 as follows:

$$\Delta\alpha_a = \min\left(\alpha_a^0, \lambda \cdot \frac{z_{a\to b}}{g_n}\right) \qquad \text{[A-27]}$$

The sub-algorithm subtracts the above value from the nonbasic approaches and adds it to the basic approach. Finally in the loop, the sub-algorithm verifies if condition [A-25] holds.

Once the sub-algorithm finds the optimal λ, it applies flow shifts in a descending topological order. This step consists in obtaining the new origin-based link flows and the origin-based node flows as shown below:

$$g_n = d_{p,n} + \sum_{\forall a \in \boldsymbol{A}_p : a_t = n} f_{a,p} \qquad \forall n : n \in \boldsymbol{N} - \{p\} \text{ [A-28]}$$

$$f_{p,a} = \alpha_a \cdot g_{a_h} \qquad \forall a : a \in \boldsymbol{A}_p \qquad \text{[A-21b']}$$

The final step consists in updating the total link flows f_a (using [A-5a]) and the link costs t_a (using the given link performance function).

When comparing the first sub-algorithm with the second sub-algorithm, Bar-Gera concludes that the former requires more computational time than the latter (for an explanation of this phenomenon, see *Section "Discussion on the Data Structures Recommended for the Implementation"*, *Chapter 2).* Therefore, he adds a modification on the algorithm shown in *Figure A-2* and transforms it into the one shown in *Figure A-5*, by dividing the *cycle* into a *full sub-cycle* and a *quick sub-cycle*. This modification simply guarantees running the second sub-algorithm more times than the first one. He introduces a parameter called *m* or *number of inner iterations*. This thesis will refer to this quick sub-cycles or inner iterations simply as *iterations* because they are analogous to the iterations defined in Aashtiani's algorithm. Before running the algorithm, if the user chooses a very large value for *m*, the computational time will increase considerably. Therefore, the user needs to have a guideline for choosing this value. Bar-Gera's software suggests a value for *m*, but the user has to run the algorithm in order to obtain that suggestion. Nevertheless, as shown in *Chapter 3*, in most the networks tested, the value suggested was always equal to one or to two.

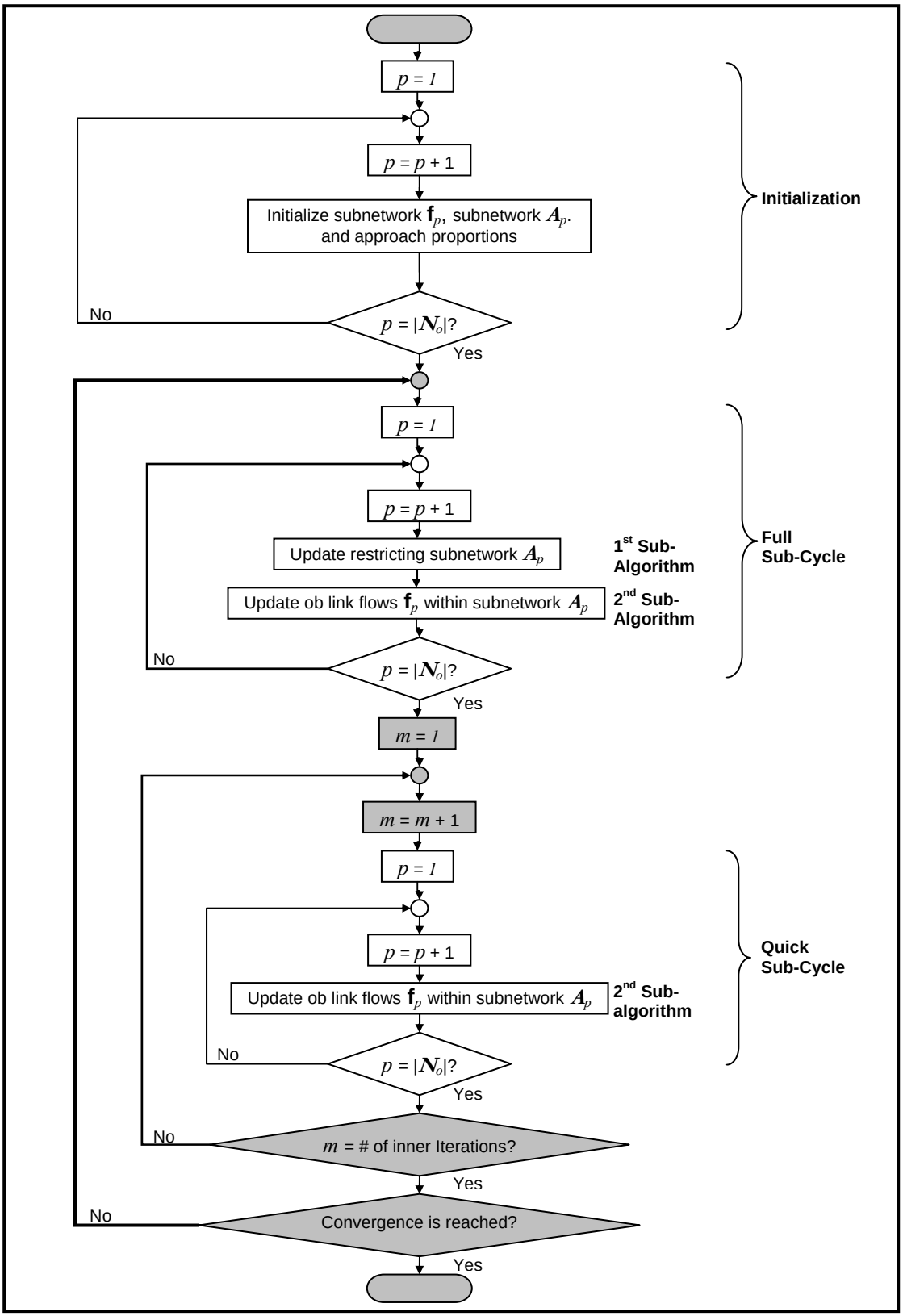


Figure A-5. Bar-Gera's origin-based link algorithm.

Aashtiani's Algorithm

Although Aashtiani (1979) presented an algorithm for different types of TAPs, this thesis adapts it to the S-TAP-F. His algorithm starts with an initial solution **h** which is calculated through the well-known "all-or-nothing" assignment (Sheffi 1985, p. 111). It then decomposes the problem into $|\boldsymbol{I}|$ subproblems, where the solution to each of them is a sub-vector $\mathbf{h}_i$. It then solves each subproblem through a linearization and iterative procedure. Finally, the algorithm verifies whether the group of sub-vectors $\mathbf{h}_i$ construct a final solution **h** that complies with an additional set of conditions. If these final conditions are not met, the algorithm iterates until it reaches a satisfactory **h**. Aashtiani refers to these outer iterations as "cycles". *Figure A-6* depicts the general scheme of his algorithm. The specifics of the general steps shown on *Figure A-6* are as follows.

At *step 2*, the algorithm executes an all-or-nothing assignment. These calculations generate a first route flow h_{1_i} for each OD pair *i*. Therefore, this initial solution **h** contains $|\boldsymbol{I}|$ elements, that is, $\mathbf{h}^{cycle=0} = \left[h_{1_1}\ h_{1_2} \ldots h_{1_{|I|}}\right]$. This step is the first one to require the execution of a one-to-all shortest path algorithm. Aashtiani suggested the use of the Bellman's shortest path algorithm (Bellman 1958) as recommended by Golden (1975).

Analyzing the conditions needed for *step 3* requires understanding first how the algorithm decomposes the problem into $|\boldsymbol{I}|$ subproblems and solves them. *Steps 5* to *7* simply control the scanning of every subproblem *i*. *Step 8* is where the algorithm extracts the sub-vector $\mathbf{h}_i$ from the vector **h**. In principle, this extraction requires taking from **h** all the route flows h_{r_i} such that $r_i \in \boldsymbol{R}_i$. Executing *step 8* in this manner would be inefficient because the cardinality of $\boldsymbol{R}_i$ is usually a very large number. In order to avoid this source of inefficiency, Aashtiani conceived the idea of constructing (and not really "extracting") simpler sub-vectors $\mathbf{h}_i^w$ which would only include so-called "working paths". This approach originates from the observation that the optimal $\mathbf{h}_i$ only contains a small percentage of non-zero elements (the results in the next chapter show that this number is never greater than four). In consequence to this approach, Aashtiani transformed the simple *step 8* into a group of new substeps. For the first cycle, (that is, when $cycle = 1$), he decided that the "set of working paths" would comprise only two routes: the only nonzero-flow route, 1_i that was calculated at *step 2*, and a new route 2_i obtained from executing once **again** the shortest-path algorithm. The algorithm will then assign a zero value to the flow of this additional route, that is, $h_{2_i} = 0$, and will include it in the sub-vector $\mathbf{h}_i^w$. As for the other cycles, (that is, when $cycle = 2, 3, \ldots$), the algorithm (1) executes a new shortest path algorithm, (2) removes any route with zero flow from the set of working paths (the reader will notice in the explanations below that these routes usually do not include the route just added in the previous cycle), and (3) adds the new calculated shortest route to the set of working paths and its corresponding route flow to the sub-

vector $\mathbf{h}_i^w$. In conclusion, the algorithm does not directly manipulate the vector $\mathbf{h}$ at every cycle. Instead, the algorithm only manipulates the sub-vectors $\mathbf{h}_i^w$ in a separately manner all along.

A more algorithmic description of the above sub-steps is as follows (assume that for every OD pair *i*, the algorithm has previously initialized the set $\boldsymbol{R}_i^w$ as $[1_i]$ and the sub-vector $\mathbf{h}_i^w$ as $[h_{1_i}]$ using the shortest routes 1_i calculated at *step 2*):

Sub-step 8.1: For a given origin *p*, execute a one-to-all shortest path algorithm. Refer to every shortest route as r_i^{shortest}.

Sub-step 8.2a: In the first cycle, (that is, when $cycle = 1$) and for every OD pair *i*, include r_i^{shortest} in $\boldsymbol{R}_i^w$ if the following condition is true:

$$\frac{c_{1_i} - c_{r_i^{\text{shortest}}}}{c_{1_i}} > \varepsilon \qquad \text{[A-25a]}$$

(Aashtiani included here a new condition to avoid increasing the number of working paths unnecessarily. This condition uses a parameter ε which usually starts with a big value such as 10^0 but at subsequent cycles, it decreases to values such as 10^{-3} or 10^{-7}. The explanation on *step 3* will revisit this parameter.) If [A-25a] is true, also add a new element h_{2_i} to the sub-vector $\mathbf{h}_i^w$ and assign to this element a value of zero. The algorithm will eventually assign flow to this route 2_i but not at this step.

Sub-step 8.2b: In other cycles, (that is, when $cycle = 2, 3, \ldots$) and for every OD pair *i*, remove from $\mathbf{h}_i^w$ any element h_{r_i} equal to zero. Also, remove the corresponding route from $\boldsymbol{R}_i^w$. Then, calculate the shortest route among the set of working paths and refer to it as u_i. Include route r_i^{shortest} in $\boldsymbol{R}_i^w$ only if the following is true:

$$\frac{u_i - c_{r_i^{\text{shortest}}}}{u_i} > \varepsilon \qquad \text{[A-25b]}$$

Again, if [A-25b] is true, also add a new element h_{r_i} to the vector $\mathbf{h}_i^w$ and assign to this element a value of zero.

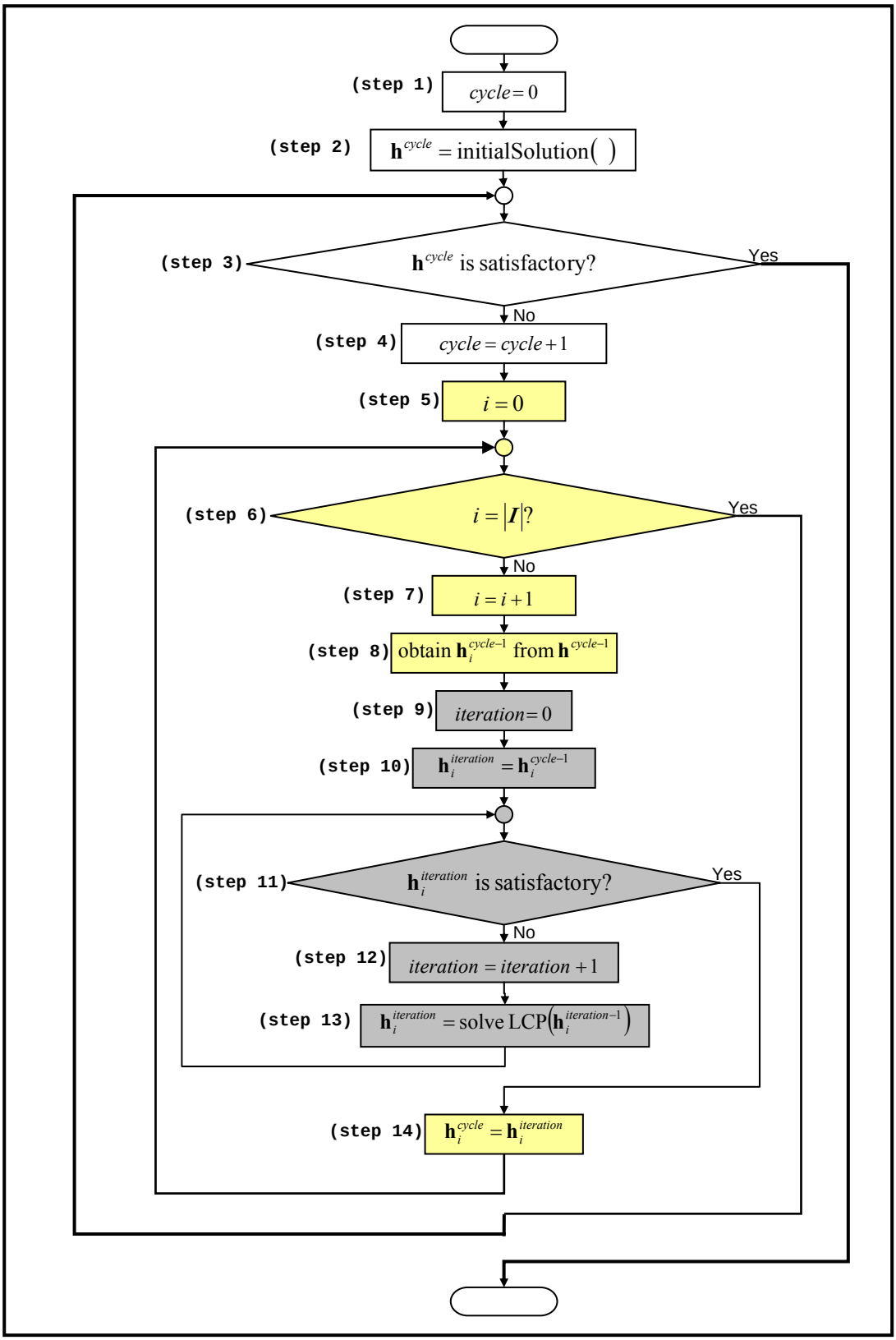


Figure A-6. The basis of Aashtiani's method: decomposition (yellow boxes) and linearization (gray boxes).

Having extracted the initial sub-vector $\mathbf{h}_i^w$ at *step 8*, *steps 9* to *12* control the iterative procedure that will solve the linearized version of the subproblem *i*. Right before executing *step 11*, the algorithm faces the following subproblem *i*:

Find a vector $\mathbf{x} = \mathbf{h}_i^w \mid [u_i] = \left[h_{1_i} \; h_{2_i} \; \ldots \; h_{\left|\boldsymbol{R}_i^w\right|_i} \; u_i \right]$ such that

$$\mathbf{x}\,\mathbf{y}(\mathbf{x})^{\mathrm{T}} = 0 \qquad \text{[A-26a]}$$

$$\mathbf{y}(\mathbf{x}) \geq 0 \qquad \text{[A-26b]}$$

$$\mathbf{x} \geq 0 \qquad \text{[A-26c]}$$

where $\mathbf{y}(\mathbf{x}) = \left[c_{1_i}\left(\mathbf{h}_i^w\right) - u_i \quad c_{2_i}\left(\mathbf{h}_i^w\right) - u_i \; \ldots \; c_{\left|\boldsymbol{R}_i^w\right|_i}\left(\mathbf{h}_i^w\right) - u_i \quad \left(\sum_{\forall r_j \in \boldsymbol{R}_i^w} h_{r_j} \right) - d_i \right]$.

Now, since function $\mathbf{y}$ is differentiable with respect to $\mathbf{x} = \mathbf{h}_i^w \mid [u_i]$, then Aashtiani was able to propose a linearized $\mathbf{y}$ as follows.

$$\mathbf{y}(\mathbf{x})^{\mathrm{T}} \approx \frac{\mathrm{d}\,\mathbf{y}\left(\bar{\mathbf{x}}\right)^{\mathrm{T}}}{\mathrm{d}\,\mathbf{x}} \left[\mathbf{x}^{\mathrm{T}} - \bar{\mathbf{x}}^{\mathrm{T}} \right] + \mathbf{y}\left(\bar{\mathbf{x}}\right)^{\mathrm{T}} =$$

$$\begin{bmatrix}
\frac{\partial y_{1_i}(\bar{\mathbf{x}})}{\partial x_{1_i}} & \frac{\partial y_{1_i}(\bar{\mathbf{x}})}{\partial x_{2_i}} & \cdots & \frac{\partial y_{1_i}(\bar{\mathbf{x}})}{\partial x_{\left|\boldsymbol{R}_i^w\right|_i}} & \frac{\partial y_{1_i}(\bar{\mathbf{x}})}{\partial x_{\left|\boldsymbol{R}_i^w\right|+1_i}} \\
\frac{\partial y_{2_i}(\bar{\mathbf{x}})}{\partial x_{1_i}} & \frac{\partial y_{2_i}(\bar{\mathbf{x}})}{\partial x_{2_i}} & \cdots & \frac{\partial y_{2_i}(\bar{\mathbf{x}})}{\partial x_{\left|\boldsymbol{R}_i^w\right|_i}} & \frac{\partial y_{2_i}(\bar{\mathbf{x}})}{\partial x_{\left|\boldsymbol{R}_i^w\right|+1_i}} \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
\frac{\partial y_{\left|\boldsymbol{R}_i^w\right|_i}(\bar{\mathbf{x}})}{\partial x_{1_i}} & \frac{\partial y_{\left|\boldsymbol{R}_i^w\right|_i}(\bar{\mathbf{x}})}{\partial x_{2_i}} & \cdots & \frac{\partial y_{\left|\boldsymbol{R}_i^w\right|_i}(\bar{\mathbf{x}})}{\partial x_{\left|\boldsymbol{R}_i^w\right|_i}} & \frac{\partial y_{\left|\boldsymbol{R}_i^w\right|_i}(\bar{\mathbf{x}})}{\partial x_{\left|\boldsymbol{R}_i^w\right|+1_i}} \\
\frac{\partial y_{\left|\boldsymbol{R}_i^w\right|+1_i}(\bar{\mathbf{x}})}{\partial x_{1_i}} & \frac{\partial y_{\left|\boldsymbol{R}_i^w\right|+1_i}(\bar{\mathbf{x}})}{\partial x_{2_i}} & \cdots & \frac{\partial y_{\left|\boldsymbol{R}_i^w\right|+1_i}(\bar{\mathbf{x}})}{\partial x_{\left|\boldsymbol{R}_i^w\right|_i}} & \frac{\partial y_{\left|\boldsymbol{R}_i^w\right|+1_i}(\bar{\mathbf{x}})}{\partial x_{\left|\boldsymbol{R}_i^w\right|+1_i}}
\end{bmatrix}
\begin{bmatrix}
x_{1_i} - \bar{x}_{1_i} \\
x_{2_i} - \bar{x}_{2_i} \\
\cdots \\
x_{\left|\boldsymbol{R}_i^w\right|_i} - \bar{x}_{\left|\boldsymbol{R}_i^w\right|_i} \\
x_{\left|\boldsymbol{R}_i^w\right|+1_i} - \bar{x}_{\left|\boldsymbol{R}_i^w\right|+1_i}
\end{bmatrix}
+
\begin{bmatrix}
y_{1_i}(\bar{\mathbf{x}}) \\
y_{2_i}(\bar{\mathbf{x}}) \\
\cdots \\
y_{\left|\boldsymbol{R}_i^w\right|_i}(\bar{\mathbf{x}}) \\
y_{\left|\boldsymbol{R}_i^w\right|+1_i}(\bar{\mathbf{x}})
\end{bmatrix} \qquad [A\text{-}27]$$

where $\bar{\mathbf{x}}$ refers to an initial value of sub-vector $\mathbf{x}$.

Differentiating $\mathbf{y}$ partially with respect to u_i is straightforward, but differentiating $\mathbf{y}$ partially with respect to $\mathbf{h}_i^w$ requires taking into account (the definition of route cost [A-6], and) the now simplified definition of total link flow:

$$f_{\bullet a}\left(\mathbf{h}_i^w\right) = \underbrace{\sum_{\forall r_i \neq \boldsymbol{R}_i} \left(h_{r_i} \cdot \delta_{ar_i}\right)}_{\text{constant}} + \sum_{\forall r_i \in \boldsymbol{R}_i^w} \left(h_{r_i} \cdot \delta_{ar_i}\right) \qquad \text{[A-28]}$$

The calculation of $f_{\bullet a}$ becomes simplified because the routes $r_i \in \left(\boldsymbol{R}_i - \boldsymbol{R}_i^w\right)$ have zero flow and therefore, do not need to be part of the calculation. Also, the constant term in [A-28] facilitates the differentiation. With these considerations on how to linearize **y,** the resulting linear complementarity problem is as follows:

Find a vector $\mathbf{x} = \mathbf{h}_i^w \mid [u_i] = \left[h_{1_i} \; h_{2_i} \; \ldots \; h_{\left|\boldsymbol{R}_i^w\right|_i} \; u_i \right]$ such that

$$\mathbf{x}\, \mathbf{y}(\mathbf{x})^{\mathrm{T}} = 0 \qquad \text{[A-29a]}$$

$$\mathbf{y}(\mathbf{x}) \geq 0 \qquad \text{[A-29b]}$$

$$\mathbf{x} \geq 0 \qquad \text{[A-29c]}$$

where $\mathbf{y}(\mathbf{x})^{\mathrm{T}} =$

$$\begin{bmatrix} \frac{\partial c_{1_i}\left(\overline{\mathbf{h}_i^w}\right)}{\partial h_{1_i}} & \frac{\partial c_{1_i}\left(\overline{\mathbf{h}_i^w}\right)}{\partial h_{2_i}} & \cdots & \frac{\partial c_{1_i}\left(\overline{\mathbf{h}_i^w}\right)}{\partial h_{\left|\boldsymbol{R}_i^w\right|_i}} & -1 \\ \frac{\partial c_{2_i}\left(\overline{\mathbf{h}_i^w}\right)}{\partial h_{1_i}} & \frac{\partial c_{2_i}\left(\overline{\mathbf{h}_i^w}\right)}{\partial h_{2_i}} & \cdots & \frac{\partial c_{2_i}\left(\overline{\mathbf{h}_i^w}\right)}{\partial h_{\left|\boldsymbol{R}_i^w\right|_i}} & -1 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial c_{\left|\boldsymbol{R}_i^w\right|_i}\left(\overline{\mathbf{h}_i^w}\right)}{\partial h_{1_i}} & \frac{\partial c_{\left|\boldsymbol{R}_i^w\right|_i}\left(\overline{\mathbf{h}_i^w}\right)}{\partial h_{2_i}} & \cdots & \frac{\partial c_{\left|\boldsymbol{R}_i^w\right|_i}\left(\overline{\mathbf{h}_i^w}\right)}{\partial h_{\left|\boldsymbol{R}_i^w\right|_i}} & -1 \\ 1 & 1 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} h_{1_i} \\ h_{2_i} \\ \cdots \\ h_{\left|\boldsymbol{R}_i^w\right|_i} \\ u_i \end{bmatrix} + \begin{bmatrix} c_{1_i}\left(\overline{\mathbf{h}_i^w}\right) - \sum_{r_i \in \boldsymbol{R}_i^w} h_{r_i} \frac{\partial c_{1_i}\left(\overline{\mathbf{h}_i^w}\right)}{\partial h_{r_i}} \\ c_{2_i}\left(\overline{\mathbf{h}_i^w}\right) - \sum_{r_i \in \boldsymbol{R}_i^w} h_{r_i} \frac{\partial c_{2_i}\left(\overline{\mathbf{h}_i^w}\right)}{\partial h_{r_i}} \\ c_{\left|\boldsymbol{R}_i^w\right|_i}\left(\overline{\mathbf{h}_i^w}\right) - \sum_{r_i \in \boldsymbol{R}_i^w} h_{r_i} \frac{\partial c_{\left|\boldsymbol{R}_i^w\right|_i}\left(\overline{\mathbf{h}_i^w}\right)}{\partial h_{r_i}} \\ -d_i \end{bmatrix}$$

,

$$\frac{\partial c_{r_i}\left(\overline{\mathbf{h}_i^w}\right)}{\partial h_{r'_i}} = \sum_{\forall a \in \boldsymbol{A}} \left\{ \delta_{a r_i} \cdot \delta_{a r'_i} \cdot \frac{\mathrm{d}t_a\left(f_{a\bullet}\right)}{\mathrm{d}\, f_{a\bullet}} \right\},$$

and $\overline{\mathbf{h}_i^w}$ is the initial value of $\mathbf{h}_i^{\mathrm{w}}$.

Now, using the linear complementarity problem as formulated above in [A-29] and plugging the sub-vector $\mathbf{h}_i^w$ obtained at *step 10* as the $\overline{\mathbf{h}_i^w}$, the algorithm solves the subproblem *i* at *step 13* using, as suggested by Aashtiani, Lemke's algorithm (Lemke 1965). After executing *step 13*, the algorithm iterates until, as required by Wardrop's algorithm, the difference between the longest and the shortest route is zero. Since in practice, solving the LCP does not render a solution with a difference exactly equal to zero, Aashtiani's recurs again to the use of parameter ε. Therefore, the iterations stop when the following condition becomes true:

$$\frac{\max_{r_i \in \boldsymbol{R}_i^w}(c_{r_i}) - u_i}{\max_{r_i \in \boldsymbol{R}_i^w}(c_{r_i})} \leq \varepsilon \qquad \text{[A-30]}$$

The reader can now see that Aashtiani uses the parameter ε for two purposes: (1) to include new routes in the set of shortest paths and (2) to reach a satisfactory solution $\mathbf{h}_i^w$. Parameter ε is therefore how the algorithm controls the precision of the solution. After finding a satisfactory $\mathbf{h}_i^w$ at *step 11*, the algorithm continues with the next subproblem.

When at *step 6*, the algorithm recognizes that it has scanned all the subproblems *i*, then, at *step 3*, it evaluates solution **h** as a whole. Two conditions render a solution **h** satisfactory. The first condition is that after having scanned all the subproblems *i*, there has to be a complete cycle in which the algorithm did not carry out any linearizations. This condition is important because every linearization, although it allows obtaining an optimal solution to a subproblem *i*, it may alter the solutions of the other subproblems. The second and final condition has to do with the parameter ε. The algorithm sets it to a high value at the beginning of the algorithm (*step 3*) such as 10^0. Then, once a cycle complies with the first condition, the algorithm reduces the value of parameter ε by eighty or ninety percent and uses it for the next cycles. Finally, when the algorithm reaches the desired value of ε such as 10^7, 10^{10} or 10^{14}, the algorithm ends.

It is important to highlight one aspect concerning how ε is reduced. An algorithm in which ε is reduced by 90 percent orders the next cycle to generate a solution **f.** with a higher precision than an algorithm where ε is reduced by 20 percent. Therefore, it is reasonable to suspect that the former will execute more iterations for each cycle than the latter. This could be seen as detrimental. But the reader should remember that, since every cycle requires the execution of a new shortest path sub-algorithm for every origin, then choosing the former algorithm reduces the number of shortest path sub-algorithms to execute. In Figure A-17, we are reducing ε by 90 percent since we are dividing it by 10. If instead, we say that we are going to divide it by m_A, then the above phenomenon could be restated as follows: An increase in m_A intensifies local search while a decrease in m_A intensifies the global search. Here we are following the specific manner in which Toobaie (1998) decreased ε. Originally, Aashtiani (1979) suggested a different but similar formula:

$$\varepsilon_n = \delta^n \cdot \varepsilon \qquad \text{[A-31]}$$

where

ε_n is the new ε, δ could have a value of 10, and n would start with a value of, say, one, and would increase by one at every cycle.

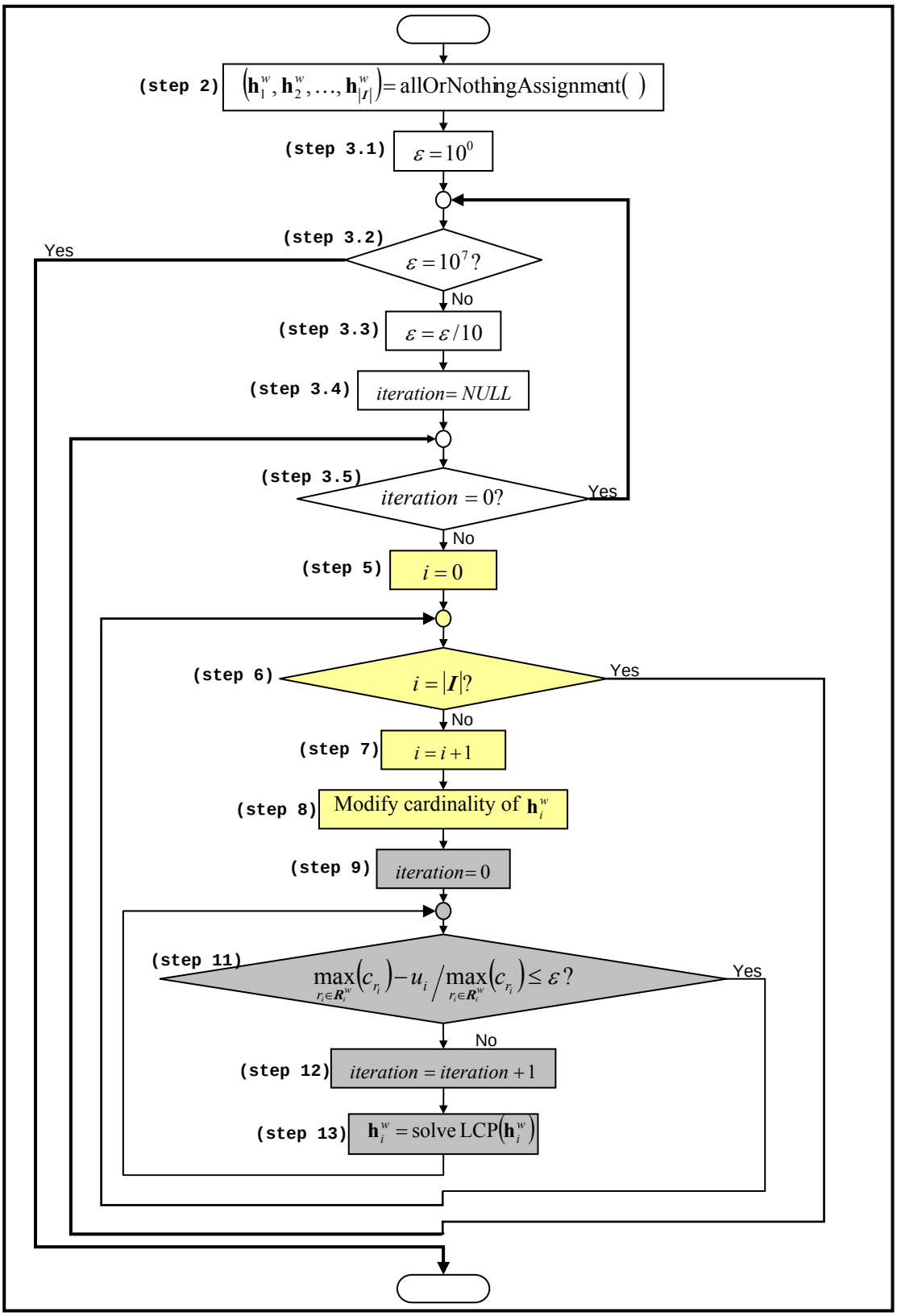


Figure A-7. Aashtiani's algorithm in detail: decomposition (yellow boxes) and linearization (gray boxes).

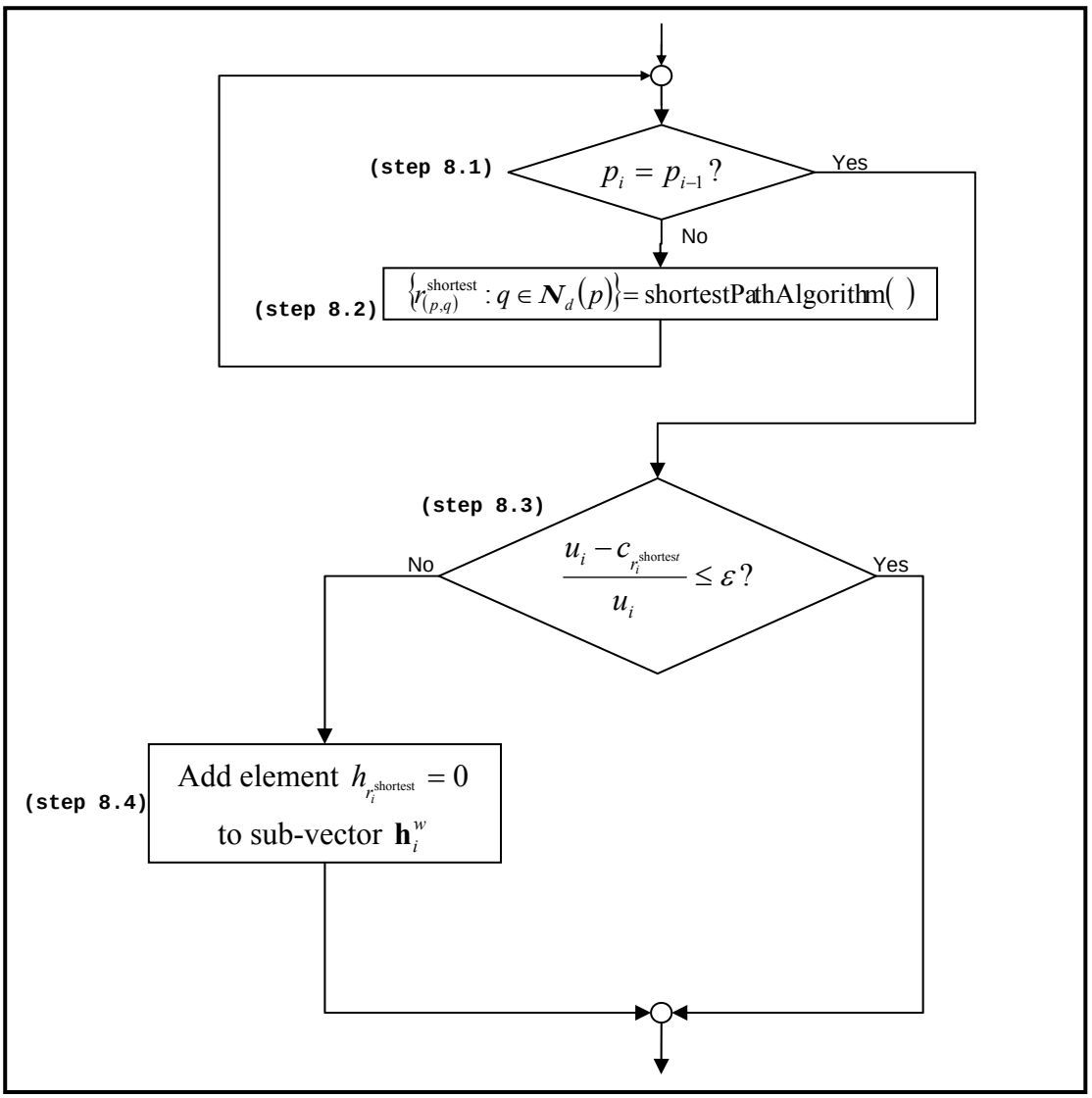


Figure A-8. Aashtiani's algorithm: *step 8* in detail. p_i refers to the origin node of OD pair i

Taking into account all of the above considerations, Aashtiani's algorithm is as described on *Figures A-7* and *A-8*. In *Figure A-7*, *step 10* and *step 14* became irrelevant due to the direct manipulation of sub-vectors $\mathbf{h}_i^w$. The use of superindices to indicate whether a solution belongs to a specific cycle or iteration also became irrelevant. *Steps 1* and *4* became irrelevant as well. *Figure A-7* shows a specific example where the parameter ε starts with a value of 10^0 and then it decreases by 10 until the algorithm achieves the value of 10^7.

In *Figure A-8* p_i refers to the origin node of OD pair *i*. The execution of *steps 8.1* and *8.2* imply that the algorithm becomes faster if the OD pairs are already (previous to the execution of the algorithm) grouped by origin.

BIBLIOGRAPHY

Aashtiani, Hedayat Zokaei. 1979. The multi-modal traffic assignment problem. (Ph.D. dissertation). Massachusetts Institute of Technology, Cambridge, MA.

Asmuth, R. 1978. *Traffic Network Equilibrium*. Technical Report SOL-78-2, Stanford University, Stanford, CA.

Bar-Gera, Hillel. 1999. *Origin-based algorithms for transportation network modeling*. Research Triangle Park, NC, U.S.A.: National Institute of Statistical Sciences.

---. 2002. Origin-Based Algorithm for the Traffic Assignment Problem. *Transportation Science* 36, no. 4: 398-417.

---. 2008. Transportation Test Problems. http://www.bgu.ac.il/~bargera/tntp/ (last accessed April 4th 2009).

Beckman, Martin, C. B. McGuire, and Christopher B. Winsten. 1956. *Studies in the Economics of Transportation*. New Haven, CT, U.S.A.: Yale University Press.

Bellman, R. 1958. On a routing problem. *Quart. Applied Math* 16: 87-90.

Boyce, David, Biljana Ralevic-Dekic, and Hillel Bar-Gera. 2004. Convergence of Traffic Assignments: How Much is Enough? *Journal of Transportation Engineering* 130, no. 1 (January): 49-55.

Boyce, David E., Hani S. Mahmassani, and Anna Nagurney. 2005. A retrospective on Beckmann, McGuire and Winsten's Studies in the Economics of Transportation. *Papers in Regional Science* 84, no. 1: 85-103.

Bureau of Public Roads. 1964. Traffic Assignment Manual. *US Department of Commerce*.

Dafermos, S. C. 1968. Traffic Assignment and Resource Allocation in Transportation Networks. Johns Hopkins.

---. 1980. Traffic equilibrium and variational inequalities. *Transportation Science* 14, no. 1: 42-54.

Dafermos, S. C., and F. T. Sparrow. 1969. The traffic assignment problem for a general network. *Journal of Research of the National Bureau of Standards, Series B* 73, no. 2: 91-118.

Frank, M., and P. Wolfe. 1956. An algorithm for quadratic programming. *Naval Research Logistics Quarterly* 3, no. 1-2: 95-110.

Golden, Bruce. 1975. Shortest-Path Algorithm: A Comparison. *Operations Research* 24, no. 6: 1164-1168.

Holmberg, Kaj, and Di Yuan. 2003. A Multicommodity Network-Flow Problem with Side Constraints on Paths Solved by Column Generation. *INFORMS Journal on Computing* 15, no. 1: 42.

INRO. 2008. EMME/3. http://www.inro.ca/en/products/emme/index.php (last accessed April 4th 2009).

Jahn, Olaf, Rolf H. Möhring, Andreas S. Schulz, and Nicolás E. Stier-Moses. 2005. System-Optimal Routing of Traffic Flows with User Constraints in Networks with Congestion. *Operations Research* 53, no. 4: 600-616.

LeBlanc, L. J., E. K. Morlok, and W. P. Pierskalla. 1975. An efficient approach to solving the road network equilibrium traffic assignment problem. *Transportation Research* 9, no. 3: 308-318.

Lemke, C. E. 1965. Bimatrix equilibrium points and mathematical programming. *Management Science* 11, no. 7: 681-689.

Nguyen, S. 1976. A unified approach to equilibrium methods for traffic assignment. *Traffic Equilibrium Methods, Lecture Notes in Economics and Mathematical Systems* 18: 148-182.

Nguyen, S., and L. James. 1975. *TRAFFIC - An Equilibrium Traffic Assignment Program*. Report. Montréal, Canada: Centre de Recherche sur les Transports, Université de Montréal.

Pallottino, S., and M. G. Scutella. 1998. Shortest Path Algorithms in Transportation Models: Classical and Innovative Aspects. *Equilibrium and Advanced Transportation Modeling*.

Pape, U. 1974. Implementation and efficiency of Moore-algorithms for the shortest route problem. *Mathematical Programming* 7, no. 1: 212-222.

Patriksson, Michael. 1994. *The Traffic Assignment Problem: Models and Methods*. Utrecht, The Netherlands.

Prager, W. 1954. Problems of traffic and transportation. In *Proceedings of the Symposium on Operations Research in Business and Industry*, 105–113. Cambridge, MA: Candlewick Press.

Ruiter, E. R. 1974. Implementation of Operational Network Equilibrium Procedures. *Transportation Research Record*, no. 491: 40-51.

Sender, J. G., and M. Netter. 1970. *Équilibre offre-demande et tarification sur un réseau de transport*. Arcueil, France: Département Économie, Institut de Recherche des Transports.

Sheffi, Yosef. 1985. *Urban Transportation Networks: Equilibrium Analysis With Mathematical Programming Methods*. Prentice Hall.

Smith, M. J. 1979. The existence, uniqueness and stability of traffic equilibria. *Transportation Research* 13, no. 3: 295-304.

Toobaie, Shahabeddin. 1998. Improvements on the Nonlinear Complementarity Problem. Sharif University of Technology, Tehran, Iran.

Wardrop, John Glenn. 1952. Some Theoretical Aspects of Road Traffic Research. In *Proceedings of the Institute of Civil Engineers*, 1:325-362. Greenford, Middlesex, UK.