Available online at www.sciencedirect.com

ScienceDirect

Transportation Research Procedia 27 (2017) 993–1000

Transportation Research Procedia

www.elsevier.com/locate/procedia

20th EURO Working Group on Transportation Meeting, EWGT 2017, 4-6 September 2017, Budapest, Hungary

Calibration of the demand structure for dynamic traffic assignment using flow and speed data: exploiting the advantage of distributed computing in derivative-free optimization algorithms

Bojan Kostic [a]*, Lorenzo Meschini [b], Guido Gentile [a]

[a] *Sapienza University of Rome, Via Eudossiana 18, Rome 00184, Italy*
[b] *SISTeMA PTV Group, Via Bonghi 11, Rome 00184, Italy*

Abstract

Stochastic optimization algorithms have been used in the recent literature as a preferred way for calibrating Dynamic Traffic Assignment (DTA) models, as the computation of explicit gradients is numerically too cumbersome on real networks. However, early experiences based on the Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm have shown performance issues when the number of variables becomes large. This suggests to focus on structural demand variables rather than to consider all components of origin-destination (O-D) matrices. Moreover, with the possibility of distributed computing, many algorithms that where not efficient in a standard configuration (i.e. sequential objective function evaluations within each iteration) can become a viable alternative to SPSA. For example, parallelization can be especially beneficial for genetic algorithms, which require a large number of independent function evaluations per iteration. In this paper we examine several optimization algorithms applied to dynamic demand calibration using flow and speed field measurements. The problem is to minimize the distance between results of a dynamic network loading and traffic data observed on road links. This approach is investigated in the context of laboratory experiments, where known O-D matrices are perturbed after its dynamic assignment on the network, to prove the effectiveness of the proposed methodology.

Peer-review under responsibility of the scientific committee of the 20th EURO Working Group on Transportation Meeting.

Keywords: Dynamic Traffic Assignment; demand calibration; optimization; derivative-free algorithms; distributed computing.

* Corresponding author. Tel.: +39-06-445-85737; fax: +39-06-445-85129.
E-mail address: bojan.kostic@uniroma1.it

2352-1465
Peer-review under responsibility of the scientific committee of the 20th EURO Working Group on Transportation Meeting.
10.1016/j.trpro.2017.12.041

1. Introduction

A simulation-based Dynamic Traffic Assignment (DTA) has been increasingly used in practice, as models able to represent traffic dynamics are becoming more available (Barceló and Casas, 2006; Ben-Akiva et al., 2001; Gentile and Meschini, 2011; Tampère et al., 2010). These models are capable of realistically representing traffic phenomena, such as queues and spillback, as they effectively capture the dependencies of a complex dynamic system. Unlike in static assignment models, where congestion on links is typically represented by a simple function between flow and travel time, in macroscopic DTA models this relation becomes non-separable in both space and time (Gentile, 2015).

In order to use DTA models with real-world networks, both off-line and on-line, many variables constituting the demand side and the supply side (including the route choice) have to be calibrated (Antoniou et al., 2011). Demand, as an essential input to a DTA model, needs to be properly adjusted to represent mobility levels of the study area in the precise manner. However, in the origin-destination (O-D) demand calibration problem, various authors proved that there is no unique solution as the problem is underdetermined. In other words, in a system of equations describing the problem, there are many more O-D demand flows with respect to the number of observations (Cascetta, 2009; Marzano et al., 2009). In addition, existing methodologies for demand calibration for static assignment typically rely on the linear relation between O-D flows and link flows due to the separability of link cost functions, which is assumed through the assignment matrix (Cascetta, 2009). Therefore, it is difficult to apply these approaches to dynamic cases, where the problem becomes increasingly non-linear because of congestion dynamics; hence, the assumption of linear relation is erroneous (Frederix et al., 2013). This proves that the calibration methodologies applied to static assignment calibration cannot effectively be transferred in dynamic context, thus making the dynamic O-D demand calibration problem much more challenging to solve.

1.1. Motivation

Initially, the research was focused on the static O-D matrix calibration. Recently, with the expanding dynamic traffic models and their practical implementations, the research tackling the dynamic aspect of demand calibration goes beyond synthetic research environments, as it has become needed for practitioners. However, beside extensive literature, the proposed methods fail to provide meaningful results when dealing with large-scale networks. Most of the test cases use small-scale networks, often with no real data. Increase in network size, which assumes increase in the problem dimensionality, more complex supply-demand interactions, dynamic route choices, time-dependent and elastic demand, makes the problem more complex, i.e. assumes non-convex objective function with non-linear relations among decision variables. Distributed computing offers many advantages over standard computing, especially when solving heavy problems, such as the demand calibration problem. With distributed computing, requirements for longer heavy computations can be significantly decreased.

1.2. Related work

The literature on the topic of demand calibration is quite extensive. Initial research was mainly focused on static demand calibration, for which an overview can be found in Cascetta (2009). With a development of DTA models, recent research, naturally, concentrates on the calibration of dynamic demand. A brief overview of dynamic demand calibration methods, for both off-line and on-line cases, can be found in Antoniou et al. (2011). To evaluate and compare the effectiveness of various methods, a study on benchmarking different methods has recently been conducted (Antoniou et al., 2016), where benchmarking cases and procedures were established. As our focus is on dynamic demand calibration, we examine only relative work in this area.

In the recent literature, stochastic iterative algorithms that do not require explicit formulation of the relationship between O-D flows and link flows are typically used. To formulate the problem, demand calibration in DTA is regarded as an optimization problem. To solve it, the most widely used algorithm in the literature is the Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm (Spall, 1998, 1992), using single O-D pairs as calibration variables. However, to improve its performance, various authors proposed modifications that increase the number of function evaluations per iteration, thus alleviating its main advantage. Having a robust and efficient optimization algorithm can have a significant impact on the calibration outcome. With a development of more sophisticated

simulation software allowing for the possibility of distributed computing, genetic and evolutionary algorithms present themselves not only as a viable alternative, but also, with no doubt, providing superior efficiency. Kostic and Gentile (2015) tested several optimization algorithms besides SPSA, such as the Covariance Matrix Adaptation – Evolution Strategy (CMA-ES) algorithm (Hansen, 2011, 2006), which was the first time it was applied in this context. In addition, they also tested the Nelder-Mead's Simplex algorithm (Nelder and Mead, 1965), and provided their comparison.

1.3. Paper Structure

The paper is organized as follows: Section 2 introduces problem formulation and solution algorithms. Section 3 discusses how the optimization algorithms can be made more efficient using distributed computing. Section 4 presents numerical tests. Finally, Section 5 contains concluding remarks.

2. Problem formulation and solution algorithms

The calibration problem is formulated as an optimization problem. It is a constrained minimization problem given by (1):

$$\min f(\boldsymbol{\theta}) = w_x \left\|\hat{\mathbf{x}}-\bar{\mathbf{x}}\right\| + w_q \left\|\hat{\mathbf{q}}-\bar{\mathbf{q}}\right\| + w_v \left\|\hat{\mathbf{v}}-\bar{\mathbf{v}}\right\| \tag{1}$$

subject to $\mathbf{x}^{\text{lb}} \le \mathbf{x} \le \mathbf{x}^{\text{ub}}$, where w represent weights, $\hat{\mathbf{x}}$ is the initial demand vector (historical matrices), $\bar{\mathbf{x}}$ is the current demand vector (estimated/assigned demand), $\hat{\mathbf{q}}$ and $\bar{\mathbf{q}}$ represent simulated and observed flows respectively, and $\hat{\mathbf{v}}$ and $\bar{\mathbf{v}}$ are simulated and observed speeds respectively. Distance function used is Normalized Root Mean Square Error (*RMSN*).

$$RMSN = \frac{\sqrt{n\sum_{i=1}^{n}(f_i - y_i)^2}}{\sum_{i=1}^{n} y_i} \tag{2}$$

To solve the formulated optimization problem, we tested three optimization algorithms. They are described below.

The standard first-order SPSA algorithm can be very efficient as it can do only two function evaluations per iteration with the simplest configuration. They are used to approximate the gradient and obtain an updated point. Gradient is calculated as follows: where is a vector of variables' values for calibration at iteration k, $\boldsymbol{\Delta}_k$ is a Bernoulli-distributed ±1 random variable at iteration k, and c_k represents a gain sequence responsible for sampling two points for function evaluation used for gradient approximation and is given by: where c and γ are algorithm coefficients. New point is then obtained using the step size and previously computed gradient: where a_k is a gain sequence responsible for the step size, i.e. the advancement in the direction of the gradient and is given by:

Algorithm 1 Simultaneous Perturbation Stochastic Approximation (SPSA)

1: f_0, $\mathbf{x}_0$, $\mathbf{x}^{ub}$, $\mathbf{x}^{lb}$	*// starting point and bounds*
2: a, α, A, c, γ, n_g; n_k, n_l, t_{min}, f_{min}	*// algorithm parameters; termination criteria*
3: $f^* \leftarrow f_0$, $\mathbf{x}^* \leftarrow \mathbf{x}_0$, $k \leftarrow 0$, $l \leftarrow 0$	*// initialize: best OF value; best parameters; counters*
4: **loop until termination criterion fulfilled** (n_k, n_l, t_{min}, f_{min})	*// main loop*
5: $k \leftarrow k + 1$	*// increment iteration counter*
6: $a_k \leftarrow a / (A + k)^{\alpha}$, $c_k \leftarrow c / k^{\gamma}$	*// update gain sequences a_k and c_k*
7: **for** $i = 1 \rightarrow n_g$ **do**	*// for each gradient replication*

Algorithm 1 Simultaneous Perturbation Stochastic Approximation (SPSA)

8:	$\mathbf{r} \sim U(\mathbf{0}, \mathbf{I})$; $\Delta_k^i = 2\|\mathbf{r}\| - 1$	*// Bernoulli ±1 outcome*
9:	$f^+_{ki} = y(\mathbf{x}_k + c_k\Delta_k^i)$, $f^-_{ki} = y(\mathbf{x}_k - c_k\Delta_k^i)$, $l \leftarrow l + 2$	*// evaluate points in '+' and '−' directions*
10:	$\hat{\mathbf{g}}_k^i(\mathbf{x}_k^i) = (f^+_{ki} - f^-_{ki}) / 2c_k \cdot [1/\Delta^i_{k1} \quad 1/\Delta^i_{k2} \quad \cdots \quad 1/\Delta^i_{k\|\mathbf{x}\|}]^T$	*// update gradient*
11:	**end for**	
12:	$\hat{\mathbf{g}}_k = \sum \hat{\mathbf{g}}_k^i / n_g$, $i = 1 \rightarrow n_g$	*// gradient averaging*
13:	$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k - a_k\hat{\mathbf{g}}_k(\mathbf{x}_k)$	*// update parameters' estimate*
14:	$f^* \leftarrow \min(f^*, f^+_{ki}, f^-_{ki} \mid i = 1 \rightarrow n_g)$, $\mathbf{x}^* \leftarrow \mathbf{x}(f^*)$	*// update best objective function value and parameters*
15:	**end loop**	*// end*

Nelder-Mead's Simplex algorithm (NMSIM) is a well-known algorithm created by Nelder and Mead (1965). It is known for its fast convergence with linear problems, whereas with non-linear problems it provides limited performance. It was used by Kostic and Gentile (2015) in the context of demand calibration.

The simplex, created at the beginning of the optimization, is gradually re-shaped trying to move toward a minimum objective function value by using geometric transformations. The starting simplex is eventually shrank toward the final solution. It generates $||\mathbf{x}_0||+1$ points ($\mathbf{x}_1$, $\mathbf{x}_2$, … , $\mathbf{x}_{||\mathbf{x}_0||+1}$), where each point is a vector of calibration parameters $\mathbf{x}_i^T = [x_1^i \quad x_2^i \quad \cdots \quad x_{||\mathbf{x}_0||}^i]$. To create the simplex (i.e. an ordered list), the points are evaluated independently ($f(\mathbf{x}_1)$, $f(\mathbf{x}_2)$, … , $f(\mathbf{x}_{||\mathbf{x}_0||+1})$) and the simplex is created $\mathbf{f}^T = [f_1 \quad f_2 \quad \cdots \quad f_{||\mathbf{x}_0||+1}]$ where $f_1 \le f_2 \le \cdots \le f_{||\mathbf{x}_0||+1}$. At each iteration, point(s) are sequentially evaluated and based on its logic, it chooses a direction and step size. This way it explores different directions and changes/adopts the simplex shape according to the newly evaluated points. The algorithm is described in detail in Nelder and Mead (1965) and is presented below.

Algorithm 2 Nelder-Mead's Simplex algorithm (NMSIM)

1:	f_0, $\mathbf{x}_0$, $\mathbf{x}^{ub}$, $\mathbf{x}^{lb}$	*// starting point and bounds*
2:	α, γ, ρ, σ, n_k, n_l, t_{min}, f_{min}	*// algorithm coefficients; termination criteria*
3:	$f^* \leftarrow f_0$; $\mathbf{x}^* \leftarrow \mathbf{x}_0$; $\mathbf{x}_1 \leftarrow \mathbf{x}_0$; $k \leftarrow 0$, $l \leftarrow 0$	*// initialize: best OF value; best parameters; counters*
4:	**for** $i = 2 \rightarrow \|\mathbf{x}\| + 1$ **do**	
5:	$\mathbf{r} \sim U(\mathbf{0}, \mathbf{I})$; $\mathbf{x}_i = \mathbf{x}^{lb} + \mathbf{r}(\mathbf{x}^{ub} - \mathbf{x}^{lb})$; $f_i \leftarrow y(\mathbf{x}_i)$	*// generate and evaluate new point*
6:	**end for**	
7:	$\mathbf{f} \leftarrow \mathbf{f}_\Delta$	*// create simplex (ordered list)*
8:	$\mathbf{x}_C = 1/\|\mathbf{x}\| \sum \mathbf{x}_i$, $i = 1 \rightarrow \|\mathbf{x}\|$	*// calculate the centre of gravity*
9:	**loop until termination criterion fulfilled** (n_k, n_l, t_{min}, f_{min})	*// main loop*
10:	$k \leftarrow k + 1$	*// increment iteration counter*
11:	$\mathbf{x}_{ref} \leftarrow \mathbf{x}_C + \alpha(\mathbf{x}_C - \mathbf{x}_{\|\mathbf{x}\|+1})$; $f_{ref} \leftarrow y(\mathbf{x}_{ref})$	*// calculate and evaluate reflected point*
12:	**if** $f_{ref} < f_1$ **then**	*// if reflected point is the best so far*
13:	$\mathbf{x}_{exp} \leftarrow \mathbf{x}_C + \gamma(\mathbf{x}_C - \mathbf{x}_{\|\mathbf{x}\|+1})$; $f_{exp} \leftarrow y(\mathbf{x}_{exp})$	*// calculate and evaluate expanded point*
14:	**if** $f_{exp} < f_{ref}$ **then**	*// if expanded point is the best so far*
15:	$\mathbf{f} \setminus \{f_{\|\mathbf{x}\|+1}\}$, $\mathbf{f} + \{f_{exp}\} \mid f_1 \leftarrow f_{exp}$	*// update simplex including expanded point*
16:	**else**	*// reflected point is the best so far*
17:	$\mathbf{f} \setminus \{f_{\|\mathbf{x}\|+1}\}$, $\mathbf{f} + \{f_{ref}\} \mid f_1 \leftarrow f_{ref}$	*// update simplex including reflected point*
18:	**end if**	
19:	**else if** $f_1 < f_{ref} < f_{\|\mathbf{x}\|}$ **then**	*// if reflected point is better than the semi-last point*
20:	$\mathbf{f} \setminus \{f_{\|\mathbf{x}\|+1}\}$, $\mathbf{f} + \{f_{ref}\}$	*// update simplex including reflected point*
21:	**else if** $f_{\|\mathbf{x}\|} < f_{ref}$ **then**	*// if reflected point is better than the worst point*
22:	$\mathbf{x}_{con} \leftarrow \mathbf{x}_C + \rho(\mathbf{x}_C - \mathbf{x}_{\|\mathbf{x}\|+1})$; $f_{con} \leftarrow y(\mathbf{x}_{con})$	*// calculate and evaluate contracted point*

Algorithm 2 Nelder-Mead's Simplex algorithm (NMSIM)

```
23:        if f_con < f_||x||+1 then                                   // if contracted point is better than the worst point
24:            f \ {f_||x||+1}, f + {f_con}                            // update simplex including contracted point
25:        else
26:            for i = 1 → ||x|| do
27:                x_red,i ← x_C + σ(x_i − x_||x||+1); f_red,i ← y(x_red,i)   // calculate and evaluate reduced point(s)
28:            end for
29:            f \ {f_2:(||x||+1)}, f + {f_red,1:||x||}                // update simplex including reduced points
30:        end if
31:    end if
32:    f* ← f_1; x* ← x(f*)                                            // update best objective function value and parameters
33: end loop                                                           // end
```

Covariance Matrix Adaptation – Evolution Strategy (CMA-ES) is a relatively recent algorithm conceived by Hansen (2006) and formalized by Hansen (2011). It belongs to a group of evolutionary algorithms, which, similarly to genetic algorithms, require many function evaluations for convergence. It has been used by Kostic and Gentile (2015) in the context of demand calibration.

The base idea is to mutate the population from generation to generation selecting only the best offspring. The critical elements for every iteration are the population size (λ), i.e. the number of search points (the number of function evaluations per iteration), the offspring size (μ), i.e. the number of selected points (the number of points taken into consideration for algorithm progression) and the step size (σ), i.e. the dispersion of the population generation. The algorithm is described in detail in Hansen (2011, 2006) and is presented below.

Algorithm 3 Covariance Matrix Adaptation - Evolution Strategy (CMA-ES)

```
1: f_0, x_0, x^ub, x^lb                                                // starting point and bounds
2: σ, λ, μ, w; n_k, n_l, t_min, f_min                                  // algorithm coefficients; termination criteria
3: μ_eff, c_σ, d_σ, c_C, c_1, c_μ, B, D, C, m, p_σ, p_C                // initialize algorithm variables
4: f* ← f_0; x* ← x_0; k ← 0, l ← 0                                    // initialize: best OF value; best parameters; counters
5: loop until termination criterion fulfilled (n_k, n_l, t_min, f_min) // main loop
6:     k ← k + 1                                                       // increment iteration counter
7:     for i = 1 → λ do
8:         z_i ~ N(0, I); y_i = BDz_i ~ N(0, C)                        // include eigenvectors and eigenvalues
9:         x_i = m + σy_i ~ N(m, σ^2C); f_i ← y(x_i)                   // calculate and evaluate new point
10:    end for
11:    f* ← min(f*, f_i | i = 1 → ||f||); x* ← x(f*)                   // update best objective function value and parameters
12:    z_W = Σ w_i z_i:λ | i = 1 → μ, Σ w_i = 1, w_i > 0 | i = 1 → μ   // weighted vector z
13:    y_W = Σ w_i y_i:λ | i = 1 → μ, Σ w_i = 1, w_i > 0 | i = 1 → μ   // weighted vector y
14:    m ← m + σy_W = Σ w_i x_i:λ | i = 1 → μ                          // update mean m
15:    p_σ ← (1 − c_σ)p_σ + (c_σ(2 − c_σ) μ_eff)^1/2 C^−1/2 y_W        // update evolution path for σ
16:    σ ← σ × exp((c_σ / d_σ) / (||p_σ|| / E||N(0, I)|| − 1))         // update step size σ
17:    p_C ← (1 − c_C)p_C + h_σ(c_C(2 − c_C)μ_eff)^1/2 y_W             // update evolution path for C
18:    C ← (1 − c_1 − c_μ)C + c_1(p_C p_C^T + δ(h_σ)C) + c_μ Σ w_i y_i:λ y_i:λ^T | i = 1 → μ   // update covariance matrix C
19:    (B, D) = Eigendecomposition(C)                                  // calculate eigenvalues and eigenvectors
20: end loop                                                           // end
```

3. Optimization algorithms and distributed computing

In this regard, SPSA can achieve from small to large savings in computation times based on its configuration. In its basic form, i.e. one gradient replication, SPSA does two independent function evaluations per iteration (i.e. per one gradient replication). Therefore, roughly speaking, this means a two times faster algorithm. In addition, as gradient replications are independent of each other, the speed up increases linearly with the number of gradient replications. In the case of 10 gradient replications in one iteration, with two function evaluations per one gradient replication, SPSA does 20 independent function evaluations, which roughly represent a 20 times faster computation. This becomes undoubtedly a game changer, and opens new possibilities in investigating methods limits and performance. The distributed computing possibility for SPSA is depicted in Fig. 1 (left).

When it comes to NMSIM, there are two occasions where distributed computing can be employed: the simplex creation and the reduced point calculation. Both cases involve a large number of independent function evaluations. The simplex is created once in the beginning, before the actual optimization takes place, conducting ||x|| + 1 function evaluations. The reduced point step does ||x|| function evaluations, hence there is the same benefit as with the simplex parallelization, but this case rarely occurs. As such, the more machines are available, the greater benefit from distributed computing. However, during optimization, there is no possibility of parallelizing simulations, as there are executed conditionally in sequence. The distributed computing possibility for NMSIM is depicted in Fig. 1 (middle).

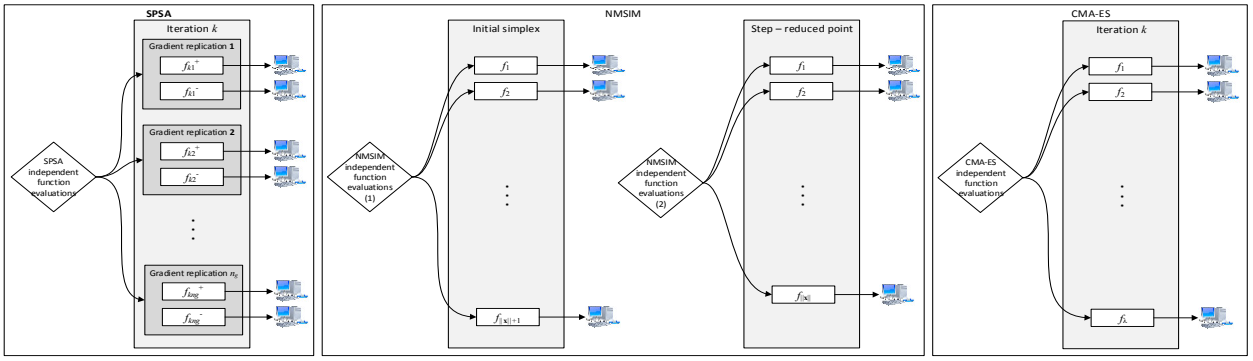

Fig. 1. Distributed computing possibilities for various algorithms: SPSA (left), NMSIM (middle) and CMA-ES (right).

CMAES, as all genetic and evolutionary types of algorithms, can profit substantially from parallelization. The greater the population size is, the greater benefit can be achieved, as those are all independent function evaluations. This property makes them ideal for this task. For instance, in the case of the population size of 30 points, one iteration can be speed up nearly 30 times. The parallelization now makes genetic and evolutionary algorithms a viable alternative, and it opens a completely new field of applications, not excluding real-time deployment. The distributed computing possibility for CMA-ES is depicted in Fig. 1 (right).

4. Numerical tests

The experiments were done in a form of laboratory experiments, where the true solution is known a priori. The simulation engine was Traffic Real-time Equilibrium (TRE; Gentile, 2010; Gentile et al., 2007). It is a first-order dynamic traffic simulation model with realistic representation of traffic phenomena, such as queues and congestion propagation. The test network used in the experiments was the southern part of Dusseldorf, Germany (Fig. 2). The network is composed of 155 zones, 118 of which being origin zones and 124 being destination zones. The zones are connected to the street network by 386 connectors. The network contains 1304 links, 500 nodes and 4060 permitted turns. As this is laboratory experiment, synthetic traffic data were created using simulation results of the DTA simulation using true demand flows. In this way, both flow and speed measurements discretized into 15-minute intervals were obtained for 99 count locations.

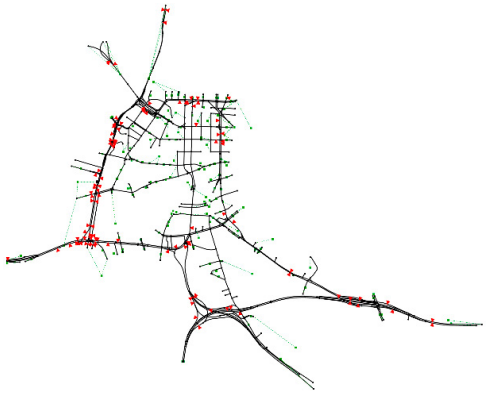

Fig. 2. Test network – the southern part of Dusseldorf, Germany.

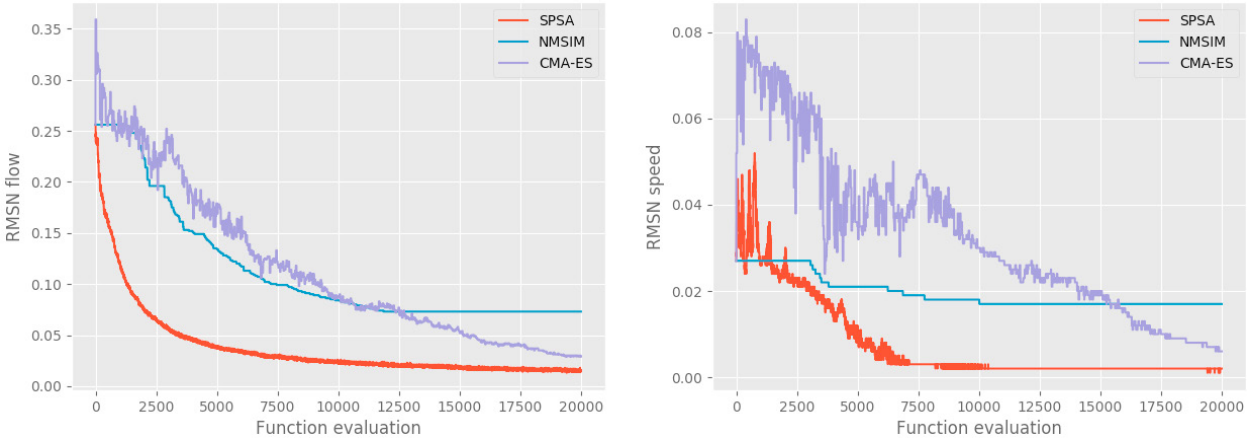

Fig. 3. Calibration performance in terms of function evaluations for flow (left) and speed (right) measurements.

5. Conclusion

Distributed computing can be used to tackle heavy calibration problems where cumbersome computations are needed. Besides providing a substantial speed up for the existing algorithms used in dynamic demand calibration, they open new possibilities for other algorithm types not considered so far due to their computational requirements. We proved on the test case that both stochastic approximation algorithms and evolutionary algorithms can achieve many times faster computation times. This property should be further investigated for both off-line and on-line use cases.

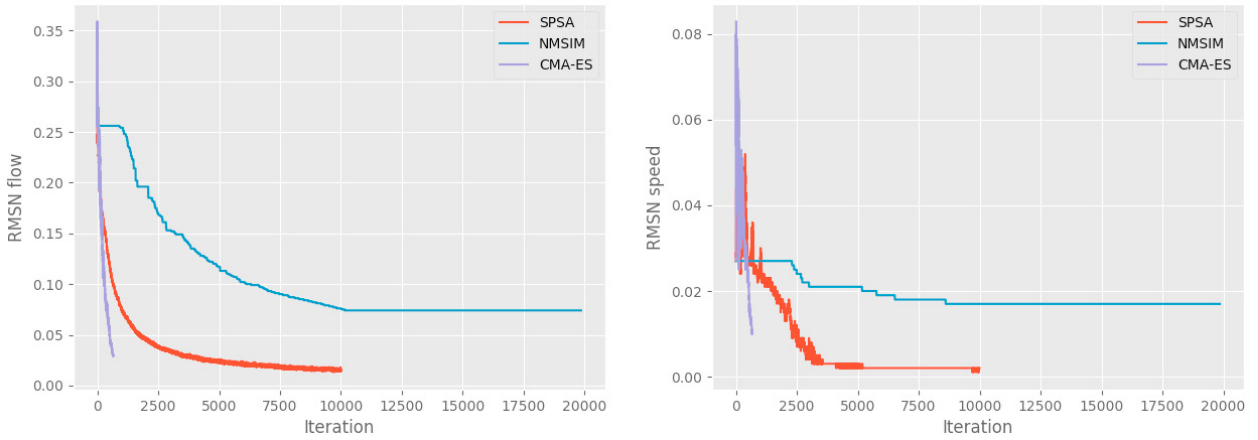

Fig. 4. Calibration performance in terms of iterations for flow (left) and speed (right) measurements.

References

Antoniou, C., Balakrishna, R., Koutsopoulos, H.N., Ben-Akiva, M., 2011. Calibration Methods for Simulation-Based Dynamic Traffic Assignment Systems. Int. J. Model. Simul. 31, 227–233. doi:10.2316/Journal.205.2011.3.205-5510

Antoniou, C., Barceló, J., Breen, M., Bullejos, M., Casas, J., Cipriani, E., Ciuffo, B., Djukic, T., Hoogendoorn, S., Marzano, V., Montero, L., Nigro, M., Perarnau, J., Punzo, V., Toledo, T., van Lint, H., 2016. Towards a generic benchmarking platform for origin-destination flows estimation/updating algorithms: Design, demonstration and validation. Transp. Res. Part C Emerg. Technol. 66, 79–98. doi:10.1016/j.trc.2015.08.009

Barceló, J., Casas, J., 2006. Stochastic Heuristic Dynamic Assignment Based on AIMSUN Microscopic Traffic Simulator. Transp. Res. Rec. J. Transp. Res. Board 1964, 70–80.

Ben-Akiva, M., Bierlaire, M., Burton, D., Koutsopoulos, H.N., Mishalani, R., 2001. Network State Estimation and Prediction for Real-time Traffic Management. Networks Spat. Econ. 1, 293–318. doi:10.1023/A:1012883811652

Cascetta, E., 2009. Transportation Systems Analysis: Models and Applications, 2nd ed. Springer. doi:10.1007/978-0-387-75857-2

Frederix, R., Viti, F., Tampère, C.M.J., 2013. Dynamic origin–destination estimation in congested networks: theoretical findings and implications in practice. Transp. A Transp. Sci. 9, 494–513. doi:10.1080/18128602.2011.619587

Gentile, G., 2015. Using the General Link Transmission Model in a Dynamic Traffic Assignment to simulate congestion on urban networks. Transp. Res. Procedia 5, 66–81. doi:10.1016/j.trpro.2015.01.011

Gentile, G., 2010. The General Link Transmission Model for Dynamic Network Loading and a comparison with the DUE algorithm, in: Tampère, C.M.J., Viti, F., Immers, L.H. (Eds.), New Developments in Transport Planning: Advances in Dynamic Traffic Assignment. Edward Elgar, Cheltenham, UK - Northampton, MA, USA.

Gentile, G., Meschini, L., 2011. Using dynamic assignment models for real-time traffic forecast on large urban networks, in: Proceedings of the 2nd International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS 2011). Leuven, Belgium.

Gentile, G., Meschini, L., Papola, N., 2007. Spillback congestion in dynamic traffic assignment: A macroscopic flow model with time-varying bottlenecks. Transp. Res. Part B Methodol. 41, 1114–1138. doi:10.1016/j.trb.2007.04.011

Hansen, N., 2011. The CMA Evolution Strategy: A Tutorial. Laboratoire de Recherche en Informatique (LRI), Paris, France. doi:10.1007/11007937_4

Hansen, N., 2006. The CMA evolution strategy: A Comparing Review. Stud. Fuzziness Soft Comput. 192, 75–102. doi:10.1007/11007937_4

Kostic, B., Gentile, G., 2015. Using Traffic Data of Various Types in the Estimation of Dynamic O-D Matrices, in: 2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS). Budapest, Hungary, pp. 66–73. doi:10.1109/MTITS.2015.7223238

Marzano, V., Papola, A., Simonelli, F., 2009. Limits and perspectives of effective O-D matrix correction using traffic counts. Transp. Res. Part C Emerg. Technol. 17, 120–132. doi:10.1016/j.trc.2008.09.001

Nelder, J.A., Mead, R., 1965. A simplex method for function minimization. Comput. J. 7, 308–313. doi:10.1093/comjnl/7.4.308

Spall, J.C., 1998. An Overview of the Simultaneous Perturbation Method for Efficient Optimization. Johns Hopkins Apl Tech. Dig. 19, 482–492.

Spall, J.C., 1992. Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation. IEEE Trans. Automat. Contr. 37, 332–341. doi:10.1109/9.119632

Tampère, C.M.J., Viti, F., Immers, L.H. (Eds.), 2010. New Developments in Transport Planning: Advances in Dynamic Traffic Assignment. Edward Elgar, Cheltenham UK and Northampton, MA, USA.