Ain Shams Engineering Journal 13 (2022) 101576

Contents lists available at ScienceDirect

Ain Shams Engineering Journal

journal homepage: www.sciencedirect.com

Civil Engineering

Parallelization of the B static traffic assignment algorithm

Tomas Potuzak [a,*], Frantisek Kolovsky [b]

[a] Department of Computer Science and Engineering, Faculty of Applied Sciences, University of West Bohemia, Univerzitni 8, 306 14 Plzen, Czech Republic
[b] Department of Geomatics, Faculty of Applied Sciences, University of West Bohemia, Univerzitni 8, 306 14 Plzen, Czech Republic

ARTICLE INFO

Article history:
Received 15 September 2020
Revised 2 September 2021
Accepted 4 September 2021
Available online 20 September 2021

Keywords:
Algorithm parallelization
B algorithm
Static traffic assignment
User equilibrium
Shared memory environment

ABSTRACT

A widely used technique for predicting traffic flows in individual roads of a road traffic network is the user-equilibrium (UE) traffic assignment (TA). This technique assigns trips from origins to destinations in a road traffic network so that all trips use the cheapest path. The cost of the path, which consists of roads (edges), is the sum of the roads costs. These costs increase with increasing flow in these roads. In this paper, we describe the parallelization of the B algorithm – a relatively new TA algorithm with a fast convergence to a solution. Since the nature of the algorithm and the nature of its fast convergence complicate the parallelization itself, we considered and implemented three parallel variants and tested them on real road traffic networks to investigate their convergence, usability, and speed. The parallelization is intended for a shared memory parallel computing environment. The description of the parallelization along with the performed tests is the main contribution of this paper.

1. Introduction

The traffic volume in road traffic networks is steadily increasing, which necessitates careful management of existing networks and planning of their future improvements. One of the important tools for these purposes is the traffic modeling. The general purpose of traffic modeling is to predict the traffic flows in individual roads of the modeled road traffic network based on the often incomplete input data. Traffic modeling incorporates wide range of techniques from deterministic graph-based algorithms considering only flows in roads to stochastic simulations considering every single vehicle in the road traffic network. One of the oldest [1] and still widely used technique is the user-equilibrium (UE) traffic assignment (TA). This technique assigns trips from origins to destinations in a road traffic network so that all trips use the cheapest path. The cost of the path, which consists of roads (edges), is the sum of roads costs. These costs increase with increasing flow in these roads [2].

Since the TA problem is here for more than half a century, there are several algorithms capable to solve it. The algorithms are in most cases sequential, so they do not exploit the mutli-core processors of nowadays computers. Despite of this setback, running on a standard desktop computer, some of these algorithms are able to find solution for quite large road traffic networks (large cities or entire states) in reasonable time of several seconds to several hours (depending on the road traffic network, the algorithm used, and other features). Still, this time can be far too high. In some cases, the TA of multiple scenarios must be completed (e.g., during a careful planning of a closure), in other cases, it is important to get the results as quickly as possible (e.g., during the planning of alternate routes immediately after a major accident). Even if there is enough time, the lower amount of waiting time always improves the user comfort.

A way, how to speedup a TA algorithm, is to use the parallel computing environment. Since the multi-core processors are incorporated virtually in all common desktop computers, the parallelization of a TA algorithm can be exploited by a wide variety of users. In this paper, we describe the parallelization of the B algorithm – a relatively new TA algorithm with a fast convergence to a solution [2]. Since the nature of the algorithm and the nature of its fast convergence complicate the parallelization itself, we considered and implemented three parallel variants and tested them on real road traffic networks to investigate their convergence, usability, and speed. The parallelization is intended for a shared memory parallel computing environment. The description of the parallelization along with the performed tests is the main contri-

* Corresponding author.
E-mail addresses: tpotuzak@kiv.zcu.cz (T. Potuzak), kolovsky@kgm.zcu.cz (F. Kolovsky).

Peer review under responsibility of Ain Shams University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.asej.2021.09.003
2090-4479/

bution of this paper. We were unable to find any parallelization of the B algorithm in existing literature.

The remainder of this paper is structured as follows. Section 2 discusses the related work. In Section 3, the B algorithm and its sequential implementation are described. In Section 4, the three parallel variants of the B algorithm, which we developed, are described in detail. The testing and its results are summarized in Section 5. The paper is concluded and our future work is described in Section 6.

2. Related work

This paper deals with a classical static traffic assignment (STA) based on the user equilibrium. These models are not time-dependent (unlike their dynamic traffic assignment counterparts) [3] and they are usually constructed for rush hours or for a daily average. The dynamic traffic assignment is a more precise assignment model, but is also more demanding and it is more difficult to obtain the result within reasonable computation time for real-size road traffic networks. Nowadays, static models are still widely used in practice [25]. In STA, there is only one time-independent origin–destination matrix (ODM) describing the numbers of vehicles driving among the zones. The zones are areas in the road traffic network, which act as origins (i.e., sources of vehicles) and/or destinations (i.e., sinks of vehicles). Each number in the ODM represents a single origin–destination pair (OD pair). From a single origin, the vehicles drive to multiple destinations. The road traffic network is then an edge-labeled oriented graph with nodes acting as crossroads and oriented edges acting as one-way roads (most neighboring nodes are interconnected with two edges for both travel directions). The label of each edge consists of its cost, capacity, and flow. The capacities of edges are constant while their costs increase with their increasing flows based on a cost function. The capacities and the initial costs are given. The flows of all edges are to be determined by a STA algorithm. Usually, the edges flows are loaded based on the OD pairs and then equalized [4].

2.1. Static traffic assignment algorithms

There are three classes of the algorithms solving the STA problem – the link-based, the path-based, and the bush-based [5,6]. Many of the algorithms (regardless their class) are iterative in nature and use the all-or-nothing (AON) assignment model to determine an initial solution. That means that, for each OD pair, the shortest path in the road traffic network is found and the flow of the OD pair is assigned to all edges of this path. So, the flow on an edge is equal to the sum of flows of all paths using this edge.

The link-based algorithms are the first algorithms developed. The well-known Frank-Wolfe (FW) algorithm and its modifications fall in this class [5,20]. They use the AON for initial assignment of OD pair flows to the road traffic network. They operate in link (edge) space. So, during individual iterations, the flows are moved between links. Their memory requirements are relatively low [6]. Nevertheless, their convergence to the high precision solution (i.e., UE) is slow [5,7]. In case of low precision results, their performance is good enough [5,20].

The path-based algorithms utilize the OD pairs separability of the STA problem. Again, they use the AON for initialization, but operate in path flows space. In each iteration, the flows are moved within one OD pair only (other pairs are fixed). For this purpose, all the paths and path flows must be stored [5], which increases the memory requirements. Compared to link-based algorithms, the path-based algorithms show faster convergence to a solution [6]. For example, the representatives are [21,22,23].

The bush-based algorithms utilize the OD pairs separability of the STA problem as well. However, the problem is decomposed to individual origins instead of individual OD pairs. In each iteration, the flows are moved within so-called *bush*, which is a directed connected acyclic sub-graph of the original graph (i.e., the road traffic network). The number of bushes corresponds to the number of origins. So, each bush contains paths from one origin to all its destinations. Each bush is usually initialized using the AON [5]. The bush-based algorithms are fast converging and memory-efficient making them a good choice for large road traffic networks [5,6,24]. After post-processing, algorithms from this group also provide the solution in path flow space, which is an advantage for ODM estimation and some engineering applications. The B algorithm, whose parallelization is the theme of this paper, is a bush-based algorithm (see Section 3).

2.2. Static traffic assignment algorithms parallelization

As stated in Section 1, we were unable to find any work describing the parallelization of the B algorithm. However, there are works describing or mentioning parallelization of other STA algorithm for a parallel (i.e., with shared memory present) or a distributed (without shared memory and with message passing as a standard form of inter-process communication) computing environment. Some of them are briefly described in this section.

One of the earliest works considering the parallelization of a STA algorithm is [8]. The paper considers distributed computing environment and two variants of decomposition of the STA problem into parallel sub-problems – one utilizing periodical state updates from other sub-problems and the other not utilizing such updates. While the second variant shows high level of parallelism, it converges slower than the first variant and also than the sequential variant. The first variant exhibits higher rate of convergence than the second variant due to the updates passed among the parallel sub-problems. The transfer of updates is performed concurrently along with the sub-problems computation to efficiently utilize the parallel or distributed computing environment and to minimize the waiting times [8].

In [9], the parallelization of STA using distributed disaggregated simplicial decomposition for a distributed computing environment is described in detail. The parallel algorithm shows the same convergence as its sequential counterpart. All processes contain the entire road traffic network, which enables them to perform shortest path searches without any inter-process communication. The working processes process sets of OD pairs. The inter-process communication is necessary to calculate the sums of flows in all edges of the road traffic network. There is also a load-balancing transferring OD pairs between the working processes based on their computation times. The implementation utilizes the single program multiple data (SPMD) paradigm and the PVM for the inter-process communication. The tests were performed using three real test cases with varying numbers of nodes (ca. thousands), edges, and OD pairs. It was observed that the increasing number of OD pairs positively influence the speedup, because the communication forms a decreasing portion of the computation time [9].

In [10], the parallelization possibilities are only mentioned. It is noted that the finding shortest path can be done concurrently without inter-process communication as long as each process contains the entire road traffic network. It is further stated that the STA algorithm can be parallelized by division of the OD pairs among working processes [10]. This approach would be useful for path-based class of the STA algorithms. No parallelization experiments are described, but there is the computation of possible speedup using Amdhal Law [10]. The parallelization of STA is only marginally mentioned in [3], which is focused mainly on the Disaggregated Simplicial Decomposition for the STA. The imple-

mentation for an efficient Matlab vector computation is described [3].

In [11], the speedup of the STA is achieved primarily using customizable contraction hierarchies. The main idea is to speedup the shortest path searching by employing a relatively slow preprocessing phase and a fast query phase. The shortest path searching is performed in all three types of the STA algorithms. However, the results are shown on the link-based conjugate Frank-Wolfe algorithm. The paper focuses on optimization of both phases for both sequential and parallel computing environments including ordering of OD pairs to optimize cache utilization. The tests were performed using three real test cases with varying numbers of nodes (ca. hundred thousands), edges, and OD pairs [11].

In [12], a framework for the STA computation designed for clusters is described. The paper proclaims the advantages of the distributed environment and the decomposition based on origins rather than road traffic network division (i.e., spatial decomposition). This approach requires less intense inter-process communication [12]. Parallelization of two algorithms – the Frank-Wolfe and a gradient-projection algorithm [7] – is described using map-reduce primitives and resilient distributed datasets to provide fault tolerance and scalability. The tests were performed using 8 various networks ranging from very small (24 nodes) to relatively large (ca. 14,000 nodes) [12].

The road traffic network division approach for a distributed computing environment is utilized in [6] and [13]. The entire road traffic network is divided into several interconnected regions and each region is solved separately. To ensure consistency of the entire road traffic network, there is a control process with a simplified model of the entire network, where the cities are replaced by artificial edges connecting their border nodes. The entire approach is iterative with alternating solving of the regions and of the simplified entire road traffic network [6]. A path-based STA algorithm is employed [13]. It is reported that the distributed approach converges to the same result as the STA of the entire original road traffic network [6]. The tests were performed on a road traffic network with thousands of nodes divided into two regions [13].

3. B algorithm description

Our implementation of the B algorithm is based on the original description in [2] with several modifications based on [14].

3.1. General algorithm description

The B algorithm is a bush-based user equilibrium (UE) static traffic assignment (STA) algorithm. As with the majority of the STA algorithms, the inputs of the algorithm are the road traffic network (an oriented graph) and the origin–destination matrix (ODM). The output is the equalized flows in individual roads (edges of the oriented graph). The algorithm is deterministic, no pseudo-random numbers are employed.

The entire computation is based on a set of bushes. As it was stated in Section 2, a bush is a connected acyclic sub-graph of the original graph. It contains all the nodes of the original graph, but only a subset of its edges. Each bush starts in a single origin and its edges carry flows from this origin to all its destinations. Initial bushes are formed using the minimal cost tree constructed from the original graph with its root in the origin of the bush. Only the edges, whose end node is further away from the origin than the start node, are added to the bush. The bush is then loaded with the flows from its origin to all its destinations using the minimal paths (i.e., the AON approach) [2]. The total flow in each edge of the graph (i.e., the road traffic network) is calculated as the sum of the flows in the corresponding edges in all the bushes.

Once all the bushes are created and loaded, the main iterative part of the algorithm begins. For each bush, the minimal cost tree and the maximal cost tree connecting the origin of the bush with all its nodes are constructed. Then, for each node, the flows are shifted from the maximal path to the minimal path to minimize their cost differences using the Newton's method. This way, the bush is equalized [2]. The current total flow in each edge of the graph is taken into account and updated during this process.

If the bush is not optimal, the bush is topologically improved (by adding and/or removing some edges) to contain some cheaper paths. Once the last bush is equalized, the overall convergence criterion (usually the *relative gap* [15] – see Section 3.2) is checked and a new iteration starts if needed [2,14].

3.2. Mathematical background

Let $G = (V, E)$ is a directed graph with set of nodes V and set of edges E. The set of zones is denoted as Z and the set of OD pairs is denoted as $W = Z \times Z$. The flow corresponding to OD pair $w_{ij} \in W$ is Q_{ij}. The origin–destination matrix (ODM) is denoted as Q. Further, the set of all used paths is denoted as P and the set $P_{ij} \subset P$ is subset of paths from zone i to zone j. Finally, x_e represents the total flow on the edge $e \in E$.

The UE can be mathematically defined as a variational inequality (VI) [16,17]. The optimal solution x^* must meet the following VI:

$$\sum_{p \in P} c_p\left(x_p^*\right)\left(x_p - x_p^*\right) \geqslant 0 \forall x \in \Lambda \tag{1}$$

where c_p represents the cost function of the path $p \in P$, $x = (x_p : p \in P)$ is a vector of path flows and Λ is a set of feasible flows defined as:

$$\Lambda = \left\{ x \geqslant 0 : \sum_{p \in P_{ij}} x_p = Q_{ij} \ \forall (i,j) \in W \right\} \tag{2}$$

This definition is the classical path-based definition with explicit path enumeration, but the B algorithm uses the implicit path enumeration provided by bushes. The bush-based formulation of the equilibrium is formulated in [18]. The equilibrium can be also defined in local form using *alternatives* [19]. The alternatives are the set of path from node n to node m. Then, the equilibrium occurs if:

$$c_a \begin{cases} = c_{mn} \text{ if } x_a > 0 \\ > c_{mn} \text{ if } x_a = 0 \end{cases} \tag{3}$$

for all $n \in V$ and $m \in V$, where c_{mn} is the minimal cost from node n to node m, c_a is the cost of alternative a and x_a is the flow on alternative a. Equation (3) expresses the first Wardrop's principle [1].

During the bush equilibration, the B algorithm finds the costliest and the cheapest alternatives for every m and n in the bush. The flow from the costliest alternative should be shifted to the cheapest alternative. This shift reduces the objective function [2]. The problem is to find Δx such that:

$$\underline{c}(\underline{x} + \Delta x) = \bar{c}\left(\bar{x} - \Delta x\right) \tag{4}$$

$$0 \leqslant \Delta x \leqslant \mu \tag{5}$$

where $\underline{c}$ ($\bar{c}$) is cost function of the cheapest (costliest) alternative and $\underline{x}$ ($\bar{x}$) represents current flow on the cheapest (costliest) alternative. The flow shift Δx must be positive and smaller then:

$$\mu = \min\left\{ x_{oe} : e \in \bar{p} \right\} \tag{6}$$

where $\bar{p}$ is the costliest path and x_{oe} is the flow from the origin $o \in Z$ on the edge $e \in E$. Condition (5) ensures that the flow on the bush is positive.

Equation (4) can be approximately solved using Newton method, which requires the path cost function and its derivative. The cost function and its derivative are:

$$c_p = \sum_{e \in p} c_e \tag{7}$$

$$c'_p = \sum_{e \in p} c'_e \tag{8}$$

where c_p is the cost of the path and c_e is the cost of edge e forming the path. In case that the cost function has shape of:

$$c = c_0\left(1 + \alpha\left(\frac{x}{C}\right)^{\beta}\right) \tag{9}$$

and c_e in (7) is substituted by (9), the derivative of path cost function with respect to flow is:

$$c'_p = \sum_{e \in p} \alpha\beta c_{0e} \frac{x_e^{\beta-1}}{C_e^{\beta}} \tag{10}$$

where c_0 is the free flow cost and C_e is the capacity of edge e forming the path p. Now, the Newton method [2] can be applied. Then, the flow shift (flow difference) is:

$$\Delta x_i = \Delta x_{i-1} + \frac{\bar{c} - \underline{c}}{\bar{c}' + \underline{c}'} \tag{11}$$

where $\Delta x_0 = 0$. The computation is iterative. In original version of the B algorithm [2], the number of iterations is set to 1.

If the bush is equilibrated, but the error is still too high, the topology of the bush must be improved to further decrease the relative gap. This topology improvement was implemented by [14]. The aim is to add links to the bush such that there is a chance to reduce the relative gap assuming that acyclicity is maintained. For this purpose, the topological distance is defined. The topological distance U_i of the node i is the maximum cost distance from the origin of the bush [14]. All edges with zero flow from the origin x_{oe} (unused edges) are removed from the bush and all edges $e_{ij} \in E$ that meet condition:

$$U_i < U_j \tag{12}$$

are added to the bush. This improved bush can be further equilibrated. The proofs of theorems described above can be found in [2,14], and [18].

To determine, whether the obtained solution is close enough to the optimal solution, or a new iteration is required, the relative gap is usually used [15]. The relative gap at kth iteration $rg(k)$ can be calculated using the gap at kth iteration $g(k)$ and the objective function at kth iteration $of(k)$. The gap can be expressed as:

$$g(k) = \sum_{e \in E} c_e(x_e(k)) \cdot (y_e(k) - x_e(k)) \leqslant 0 \tag{13}$$

where $c_e(x_e(k))$ is the cost of the edge e with total flow $x_e(k)$, $x_e(k)$ is the total flow on edge e in kth iteration, and $y_e(k)$ is the total flow on edge e given by the AON assignment based on costs in kth iteration $\{c_e(x_e(k))\}$. The objective function can be expressed as:

$$of(k) = \sum_{e \in E} \int_0^{x_e(k)} c_e(t)dt \tag{14}$$

The lower bound of the objective function at kth iteration $lb(k)$ can be then expressed using the objective function at kth iteration $lb(k)$ and the gap at kth iteration $g(k)$ as:

$$lb(k) = of(k) + g(k) \tag{15}$$

After the substitution of (13) and (14) into (15), the expression has the form of:

$$lb(k) = \sum_{e \in E} \int_0^{x_e(k)} c_e(t)dt + \sum_{e \in E} c_e(x_e(k)) \cdot (y_e(k) - x_e(k)) \tag{16}$$

and the relative gap is expressed as:

$$rg(k) = \frac{-g(k)}{|\max_k(lb(k))|} \geqslant 0 \tag{17}$$

where $\max_k(lb(k))$ is the maximum of lower bounds calculated from the start to the kth iteration [15].

3.3. Sequential implementation description

Our implementation, which was used for the testing, was written in Java. It follows the general description in Section 3.1. The algorithm consists of the initialization phase and the iterative phase. The scheme of the entire algorithm including the persistent data structures used during the entire computation is depicted in Fig. 1 with 16 nodes, 48 edges, 8 zones, 32 OD pairs, and 8 bushes.

The bushes are created in the initialization phase using the set of all edges E of the graph G and the set of all origins. For each origin, which coincides with a single node of the graph, a new bush is formed as follows. The distances to all nodes are determined using the Dijkstra shortest path algorithm. Then, the list of all edges is explored and the edges, whose end node is further away from the origin than its start node is added to the bush. Once all the bushes are formed, they are loaded with the flows using the ODM. For each bush, the minimal cost tree is constructed using a modified breadth-first search on its edges. Then, the flows of the OD pairs with the origin of the bush are added to the edges of the minimal paths to their destinations. The resulting set of bushes is designated as B.

Once all the bushes are loaded with the flows, the total flows of all the graph edges are calculated as the sum of the flows over all the bushes. The current total flow x_e of the edge e is calculated as:

$$x_e = \sum_{i \in B} x_{ei} \tag{18}$$

where x_{ei} is the total flow of edge e in ith bush. The current costs are calculated from these total flows and the initial (free flow) costs for all the edges using the cost function expressed as [2] (coefficients α and β inserted into (9)):

$$c_e(x_e) = c_{0e}\left(1 + 0.15\left(\frac{x_e}{C_e}\right)^4\right) \tag{19}$$

where $c_e(x_e)$ is the current cost, c_{0e} is the initial (free flow) cost, x_e is the current total flow, and C_e is the capacity of edge e. The relative gap is initialized to infinity.

The iterative phase is then performed. It is stopped when relative gap at kth iteration $rg(k)$ decreases under the preset target value $rg_{\max}$ or the number of completed iterations k reaches the preset maximal number of iterations $k_{\max}$. In each iteration, all the bushes are processed sequentially and, for each bush, several steps are performed (see Fig. 1).

First, the bush is topologically improved by adding some links while maintaining acyclicity (see Section 3.2). Then, the minimal cost tree and the maximal cost tree are constructed with their roots in the origin of the bush using a modified breadth-first search and the current edges costs. The flows shift then follows. Using the constructed trees, for each node of the graph (and of the bush) other then the origin of the bush, the minimal cost path and the maximal cost path to the origin are explored and the different segments of these paths are found. The iterative Newton method (see Section 3.2) is then used to partially shift flow from the maximal

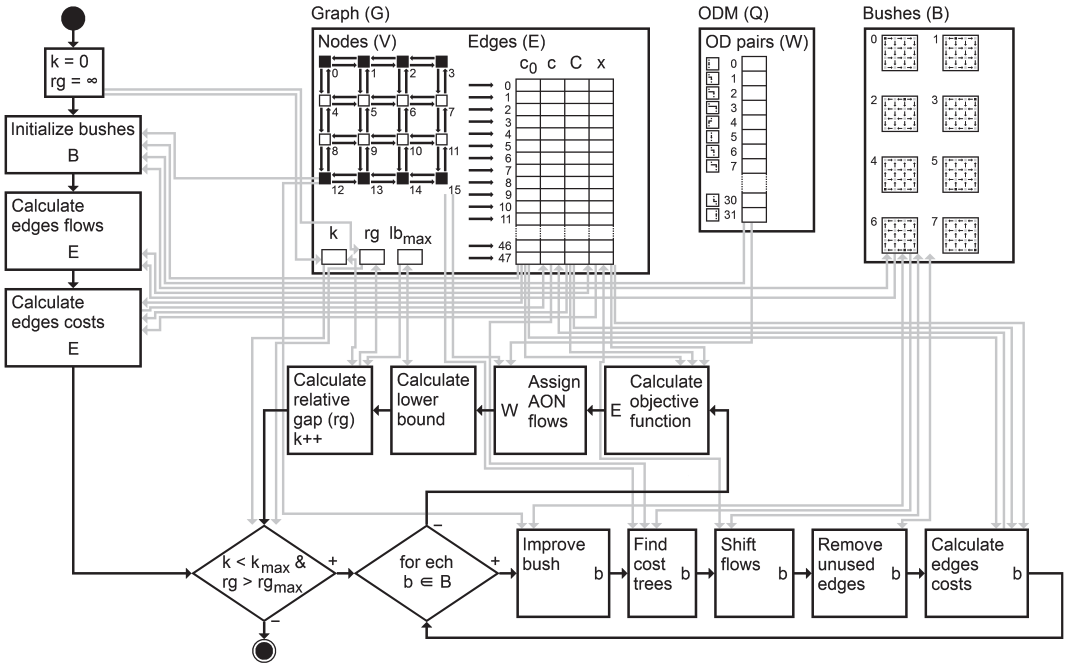


Fig. 1. The scheme of the entire B algorithm.

cost path segment to the minimal cost path segment to equalize flows of both segments. The maximal Newton method iterations count is set to 100. However, only a few iterations (ca. 4) are usually required to achieve equality of flows in order of 10^{-10}. The flows differences used to equalize the maximal and minimal flows are added not only to the flows of the bush, but also to the total flows of the graph edges. Once the flows shift is completed for all nodes of the bush, the unused (i.e., with zero flow) edges of the bush are removed. Finally, the current costs of the graph edges are recalculated using (19) and the current total flows of the graph edges. The algorithm then proceeds with the next bush.

Once all the bushes are processed, the current iteration continues with the computation of the relative gap $rg(k)$. For this purpose, the objective function at kth iteration $of(k)$ is calculated from the current total flows of the graph edges as (the integral expressed and the utilized values of the coefficients α and β inserted into (14)):

$$of(k) = \sum_{e \in E} c_{0e} \left(x_e(k) + 0.15 \cdot \frac{x_e^5(k)}{5C_e^4} \right) \tag{20}$$

where $x_e(k)$ is the current total flow at kth iteration, c_{0e} is the initial (free flow) cost, and C_e is the capacity of the edge e. Then, the AON assignment is performed with the current costs of the graph edges. For each OD pair w ($w \in W$), the shortest path $\underline{p_w}$ is found using the Dijkstra shortest path algorithm and the flow of the OD pair Q_w is added to all the edges forming this path. So, the resulting AON total flow of edge e at kth iteration $y_e(k)$ can be expressed as:

$$y_e(k) = \sum_{w \in W,\ e \in \underline{p_w}} Q_w \tag{21}$$

The AON total flows of the graph edges together with the current total flows of the graph edges are used to calculate the gap $g(k)$ using (13). The lower bound of the objective function $lb(k)$ is calculated using (15) and if this value is higher then the values calculated in previous iterations, it is stored as the maximum. This maximum is used for the calculation of the relative gap using (17). If the relative gap $rg(k)$ decreases under the target rg_{max}, the required fidelity was achieved, and the B algorithm ends. Otherwise, if the maximal number of iterations k_{max} was not reached, the next iteration starts.

The output of the algorithm is the equalized flows in the individual graph edges (i.e., in roads of the road traffic network). Since the algorithm is deterministic, the output is always the same for the same input.

4. B algorithm parallelization

In order to improve the speed of the B algorithm, we parallelized it for a shared-memory parallel computing environment. An example of this environment is a standard desktop computer with a multi-core processor, which is nowadays available to a wide variety of users. The description of the parallelization of the B algorithm along with the performed test is the main contribution of this paper.

4.1. B algorithm parallelization issues and main idea

In order to achieve a good speedup of a parallel computation, the overhead associated with the parallel execution must be minimized. The overhead is generally caused by the interactions (i.e., the synchronization and writing and reading to and from the shared memory) among the individual threads of the computation. So, it is important to divide the computation among the threads in a way that the number of their interactions is low.

The natural division seems to be based on bushes – each thread could process an assigned subset of bushes. However, the computations performed on individual bushes are not independent. During the processing of a bush, the total flows and costs of the graph edges changes (see Section 3.3). These changes influence the processing of the next bush, its processing influence the next one, and so on. For this reason, it is difficult to parallelize the B algorithm and to preserve its exact behavior and its fast convergence.

Of course, it would be possible to synchronize the threads after each bush to maintain the total edge flows and costs up-to-date. This would still change the behavior of the B algorithm, since the

order of the processing of the bushes would not be kept, but it is reasonable to expect that the influence on the convergence of the algorithm would be minimal. The major problem is that the number of bushes usually ranges in thousands meaning that there would be thousands of threads synchronizations per thread per iteration. The total overhead (i.e., waiting) associated with the synchronization would negate any speedup gained by the utilization of multiple threads.

A way how to implement the parallelization efficiently is to slightly modify the B algorithm to enable partially independent processing of the bushes. One possible modification is to recalculate the costs of the graph edges only once per iteration, not after each bush. Another possible modification is to recalculate the total flows of the graph edges once per iteration, not after each change in a bush. Both modifications are expected to negatively influence the convergence of the B algorithm. We implemented three variants, which are described in following sections in detail, and performed thorough tests to determine their efficiency and convergence (see Section 6). The parallel variant 1 (main parallel variant) utilizes only the former modification. The parallel variants 2 and 3 utilize both the former and the latter modifications and mutually differ in the implementation of the latter modification (i.e., the recalculation of the total flows of the graph edges).

4.2. B algorithm parallel implementation description

Our parallel implementation of the B algorithm is written in Java and utilizes standard Java threads. We implemented three variants, which differ only in the flows shift part. The main variant (parallel variant 1), which gives best results, is described in this section, the other two (parallel variants 2 and 3) are described in Sections 4.3 and 4.4.

The entire computation is divided into T ($T \geq 2$) working threads. One working thread also serves as the control thread. All the threads have the access to the data structures in shared memory. All the threads are being synchronized using a barrier and all potentially conflicting writing to the data structures are performed solely by the control thread during the synchronization. The control thread also performs any sequential parts of the computation. Between the synchronizations, the working threads can write only to parts of the data structures, which are exclusively assigned to them. The load is divided uniformly among the working threads, since we presume a homogeneous computing environment (i.e., all the processor cores with the same computing power).

As arise from Section 4.1, our parallelization is based on the parallel processing of bushes. However, there are also other computations on the entire graph (more specifically its edges), which can be parallelized as well, but cannot be conveniently divided using bushes (e.g., the calculation of total flows and costs of the graph edges). For this reason, ith thread has assigned not only a subset of bushes B_i ($B_i \subset B$), but also a subset of edges of the entire graph E_i ($E_i \subset E$). These subsets are constant during the entire computation.

The entire parallel computation including the persistent data structures used during the entire computation is depicted in Fig. 2 with 16 nodes, 48 edges, 8 zones, 32 origin–destination pairs, and 8 bushes. At the start, the bushes and edges of the entire graph are divided into subsets assigned to individual working threads. The threads then proceed with the initialization phase. They create the assigned bushes and fill them with flows based on the ODM (ith thread creates the subset of bushes B_i) the same way as in the sequential computation (see Section 3.3). Then, the barrier synchronization is performed to ensure that all threads completed their parts. The threads then continue with the calculation of total flows and costs of the edges (ith thread handles its set of edges E_i). The initialization phase is ended by the barrier synchronization and the iterative phase begins.

The iterative phase is, similarly to the sequential version, stopped when the relative gap at kth iteration $rg(k)$ decreases under preset target value $rg_{\max}$ or the number of completed iterations k reaches the preset maximal number of iterations $k_{\max}$. This is checked by all the working threads. In each iteration, the working threads topologically improve all their bushes first. Then, each working thread constructs and stores the minimal and the maximal cost trees for each bush of its subset B_i using the current edges costs. The edges cost are the same for all the bushes. This is a difference in comparison to the sequential version, where the edges costs are updated along with the edges flows after processing of each bush. So, the constructed cost trees may slightly differ from the cost trees of the sequential version. In the main parallel variant, this is the only difference influencing the convergence in comparison to the sequential version. After the trees construction, the threads perform the barrier synchronization.

The flows shift is performed sequentially by the control thread in the same manner as in the sequential version. The flows differences used to equalize the maximal and minimal flows are added to the total flows of the graph edges immediately after each minimal/maximal cost path pair flow shift is completed as in the sequential version. This is important for the convergence, because all minimal/maximal cost path pair flow shifts performed prior to the currently processed shift are taken into account across all the bushes. The pseudocode for the flows shift part of the main parallel variant is depicted in Fig. 3. This flows shift part of the B algorithm is the only difference of the three implemented parallel variants. It ends with the barrier synchronization.

After the flows shift, the unused edges of the bushes (i.e., edges with zero flow) are removed. Each working thread processes its subset of bushes B_i. Then, the current costs of the graph edges are recalculated using (19) and the current total flows of the graph edges updated during the flows shift. Each working thread processes its subset of edges E_i. It should be noted that there is no synchronization between the unused edge removal and the costs recalculation. The reason is that the edges removed from individual bushes have no effect on the cost recalculation. Hence, the synchronization is performed only after the costs recalculation.

Each iteration is finished with the parallel computation of the relative gap $rg(k)$. First, the objective function at kth iteration $of(k)$ is calculated from current total flows of edges. Each thread calculates its part of the objective function $of_i(k)$ from its subset of edges E_i as:

$$of_i(k) = \sum_{e \in E_i} c_{0e} \left(x_e(k) + 0.15 \cdot \frac{x_e^5(k)}{5C_e^4} \right) \tag{22}$$

where $x_e(k)$ is the current total flow at kth iteration, c_{0e} is the initial (free flow) cost, and C_e is the capacity of the edge e. Equation (22) is based on (20), but only for a subset of the graph edges E_i. Once the working threads finish the calculation of their parts of the objective function $of_i(k)$, they perform the barrier synchronization. The total objective function at kth iteration $of(k)$ is then calculated by the control thread as:

$$of(k) = \sum_{i=1}^{T} of_i(k) \tag{23}$$

where T is the number of working threads and the $of_i(k)$ is the part of the objective function in kth iteration calculated by ith thread. Then, another barrier synchronization follows and the parallel AON is performed. The AON can be performed in parallel easily, since multiple shortest path searches can be performed concurrently. Each working thread processes a subset of origin–destina-

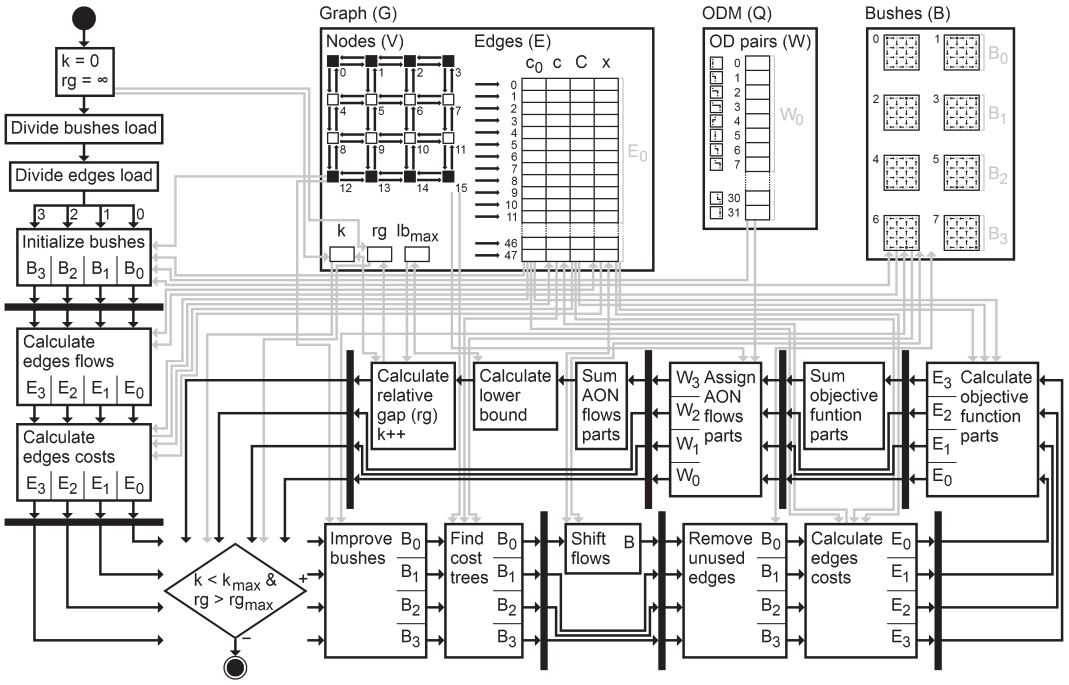


Fig. 2. The scheme of the entire parallel B algorithm.

tion pairs W_i. This subset corresponds to its subset of bushes B_i – the W_i subset contains all OD pairs with the same origins as the bushes in the B_i subset. To avoid the concurrent writing to the total flows of the graph edges during the adding of an OD pair flow, each working thread has its own set of AON partial flows of the graph edges. So, the resulting partial flow of the edge e at kth iteration in ith thread $y_{ei}(k)$ can be expressed as:

$$y_{ei}(k) = \sum_{w \in W_i,\ e \in \underline{p_w}} Q_w \tag{24}$$

where w ($w \in W_i$) is the OD pair, $\underline{p_w}$($e \in \underline{p_w}$) is the shortest path from the origin to the destination, and Q_w is the flow of the OD pair w. Once the working threads process their subsets of OD pairs W_i, the threads perform the barrier synchronization and the partial flows of the graph edges in these threads should have values corresponding to (24). The resulting AON total flow of each edge is then calculated by the control thread as:

$$y_e(k) = \sum_{i=1}^{T} y_{ei}(k) \tag{25}$$

where T is the number of working threads. The control thread then finishes the computation of the relative gap $rg(k)$ (including the calculation of lower bound) in the same way as in the sequential version (see Section 3.3). Then, the last barrier synchronization in the current iteration is performed. Again, if the relative gap decreases under the target value rg_{max}, the required fidelity was achieved,

$G(E, V)$ – the graph representing road traffic network with set of nodes V and set of edges E
B – the set of bushes
x_{Ge} – the flow of edge $e \in E$ stored in graph G
x_{be} – the flow of edge $e \in E$ stored in bush $b \in B$
ΔX_b – the edge flow differences from bush $b \in B$
Δx_{be} – the edge flow difference of edge $e \in E$ from bush b $\in B$
$ct_{b\min}$ – the minimal cost tree of bush $b \in B$
$ct_{b\max}$ – the maximal cost tree of bush $b \in B$
$\Delta p_{bn\min}$ – different path segment of the minimal cost path from node $n \in V$ to bush $b \in B$ origin
$\Delta p_{bn\max}$ – different path segment of the maximal cost path from node $n \in V$ to bush $b \in B$ origin

```
shiftFlows()
  for all b ∈ B do
    for all n ∈ V do
      (Δp_bnmin, Δp_bnmax) ← find different path segments using n, ct_bmin, and ct_bmax
      ΔX_b ← determine edges flow differences with Newton method using b, Δp_bnmin, and Δp_bnmax
      for all Δx_be ∈ ΔX do
        x_Ge ← x_Ge + Δx_be
        x_be ← x_be + Δx_be
      end for
    end for
  end for
end shiftFlows()
```

Fig. 3. Pseudocode for the flows shift part of the main parallel variant (parallel variant 1).

and the parallel B algorithm ends. Otherwise, if the maximal number of iterations k_{max} was not reached, the next iteration starts.

The main parallel variant described above has a potential for a high speedup in comparison to the sequential version, since there is a low number of barrier synchronizations. There are only two barrier synchronizations in the initiation phase and only seven barrier synchronizations per iteration in the iterative phase (see Fig. 2). It should be also noted that the number of iterations is low and ranges from one to several tens even for very large road traffic networks (see Section 5.2). On the other hand, there are sequential parts performed by the control thread, which negatively influence the speedup – the entire flows shift part and the sums performed during the computation of the relative gap (see above). Also, since the main parallel variant is a slightly different algorithm in comparison to the sequential version (see above), it was expected that its convergence will be slower. Nevertheless, the performed tests confirmed, that the convergence is fast enough to achieve a good speedup while achieving similar relative gap to the sequential version (see Sections 5.2 and 5.3).

The described main parallel variant (parallel variant 1) is deterministic and, for the same input, gives the same output regardless of the number of working threads. There are negligible differences caused by the limited fidelity of the double type (see Section 5.2 for details).

4.3. Parallel flows shift with Bush-based edge flows differences

In addition to the main parallel variant described in previous section (parallel variant 1), we experimented with the parallel flows shift to achieve additional speedup of the computation. Two variants of the parallel flows shift were implement and tested – with bush-based edge flows differences (parallel variant 2), which is described here, and with thread-based edge flows differences (parallel variant 3), which is described in Section 4.4. Both variants differ from the main variant only in the flows shift part, the other parts of all three variants are identical.

In the variant with bush-based edge flow differences, each working thread performs the flows shift for its subset of bushes B_i. The required minimal and maximal costs trees for all the bushes are already constructed before the flows shift part begins (see Section 4.2). For each bush, the maximal and minimal cost paths are explored for each node other than the origin of the bush and the different segments of these paths are found. Using the iterative Newton method, the flows differences are calculated, which are used to equalize flows on these segments. This is very similar to the main parallel variant and the sequential version of the B algorithm. However, one important difference is that the total flows of the graph edges are not changed during the entire flows shift. The calculated flows differences are added to the corresponding bush and are also stored for each bush in a separate data structure. These stored differences are taken into account during the processing of the current bush, but not the other bushes assigned to the working thread. Once the flows shift is finished for all the bushes, the working threads perform the barrier synchronization. Only then, the control thread adds the stored flow differences to the total flows of the graph edges. So, the total flow of edge e in $(k+1)$th iteration $x_e(k+1)$ is calculated using following expression:

$$x_e(k+1) = x_e(k) + \sum_{i \in B} \Delta x_{ei}(k) \tag{26}$$

where $x_e(k)$ is the total flow of edge e in kth iteration, and $\Delta x_{ei}(k)$ is the flow difference of edge e from ith bush from the set of all bushes B. Then, the control thread performs the barrier synchronization. The pseudocode for the flows shift part of the parallel variant 2 is depicted in Fig. 4.

The immutability of the total flows of the graph edges during the parallel flows shift is necessary in order to avoid their concurrent modification. The obvious consequence is that, in one iteration, a flows equalization performed in a bush influences the following flows equalizations in the same bush, but not the flows equalizations in the other bushes (of the same thread or of the other threads). The other bushes are influenced via the total flows of the graph edges only in the following iteration.

It was expected that the features of this variant described above will lead to a slight additional speedup of the entire computation, since all the parts of the computation are now parallel, but for the price of a slower convergence. However, the performed tests indicate that there is negligible additional speedup and the convergence is affected up to the point of unusability in most cases (see Section 5.2). Similarly to the main parallel variant, this variant is deterministic and, for the same input, gives the same output regardless of the number of working threads (see Section 5.2).

4.4. Parallel flows shift with Thread-based flows differences

The variant with thread-based flow differences (parallel variant 3) is very similar to its bush-based counterpart (parallel variant 2) described in Section 4.3. The total flows of the graph edges remain immutable during the parallel flows shift. The only difference is that there is a separate data structure for the storing of the calculated flows differences not for each bush, but rather for each thread. So, in each working thread, the stored flows differences are taken into account across all the bushes B_i assigned to the working thread. Again, the control thread adds the stored flow differences to the total flows of the graph edges once the flows shift is finished for all the bushes and the working threads performed the barrier synchronization. However, the total flow of edge e in $(k+1)$ th iteration $x_e(k+1)$ is calculated using the following expression:

$$x_e(k+1) = x_e(k) + \sum_{i=1}^{T} \Delta x_{ei}(k) \tag{27}$$

where $x_e(k)$ is the total flow of edge e in kth iteration, T is the number of working threads, and $\Delta x_{ei}(k)$ is the flow difference of edge e from ith thread. Expression (27) should be calculated faster than (26) since there is a far lower number of addends (corresponding to the number of working threads) than in (26), where the number of addends corresponds to the number of bushes. The pseudocode for the flows shift part of the parallel variant 3 is depicted in Fig. 5.

The obvious consequence is that, in one iteration, a flows equalization performed in a bush influences the following flows equalization both in the same bush and in the other bushes assigned to the same working thread. The bushes of the other working threads are influenced via the total flows of edges only in the following iteration. Another consequence is that the convergence is, unlike parallel variant 1 and 2, dependent on the number of working threads. With the growing number of threads a worse convergence can be expected. This is not a desired behavior. Still, it was expected that the features of this parallel variant 3 improve its convergence in comparison to the parallel variant 2 while preserving similar speed. The performed tests indicate that the convergence of the parallel variant 3 is indeed superior to the convergence of the parallel variant 2 (but inferior to that of the main parallel variant) and its speed is slightly higher as well. The details are described in Sections 5.2 and 5.3. The summary of all the parallel variants differences is provided in Table 1.

5. Tests and results

All the described parallel variants were thoroughly tested and compared with each other and with the sequential version. The

```
G(E, V) – the graph representing road traffic network with set of nodes V and set of edges E
B_i ⊂ B – the subset of bushes processed by ith working thread
x_Ge – the flow of edge e ∈ E stored in graph G
x_be – the flow of edge e ∈ E stored in bush b ∈ B_i
ΔX_B – the bush-based edge flow differences
Δx_Bbe ∈ΔX_B – the edge flow difference of edge e ∈ E from bush b ∈ B_i stored in ΔX_B
ΔX_b – the edge flow differences from bush b ∈ B_i
Δx_be – the edge flow difference of edge e ∈ E from bush b ∈ B_i
ct_bmin – the minimal cost tree of bush b ∈ B_i
ct_bmax – the maximal cost tree of bush b ∈ B_i
Δp_bmmin – different path segment of the minimal cost path from node n ∈ V to bush b ∈ B_i origin
Δp_bmmax – different path segment of the maximal cost path from node n ∈ V to bush b ∈ Bi origin
--------------------------------------------------------------------------------
ΔX_B ← initialize 2D array with first dimension corresponding to bushes count and second to edges count
...
bushParallelShiftFlows() – performed by ith working thread
  for all b ∈ B_i do
    for all n ∈ V do
      (Δp_bmmin, Δp_bmmax) ← find different path segments using n, ct_bmin, and ct_bmax
      ΔX_b ← determine edges flow differences with Newton method using b, Δp_bmmin, Δp_bmmax, and ΔX_B
      for all Δx_be ∈ΔX_b do
        x_be ← x_be + Δx_be
        x_Bbe ← x_Bbe + Δx_be
      end for
    end for
  end for
end bushParallelShiftFlows()

bushParallelShiftFlowsGather() – performed by the control thread
  for all b ∈ B do
    for all e ∈E do
      x_Ge ← x_Ge + Δx_Bbe
      Δx_Bbe ← 0
    end for
  end for
end bushParallelShiftFlowsGather()
```

Fig. 4. Pseudocode for the flows shift part of the parallel variant 2.

tests were focused on the convergence of the variants (see Section 5.2) and on their computation time (see Section 5.3). The testing environment and the course of testing are described in Section 5.1.

5.1. Testing environment and course of testing

The testing was performed on two desktop computers designated as HW1 and HW2. The HW1 incorporates Intel Core i7-4770 CPU at 3.40 GHz with 4 physical cores with Hyperthreading (8 logical cores) and 16 GB of RAM. It uses Windows 7 Pro 64 bit operating system and Java 8. The HW2 incorporates Intel Xeon E5-2630 v2 CPU at 2.60 GHz with 6 physical cores with Hyperthreading (12 logical cores) and 16 GB of RAM. It uses Windows 10 Pro 64 bit operating system and Java 11.

For the testing, one small and two large real road traffic networks were used. The small road traffic network designated as *network* 1 is the road traffic network of Pilsen, fourth largest city in Czech Republic. It incorporates 3 727 nodes, 9 036 edges, and 67 043 OD pairs. The large road traffic network designated as *network* 2 is the road traffic network around Antwerp in Belgium, incorporating 15 996 nodes, 35 930 edges, and 433 614 OD pairs. The large road traffic network designated as *network 3* is the road traffic network of Birmingham incorporating 14 639 nodes, 33 937 edges, and 471 637 OD pairs. This network was downloaded from the publicly available repository of road traffic networks [26].

The sequential version and all three parallel variants of the B algorithm were tested using both computers and both road traffic networks. There were three values of the target relative gap $rg_{\max}$ – 0.01, 0.001, and 0.0001. In many comparison works, the target relative gaps are set to smaller values between 10^{-8} and 10^{-14}, but these precision levels are unnecessarily high for most applications. Boyce et al [15] suggested that the relative gap 0.0001 is enough. Also, Mitradjieva et al [20] used this value as a target relative gap. For example, we use the parallel B algorithm as a part of the computing engine that provides the UE for the interactive application where speed is more important than precision. The maximal number of iterations $k_{\max}$ was set to 10 for the networks 1 and 3 and 75 for the network 2. These values were set based on preliminary testing according to the number of iterations required by the sequential version and the main parallel variant of the B algorithm.

The observed parameters were the relative gap rg in each iteration, the relative gap rg_f achieved in the last iteration k_f, and the computation time τ_f. Since all the tested algorithms are deterministic, it would be sufficient to perform one run for each inputs setting to determine the relative gap and the iteration. However, for the computation time, multiple runs are required to mitigate the influence of other essential system processes running by the operating system. All unessential processes where shut down before the testing. So, 12 runs were performed for each inputs setting. Two runs with the highest measured computation time were discarded and the remaining ten values were averaged. The loading of the road traffic network and of the origin–destination matrix to memory was not included to the observed computation time.

The parallel variants of the B algorithm were tested for various numbers of working threads ranging from 2 to 8 on the HW1 and from 2 to 12 on the HW2. The maximal values correspond to the number of logical cores of both computers. Only the main parallel variant, which gives by far the best results, was tested on both the HW1 and HW2. The remaining two variants were tested on the

```
T – the number of working threads
G(E, V) – the graph representing road traffic network with set of nodes V and set of edges E
B_i ⊂ B – the subset of bushes processed by ith working thread
x_Ge – the flow of edge e ∈ E stored in graph G
x_be – the flow of edge e ∈ E stored in bush b ∈ B_i
ΔX_T – the thread-based edge flow differences
Δx_Tie ∈ΔX_T – the edge flow difference of edge e ∈ E from ith working thread stored in ΔX_T
ΔX_b – the edge flow differences from bush b ∈ B_i
Δx_be – the edge flow difference of edge e ∈ E from bush b ∈ B_i
ct_bmin – the minimal cost tree of bush b ∈ B_i
ct_bmax – the maximal cost tree of bush b ∈ B_i
Δp_bnmin – different path segment of the minimal cost path from node n ∈ V to bush b ∈ B_i origin
Δp_bnmax – different path segment of the maximal cost path from node n ∈ V to bush b ∈ B_i origin
-------------------------------------------------------------------------
ΔX_T ← initialize 2D array with first dimension corresponding to threads count and second to edges count
...
threadParallelShiftFlows() – performed by ith working thread
  for all b ∈ B_i do
    for all n ∈ V do
      (Δp_bnmin, Δp_bnmax) ← find different path segments using n, ct_bmin, and ct_bmax
      ΔX_b ← determine edges flow differences with Newton method using b, Δp_bnmin, Δp_bnmax, and ΔX_T
      for all Δx_be ∈ΔX_b do
        x_be ← x_be + Δx_be
        x_Tie ← x_Tie + Δx_be
      end for
    end for
  end for
end threadParallelShiftFlows()

threadParallelShiftFlowsGather() – performed by the control thread
  for i in <1; T> do
    for all e ∈E do
      x_Ge ← x_Ge + Δx_Tie
      Δx_Tie ← 0
    end for
  end for
end threadParallelShiftFlowsGather()
```

Fig. 5. Pseudocode for the flows shift part of the parallel variant 3.

HW1 only, since these tests adequately showed their issues. Also, only the computation time was observed on the HW2, since the utilized computer has no effect on the convergence of the algorithm. All the performed tests are summarized in Table 2.

5.2. Convergence results

The convergence of the parallel variants of the B algorithm was expected to be worse than of its sequential version. It was investigated using the trend of the relative gap in individual iterations including the last one. The tests were performed only using the HW1. The relative gaps (rg_f) achieved in last iteration (k_f) for the network 1 (small) are summarized in Table 3 for the sequential version and in Table 4 for all the parallel variants (the main variant is emphasized by a bold font).

For the network 1 (small), the sequential version and all the parallel variants required only one iteration to achieve target relative gap rg_{max} of 0.01 and 0.001. The relative gap rg_{max} of 0.0001 is achieved only by the sequential version and the parallel variant 1 (main) in two iterations. The remaining two parallel variants were unable to reach this value in the preset maximum of 10 iterations. Instead, the achieved relative gap oscillated and did not converge to the target relative gap. In fact, both parallel variant 2 and 3 achieved best values of relative gap in first iteration with the exception of variant 3 with two working threads, which achieved the best value in second iteration (see Fig. 6a). Although it would be possible to allow more iterations to be performed, there is no point, since the significantly higher number of iterations hampers any possible speedup in comparison to the sequential version. On the other hand, the main parallel variant gives only slightly worse results then the sequential version and requires the same numbers of iterations (see Fig. 6b).

It should be noted that the parallel variants 1 and 2 give the same results regardless the number of working threads as expected. The results of the parallel variant 3 depend on the number of threads – this is the reason why there are four curves for the parallel variant 3 in Fig. 6. This is an expected behavior (see Section 4.4). However, we can observe in Table 4 that the achieved relative gap rg_f is not steadily increasing with the increasing number of threads as expected – the highest (i.e., worst) rg_f is achieved for 4 working threads, not for 8. Nevertheless, the trend of relative gaps from the entire computation depicted in Fig. 3a shows that the relative gap values for 4, 6, and 8 threads significantly oscillates and

Table 1
Summary of all the parallel variant differences.

Feature	Parallel variant 1	Parallel variant 2	Parallel variant 3
Parallel flows shift	No	Yes	Yes
Additional structure to store flow differences	No	Yes	Yes
Additional structure size	N/A	$\|B\| \times \|E\|$	$T \times \|E\|$
Flows shift in a bush influencing further bushes in the same iteration	Yes	No	Only bushes of the same thread
Result dependent on the number of threads (T)	No	No	Yes
Sufficient convergence	Yes	No	No

Table 2
Summary of performed tests.

Computer	Traffic network	rg_{max}	Algorithm	Threads (T)	Observed parameters
HW1	Network 1 (small)	0.01	Sequential	1	Computation time (τ_f) & relative gap (rg, rg_f)
			Parallel variant 1	2, 4, 6, 8	
			Parallel variant 2		
			Parallel variant 3		
		0.001	Sequential	1	
			Parallel variant 1	2, 4, 6, 8	
			Parallel variant 2		
			Parallel variant 3		
		0.0001	Sequential	1	
			Parallel variant 1	2, 4, 6, 8	
			Parallel variant 2		
			Parallel variant 3		
	Network 2 (large)	0.01	Sequential	1	
			Parallel variant 1	2, 4, 6, 8	
			Parallel variant 2		
			Parallel variant 3		
		0.001	Sequential	1	
			Parallel variant 1	2, 4, 6, 8	
			Parallel variant 2		
			Parallel variant 3		
		0.0001	Sequential	1	
			Parallel variant 1	2, 4, 6, 8	
			Parallel variant 2		
			Parallel variant 3		
HW2	Network 1 (small)	0.01	Sequential	1	Computation time (τ_f)
			Parallel variant 1	2, 4, 6, 8, 10, 12	
		0.001	Sequential	1	
			Parallel variant 1	2, 4, 6, 8, 10, 12	
		0.0001	Sequential	1	
			Parallel variant 1	2, 4, 6, 8, 10, 12	
	Network 2 (large)	0.01	Sequential	1	
			Parallel variant 1	2, 4, 6, 8, 10, 12	
		0.001	Sequential	1	
			Parallel variant 1	2, 4, 6, 8, 10, 12	
		0.0001	Sequential	1	
			Parallel variant 1	2, 4, 6, 8, 10, 12	
	Network 3 (large)	0.01	Sequential	1	
			Parallel variant 1	2, 4, 6, 8, 10, 12	
		0.001	Sequential	1	
			Parallel variant 1	2, 4, 6, 8, 10, 12	
		0.0001	Sequential	1	
			Parallel variant 1	2, 4, 6, 8, 10, 12	

Table 3
Relative gap (rg_f) in last iteration (k_f) for network 1 (small) for the sequential version.

rg_{max}	k_f	rg_f
0.01 (10^{-2})	1	$2.31 \cdot 10^{-4}$
0.001 (10^{-3})	1	$2.31 \cdot 10^{-4}$
0.0001 (10^{-4})	2	$6.70 \cdot 10^{-6}$

the achieved relative gap rg_f highly depends on the maximal number of iterations k_{max}, which was set to 10. The relative gap for 4 working threads is indeed better than for 6 and 8 working threads in all the iterations except the last one. With k_{max} set for example to 8, the values of rg_f for 2, 4, 6, and 8 threads would be steadily increasing as expected (see Fig. 6a). The exact nature of the oscillation depicted in Fig. 6 is highly network-dependent – the individual trends are quite different for the large road traffic network (network 2 – see Fig. 7).

Table 4
Relative gap (rg_f) in last iteration k_f for network 1 (small) for the parallel variants.

rg_{max}	Threads (T)	Variant 1 (main)		Variant 2		Variant 3	
		k_f	rg_f	k_f	rg_f	k_f	rg_f
0.01	2	**1**	$\mathbf{2.33 \cdot 10^{-4}}$	1	$9.42 \cdot 10^{-4}$	1	$8.68 \cdot 10^{-4}$
	4	**1**	$\mathbf{2.33 \cdot 10^{-4}}$	1	$9.42 \cdot 10^{-4}$	1	$8.89 \cdot 10^{-4}$
	6	**1**	$\mathbf{2.33 \cdot 10^{-4}}$	1	$9.42 \cdot 10^{-4}$	1	$9.11 \cdot 10^{-4}$
	8	**1**	$\mathbf{2.33 \cdot 10^{-4}}$	1	$9.42 \cdot 10^{-4}$	1	$8.92 \cdot 10^{-4}$
0.001	2	**1**	$\mathbf{2.33 \cdot 10^{-4}}$	1	$9.42 \cdot 10^{-4}$	1	$8.68 \cdot 10^{-4}$
	4	**1**	$\mathbf{2.33 \cdot 10^{-4}}$	1	$9.42 \cdot 10^{-4}$	1	$8.89 \cdot 10^{-4}$
	6	**1**	$\mathbf{2.33 \cdot 10^{-4}}$	1	$9.42 \cdot 10^{-4}$	1	$9.11 \cdot 10^{-4}$
	8	**1**	$\mathbf{2.33 \cdot 10^{-4}}$	1	$9.42 \cdot 10^{-4}$	1	$8.92 \cdot 10^{-4}$
0.0001	2	**2**	$\mathbf{3.31 \cdot 10^{-5}}$	10	$3.67 \cdot 10^{-3}$	10	$1.59 \cdot 10^{-3}$
	4	**2**	$\mathbf{3.31 \cdot 10^{-5}}$	10	$3.67 \cdot 10^{-3}$	10	$5.12 \cdot 10^{-3}$
	6	**2**	$\mathbf{3.31 \cdot 10^{-5}}$	10	$3.67 \cdot 10^{-3}$	10	$2.54 \cdot 10^{-3}$
	8	**2**	$\mathbf{3.31 \cdot 10^{-5}}$	10	$3.67 \cdot 10^{-3}$	10	$3.39 \cdot 10^{-3}$

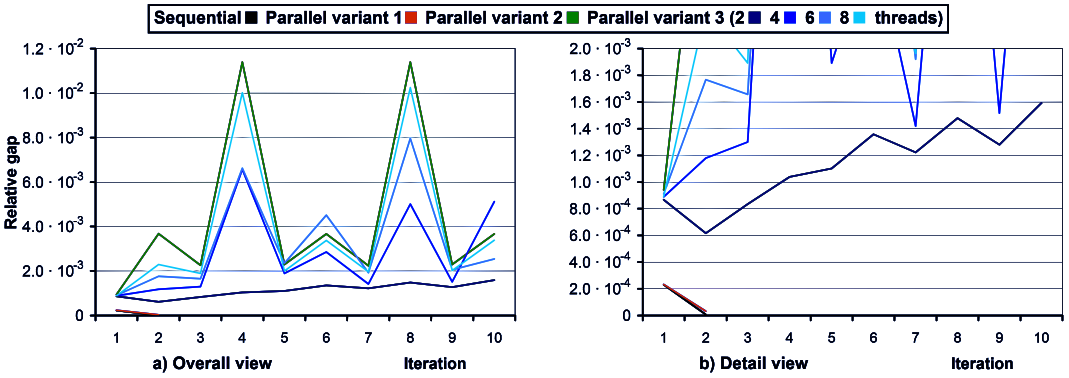


Fig. 6. The trend of the relative gap for network 1 (small) and the target relative gap rg_{max} of 0.0001.

It should be also noted that the relative gaps achieved by the parallel variant 1 (main) and 2 are not completely identical for the same inputs but various numbers of working threads. A closer look on the values for the main parallel variant and the target relative gap of 0.0001 reveals that there are slight differences in order of 10^{-15} (see Table 5). These differences are caused by a limited precision of the `double` data type used for the computations. The sums of the objective function parts and of the AON flows parts (see Fig. 2) are calculated from different number of summands depending on the number of working threads. These sums should be mathematically identical, but the rounding errors due to the limited precision cause these slight differences. Since they are 10 orders of magnitude lower than the lowest target relative gap, they are completely negligible.

The achieved relative gaps for the network 2 (large) are summarized in Table 6 for the sequential version and in Table 7 for all the parallel variants. We can observe that the slower convergence of the main parallel variant in comparison to the sequential version is more prominent (see Fig. 4b and Tables 6 and 7) – it requires more iterations to achieve similar results. The ratio of the parallel iterations count to the sequential iterations count increases with the decreasing target relative gap rg_{max}. It is 1.50 for rg_{max} of 0.01, 1.69 for rg_{max} of 0.001, and 2.75 for rg_{max} of 0.0001. This trend is not desirable, but since the rg_{max} of 0.0001 is considered sufficient for many applications [15,20], it is not so problematic. Even with the additional iterations, the main parallel variant reaches significant speedup (see Section 5.3). As expected, the number of working threads does not influence the results (see Table 7), similarly to the network 1 (small).

The remaining two parallel variants were unable to reach even the highest target relative gap rg_{max} of 0.01. Thus, the preset maximum of 75 iterations was always performed and the computations with the same input settings were identical regardless of the rg_{max} value. For the parallel variant 2, the relative gap oscillates and the best value is achieved in first iteration similarly to the small road traffic network (see Fig. 7a). Again, the number of working threads does not influence the results (see Table 7). On the other hand, for the parallel variant 3, the achieved relative gap rg_f depends on the number of working threads. The best value is achieved for two working threads (see Table 7), similarly to the network 1 (see above). Unlike the network 1 (small), the achieved relative gap consistently increases with the increasing number of working threads, which is an expected behavior. As can be seen in Fig. 7, the relative gap oscillates similarly to the parallel variant 2, but the iteration, in which the best value of relative gap is achieved, varies with the varying number of working threads.

Table 5
Differences of relative gaps achieved by the parallel variant 1 (main) for various threads counts.

rg_{max}	Threads (T)	k_f	rg_f
0.0001	2	2	0.0000331180381**22366**
	4	2	0.0000331180381**31401**
	6	2	0.0000331180381**33739**
	8	2	0.0000331180381**38167**

Table 6
Relative gap (rg_f) in last iteration (k_f) for network 2 (large) for the sequential version.

rg_{max}	k_f	rg_f
0.01 (10^{-2})	6	$9.38 \cdot 10^{-3}$
0.001 (10^{-3})	13	$6.59 \cdot 10^{-4}$
0.0001 (10^{-4})	27	$9.48 \cdot 10^{-5}$

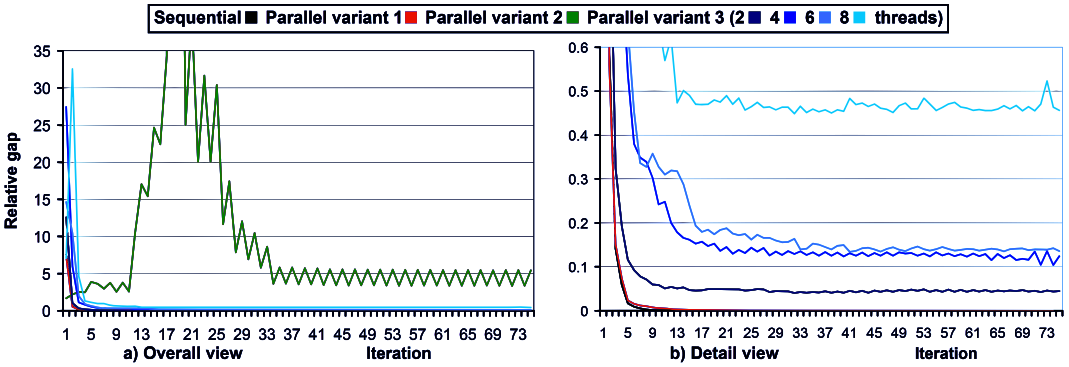


Fig. 7. The trend of the relative gap for network 2 (large) and the target relative gap rg_{max} of 0.0001.

Table 7
Relative gap (rg_f) in last iteration k_f for network 2 (large) for the parallel variants.

rg_{max}	Threads (T)	Variant 1 (main)		Variant 2		Variant 3	
		k_f	rg_f	k_f	rg_f	k_f	rg_f
0.01	2	**9**	$\mathbf{7.48 \cdot 10^{-3}}$	75	5.46	75	$4.50 \cdot 10^{-2}$
	4	**9**	$\mathbf{7.48 \cdot 10^{-3}}$	75	5.46	75	$1.24 \cdot 10^{-1}$
	6	**9**	$\mathbf{7.48 \cdot 10^{-3}}$	75	5.46	75	$1.36 \cdot 10^{-1}$
	8	**9**	$\mathbf{7.48 \cdot 10^{-3}}$	75	5.46	75	$4.56 \cdot 10^{-1}$
0.001	2	**22**	$\mathbf{9.14 \cdot 10^{-4}}$	75	5.46	75	$4.50 \cdot 10^{-2}$
	4	**22**	$\mathbf{9.14 \cdot 10^{-4}}$	75	5.46	75	$1.24 \cdot 10^{-1}$
	6	**22**	$\mathbf{9.14 \cdot 10^{-4}}$	75	5.46	75	$1.36 \cdot 10^{-1}$
	8	**22**	$\mathbf{9.14 \cdot 10^{-4}}$	75	5.46	75	$4.56 \cdot 10^{-1}$
0.0001	2	**74**	$\mathbf{9.05 \cdot 10^{-5}}$	75	5.46	75	$4.50 \cdot 10^{-2}$
	4	**74**	$\mathbf{9.05 \cdot 10^{-5}}$	75	5.46	75	$1.24 \cdot 10^{-1}$
	6	**74**	$\mathbf{9.05 \cdot 10^{-5}}$	75	5.46	75	$1.36 \cdot 10^{-1}$
	8	**74**	$\mathbf{9.05 \cdot 10^{-5}}$	75	5.46	75	$4.56 \cdot 10^{-1}$

Although the parallel variant 3 achieves significantly better results than the parallel variant 2 (see Fig. 7), it is still significantly worse than the main parallel variant. Hence, we can conclude that, from the precision point of view, the parallel variants 2 and 3 are not practically utilizable.

5.3. Computation time results

Since the main goal of the parallelization of the B algorithm was to speed up the computation, the resulting computation time τ_f is the vital parameter. To determine the computation times, the test were performed on both computers – the HW1 and HW2. For the network 1 (small) on the HW1, the results are summarized in Table 8 for the sequential version and in Table 9 for all the parallel variants. The breakdown of the computation time into parallel time, sequential time, and time spent on barrier is summarized in Table 10.

The main parallel variant is significantly faster than the sequential version for 2 working threads (speedup up to 1.62) and even faster for 4 working threads (speedup up to 2.25). However, there is only a little additional speedup for 6 working threads (speedup up to 2.38) and no additional speedup for 8 working threads. The reason is that the overhead associated with the interactions of the threads is increasing with the increasing number of threads. Each barrier synchronization can potentially last longer with a higher number of synchronized threads and the sums of the objective function parts and of the AON flows parts (see Fig. 2) are calculated from a larger number of summands. The increase of the time per barrier synchronization can be observed in Table 10. Since the computation for the small road traffic network is relatively fast, this overhead makes a non-negligible part of the computation time (from 10 % to 26 % – see Table 10), which partially compensates the speedup gained by using 6 and 8 threads. Another possible influence could be that the HW1 has 8 logical cores due to the Hyper-threading, but only 4 physical cores. Hence, the same computing power as if there were 8 physical cores cannot be expected. Indeed, the parallel time for 6 and 8 working threads is only slightly lower than for 4 working threads (see Table 10). Moreover, the tests performed for the small road traffic network on the HW2 show similar results to the HW1, although there is 6 physical cores (see bellow). So, the major influence can be attributed to the overhead described above. Nevertheless, the tests performed for the network 2 (large) on the HW1 show noticeable additional speedup for 6 and 8 working threads (see below).

Table 8
Computation time for network 1 (small) for the sequential version on HW1.

rg_{max}	k_f	Time (τ_f) [ms]
0.01 (10^{-2})	1	1 713
0.001 (10^{-3})	1	1 691
0.0001 (10^{-4})	2	2 671

The remaining parallel variants 2 and 3 were unable to reach the target relative gap rg_{max} of 0.0001 (see Section 5.2). So, the computation was stopped when the maximal number of iteration k_{max} of 10 was reached. Hence, their computation times were worse then the computation time of the sequential version even for 8 threads. So, the variants 2 and 3 are unusable for rg_{max} of 0.0001 even for the small road traffic network. On the other hand, the variants 2 and 3 were able to reach the target relative gaps rg_{max} of 0.01 and 0.001 with the same number of iterations as the main parallel variant. They also achieved very similar computation times. Nevertheless, since the parallel flows shift employed in variant 2 and 3 was intended to reduce the computation time, a slight speedup in comparison to the main parallel variant was expected, but was virtually not observed.

Table 10 shows that the variants 2 and 3 spent much less time on sequential computations and barrier synchronizations than the variant 1, but their parallel time is higher. For variant 1, the time of sequential computation makes 13 % of the total computation time on average. For variants 2 and 3, it is less than 1 %. The barrier synchronization makes from 10 % to 26 % of the total computation time for the variant 1 and 3 % to 16 % for the variants 2 and 3. Yet, the increment of the parallel time in the variants 2 and 3 nearly compensates the decrement of the sequential time and of the barrier synchronization.

A possible reason for this observation is that both the parallel flows shifts in variant 2 and 3 employ additional storing of edge flow additions to local data structures – for individual bushes (variant 2) or for individual threads (variant 3). These additions are used by the Newton method (see Figs. 4 and 5), bringing additional arithmetic operations, which are not present in the main parallel variant. Additionally, the parallel flows shift in variant 2 and 3 incorporate sequential phase for adding of these stored edge flow additions to the total flows of the graph edges (see Sections 4.3 and 4.4). Again, this phase is not present in the main parallel variant. This phase is more time consuming in the parallel variant 2, since there are more addends to the sum (corresponding to the number of bushes for each edge) than in the parallel variant 3 (corresponding to the number of threads for each edge). Hence, the parallel variant 3 should be faster than the parallel variant 2. For the network 1 (small), this is barely observable, but the difference is more pronounced for the network 2 (large – see below) for all values of rg_{max}.

As it was stated above and can be observed in Table 10, the barrier synchronization makes from 10 % to 26 % of the total computation time of the main parallel variant. The sequential part makes consistently ca. 13 % of the computation time and do not increases

Table 9
Computation time for network 1 (small) for the parallel variants on HW1.

rg_{max}	Threads (T)	Variant 1 (main)		Variant 2		Variant 3	
		k_f	Time (τ_f) [ms]	k_f	Time (τ_f) [ms]	k_f	Time (τ_f) [ms]
0.01	2	**1**	**1 124**	1	1 129	1	1 107
	4	**1**	**886**	1	892	1	886
	6	**1**	**854**	1	849	1	871
	8	**1**	**908**	1	855	1	873
0.001	2	**1**	**1 126**	1	1 132	1	1 124
	4	**1**	**889**	1	914	1	889
	6	**1**	**861**	1	858	1	893
	8	**1**	**895**	1	870	1	855
0.0001	2	**2**	**1 644**	10	5 831	10	5 756
	4	**2**	**1 186**	10	3 738	10	3 616
	6	**2**	**1 123**	10	3 232	10	3 114
	8	**2**	**1 129**	10	2 827	10	2 720

with increasing number of threads. Hence, in an ideal parallel computing environment, without considering the synchronization time, the computation time of the main parallel variant would be by up to 26 % lower than in a real environment. This estimate is not based solely on the network 1 results. For the network 2, the results are very similar.

The dependency of the speedup achieved by the individual parallel variants in comparison to the sequential version on rg_{max} and the number of threads for the network 1 (small) is depicted in Fig. 8. The speedup below 1.0 for the parallel variants 2 and 3 in Fig. 8b and 8c shows the unusability of these variants for rg_{max} of 0.0001.

For the large road traffic network (network 2) on the HW1, the results are summarized in Table 11 for the sequential version and in Table 12 for all the parallel variants.

The parallel variants 2 and 3 were unable to reach even the highest target relative gap rg_{max} of 0.01 (see Section 5.2). So, the computation was stopped when the maximal number of iteration k_{max} of 75 was reached. Hence, their computation times were worse then the computation time of the sequential version in all instances making them (together with insufficient achieved relative gab) unusable for the large road traffic network. Nevertheless, we can observe that the parallel variant 3 is noticeably faster than the parallel variant 2 in all instances. This is caused by a more time

Table 10
Breakout of the computation times for network 1 (small) for the parallel variants on HW1.

Variant	rg_{max}	Threads (T)	Time [ms]				Barriers count
			Parallel	Sequential	Barrier total	Per barrier	
1 (main)	0.01	2	886	117	121	13	9
		4	584	127	175	19	9
		6	545	125	185	21	9
		8	572	119	217	24	9
	0.001	2	884	120	122	14	9
		4	623	119	147	16	9
		6	569	120	172	19	9
		8	539	122	234	26	9
	0.0001	2	1327	154	163	10	16
		4	820	157	209	13	16
		6	728	152	243	15	16
		8	683	155	291	18	16
2	0.01	2	1088	9	32	4	9
		4	825	10	57	6	9
		6	772	10	67	7	9
		8	747	11	97	11	9
	0.001	2	1093	9	30	3	9
		4	858	10	46	5	9
		6	785	10	63	7	9
		8	780	11	79	9	9
	0.0001	2	5490	42	299	4	72
		4	3252	45	441	6	72
		6	2736	42	454	6	72
		8	2327	42	458	6	72
3	0.01	2	1077	2	29	3	9
		4	825	3	58	6	9
		6	808	4	59	7	9
		8	754	5	115	13	9
	0.001	2	1085	2	37	4	9
		4	833	3	53	6	9
		6	822	4	67	7	9
		8	779	5	71	8	9
	0.0001	2	5438	13	305	4	72
		4	3255	12	349	5	72
		6	2716	12	386	5	72
		8	2337	13	370	5	72

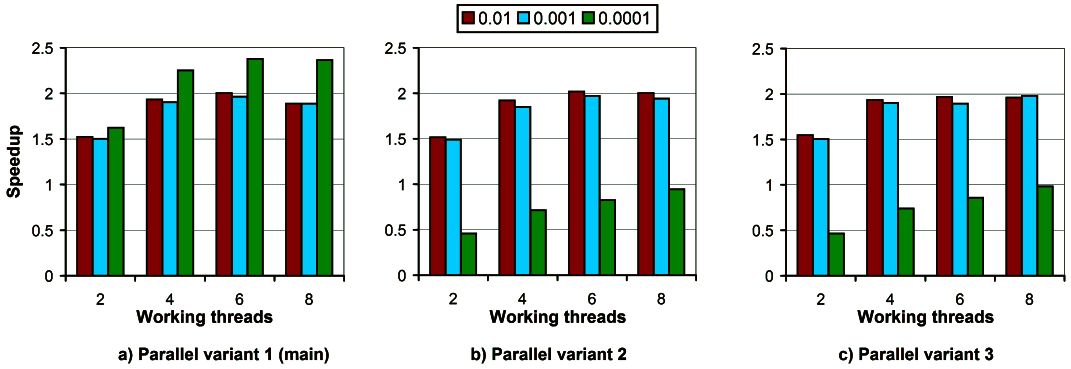


Fig. 8. Speedups of the parallel variants in comparison to the sequential version for network 1 on HW1.

Table 11
Computation time for network 2 (large) for the sequential version on HW1.

rg_{max}	k_f	Time (τ_f) [ms]
0.01	6	531 777
0.001	13	1 097 883
0.0001	27	2 232 585

consuming sequential phase for adding of edge flow differences to the total flows of the graph edges in the parallel variant 2 (see above).

Unlike the parallel variants 2 and 3, the main parallel variant was able to reach even the lowest target relative gap rg_{max} of 0.0001, but required higher number of iterations than the sequential version (see Section 5.2). Despite of this, it is faster than the sequential version in almost all instances, with only one exception – rg_{max} of 0.0001 and 2 working threads. In this case, the sequential version is faster, so the speedup is below 1.0. In all other instances, the speedup is above 1.0 and increases with the increasing number of working threads (see Fig. 9), which is a desired behavior. It should be also noted that, unlike the small road traffic network, utilization of 6 and 8 working threads brings noticeable additional speedup in comparison to 4 working threads, although there are only 4 physical cores. This confirms that a little or no additional speedup observed for the small road traffic network for 6 and 8 working threads were caused by the increasing overhead associated with the interactions of the threads. This effect is of course present for the large road traffic network as well. However, since the computation of the large road traffic network lasts comparatively very long, this overhead makes a substantially smaller part of the computation time than for the small road traffic network.

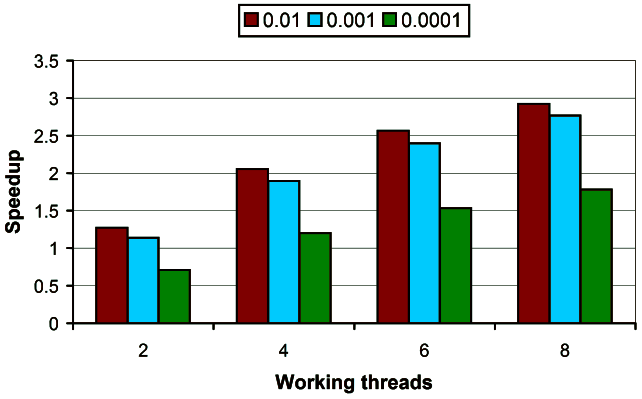


Fig. 9. Speedups of the main parallel variant in comparison to the sequential version for network 2 on HW1.

Hence, the effect is not so pronounced. The speedup achieved using 8 threads is 2.92 for rg_{max} of 0.01, 2.77 for rg_{max} of 0.001, and 1.78 for rg_{max} of 0.0001 (see Fig. 9).

The tests on the HW2 were performed only for the sequential version and the main parallel variant. The tests performed on the HW1 showed that the parallel variants 2 and 3 show insufficient convergence in most instances and worse or similar speedup as the main parallel variant in all instances. So, there is no reason to use them instead of the main parallel variant. The results for all three road traffic networks on the HW2 are summarized in Table 13 for the sequential version and in Table 14 for the main parallel variant.

Table 12
Computation time for network 2 (large) for the parallel variants on HW1.

rg_{max}	Threads	Variant 1 (main)		Variant 2		Variant 3	
		k_f	Time [ms]	k_f	Time [ms]	k_f	Time [ms]
0.01	2	**9**	**418 588**	75	3 678 970	75	3 182 508
	4	**9**	**258 598**	75	2 134 622	75	1 874 399
	6	**9**	**207 473**	75	1 783 638	75	1 573 132
	8	**9**	**181 937**	75	1 700 040	75	1 500 301
0.001	2	**22**	**964 403**	75	3 683 239	75	3 186 456
	4	**22**	**579 529**	75	2 134 697	75	1 876 210
	6	**22**	**457 652**	75	1 784 140	75	1 572 461
	8	**22**	**396 787**	75	1 702 769	75	1 499 138
0.0001	2	**74**	**314 3022**	75	3 690 537	75	3 189 929
	4	**74**	**1 855 870**	75	2 116 782	75	1 876 648
	6	**74**	**1 455 223**	75	1 743 547	75	1 578 967
	8	**74**	**1 253 008**	75	1 701 799	75	1 500 270

Table 13
Computation time for both networks for the sequential version on HW2.

Network	rg_f	k_f	Time (τ_f) [ms]
Network 1 (small)	0.01	1	2 906
	0.001	1	2 930
	0.0001	2	4 294
Network 2 (large)	0.01	6	621 952
	0.001	13	1 260 125
	0.0001	27	2 538 783
Network 3 (large)	0.01	1	41 192
	0.001	2	70 223
	0.0001	5	153 366

The results on the HW2 are similar to the HW1. For the network 1 (small), the main parallel variant is faster than the sequential version in all instances. Similarly to the HW1, the maximal speedup is achieved for 6 working threads (speedup up to 2.49), but there is only a little difference between 4 and 6 threads and there is no additional speedup for 8, 10, and 12 threads (see Fig. 10). This again confirms that the main cause is the increasing overhead associated with the interactions of the threads.

For the network 2 (large), the main parallel variant is faster than the sequential version in almost all instances, with only one exception – rg_{max} of 0.0001 and 2 working threads, similarly to the HW1. In all other instances, the speedup is above 1.0 and increases with the increasing number of working threads (see Fig. 11). The maximal achieved speedup is for 12 working threads (up to 3.53).

The network 3 (large) was not tested on the HW1. On the HW2, its speedup in comparison to the sequential version is always above 1.0 and increases with the increasing number of working threads (see Fig. 12). The maximal achieved speedup is for 12 working threads, similar to networks 2 (up to 4.32).

Table 14
Computation time for all three networks for the main parallel variant on HW2.

Network	rg_f	Threads (T)	k_f	Time (τ_f) [ms]
Network 1 (small)	0.01	2	1	2 041
		4	1	1 447
		6	1	1 437
		8	1	1 437
		10	1	1 496
		12	1	1 443
	0.001	2	1	2 081
		4	1	1 447
		6	1	1 438
		8	1	1 459
		10	1	1 459
		12	1	1 475
	0.0001	2	2	2 894
		4	2	1 859
		6	2	1 736
		8	2	1 722
		10	2	1 759
		12	2	1 726
Network 2 (large)	0.01	2	9	490 554
		4	9	286 587
		6	9	223 143
		8	9	200 453
		10	9	186 401
		12	9	176 400
	0.001	2	22	1 131 726
		4	22	632 535
		6	22	465 010
		8	22	422 372
		10	22	385 714
		12	22	360 701
	0.0001	2	74	3 710 248
		4	74	1 965 092
		6	74	1 420 514
		8	74	1 312 271
		10	74	1 180 246
		12	74	1 080 896
Network 3 (large)	0.01	2	1	26 103
		4	1	18 668
		6	1	14 477
		8	1	12 779
		10	1	12 247
		12	1	11 722
	0.001	2	3	56 666
		4	3	38 704
		6	3	28 224
		8	3	25 877
		10	3	23 813
		12	3	22 889
	0.0001	2	6	97 588
		4	6	64 169
		6	6	46 372
		8	6	42 977
		10	6	38 408
		12	6	35 511

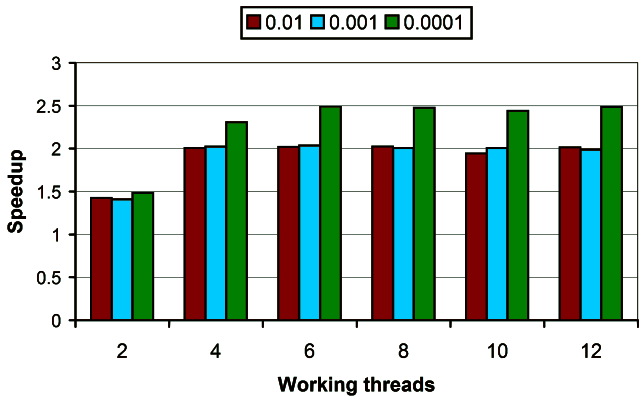


Fig. 10. Speedups of the main parallel variant in comparison to the sequential version for network 1 on HW2.

5.4. Results discussion

The performed tests described in Section 5.1 to 5.3 shows that the convergence of the parallel variants 2 and 3 is insufficient in most instances. For the network 1 (small), these variants were able to achieve the target relative gap rg_{max} of 0.001. However, their computation times were similar to the computation times of the main parallel variant. The parallel variants 2 and 3 were unable to achieve the target relative gap rg_{max} of 0.0001 for the network 1 (small). For the network 2 (large), they were unable to achieve even the rg_{max} of 0.01. The inability to achieve the target relative gap caused that the computation was stopped when maximal number of iterations k_{max} was reached. So, the number of performed iterations was in most instances significantly higher than the number of iterations required by the main parallel version leading to substantially higher computation times. In total, there was not a single instance, in which the variant 2 and variant 3 showed better convergence and/or significantly better computation time. For this reason, there is no point of using them instead of the main parallel version.

The tests showed that the parallel flows shift utilized in the parallel versions 2 and 3 has too high negative impact on the convergence, while the speed gain in comparison to the main parallel variant is negligible if any. The main parallel variant, which utilizes the sequential flows shift, on the other hand, shows very good results for both the small and the large road traffic networks, although its convergence is slower than the convergence of the sequential version.

For the network 1 (small), the main parallel variant requires the same numbers of iterations as the sequential version for all values

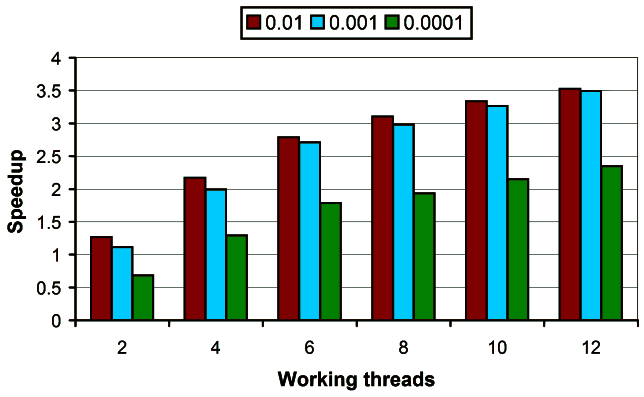


Fig. 11. Speedups of the main parallel variant in comparison to the sequential version for network 2 on HW2.

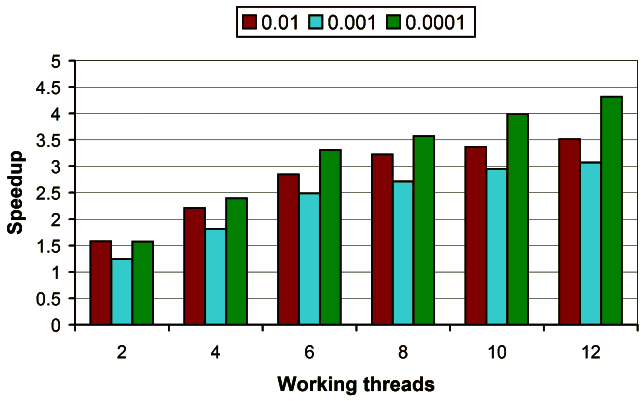


Fig. 12. Speedups of the main parallel variant in comparison to the sequential version for network 3 on HW2.

of rg_{max} and is faster in all instances. The maximal speedup (up to 2.49) was achieved for 6 working threads on both the HW1 and HW2 in most instances. There was no additional speedup gained by adding more working threads on either computer. This is caused by the increasing overhead associated with the interactions of the threads, which makes a nonnegligible part of the computation time for the small road traffic network.

For the network 2 (large), the main parallel variant requires a larger number of iterations than the sequential version, but is faster in all instances with one exception – rg_{max} of 0.0001 and 2 working threads. The speedup increased steadily with the increasing number of working threads, which is a desired behavior. Hence, the maximal speedup is achieved for the maximal number of working threads for all values of rg_{max}. For the HW1, the maximal speedup is achieved for 8 working threads (up to 2.92). For the HW2, the maximal speedup is achieved for 12 working threads (up to 3.53). It should be noted that, even for the large road traffic network, the increase of the speedup with the increasing number of working processes is sub-linear, which is caused by the increasing overhead associated with the interactions of the threads, similarly to the small road traffic network. This is an expected (though not desired) behavior. In should be also reminded that both the HW1 and HW2 incorporate the Hyperthreading technology, which means that there are only 4 and 6 physical cores for 8 and 12 logical cores, respectively. Hence, the same computing power as if there were 8 and 12 physical cores cannot be expected.

For the network 3 (large), which was tested on the HW2 only, the main parallel variant requires the same or only slightly larger number of iterations than the sequential version. It is faster in all instances. In fact, the highest speedup from all tests (up to 4.32) was achieved for the network 3 on HW2 (for 12 working threads).

It can be also observed that the sequential version (performed using one core) and the main parallel variant performed on the same number of cores is faster on the HW1. So, one core of the HW1 is faster than one core of the HW2, at least for the B algorithm computation. Nevertheless, the higher number of cores enables to perform the computation of the large road traffic network faster on the HW2. The computation of the small road traffic network is faster on the HW1, since addition of more working threads has no positive effect on the achieved speedup above 6 working threads (see above).

Overall, the main parallel variant offers significant computation time savings on both the HW1 and HW2. The highest achieved time savings are summarized in Table 15.

6. Conclusion and future work

In this paper, the parallelization of the B algorithm, a bush-based algorithm for the user equilibrium (UE) static traffic

Table 15
Highest computation time saving achieved by the main parallel variant.

Network	Computer	rg_{max}	Threads	Sequential time [ms]	Parallel time [ms]	Savings [%]
Network 1 (small)	HW1	0.01	6	1 713	855	50.09%
		0.001	6	1 691	861	49.06%
		0.0001	6	2 671	1 123	57.94%
	HW2	0.01	6	2 906	1 437	50.54%
		0.001	6	2 930	1 438	50.93%
		0.0001	6	4 294	1 722	59.90%
Network 2 (large)	HW1	0.01	8	531 777	181 937	65.79%
		0.001	8	1 097 883	396 787	63.86%
		0.0001	8	2 232 585	1 253 008	43.88%
	HW2	0.01	12	621 952	176 400	71.64%
		0.001	12	1 260 125	360 701	71.38%
		0.0001	12	2 538 783	1 080 896	57.42%
Network 3 (large)	HW2	0.01	12	41,192	11,722	71.54%
		0.001	12	70,223	22,889	67.40%
		0.0001	12	153,366	35,511	76.85%

assignment (STA) was thoroughly described. Three parallel variants were implemented. Although the parallelization was implemented and tested for a specific implementation of the B algorithm, its core idea is utilizable for other implementations as well.

The core idea is to enable partially independent processing of the bushes by recalculating the costs of edges only once per iteration rather than after each bush while preserving sequential shift of flows in bushes. This combination of features implemented in the main parallel variant (parallel variant 1) ensures both good convergence (although slightly worse than the convergence of the sequential version) and a significant speedup of the computation. The maximal achieved speedup was 4.32 using 12 working threads (6 physical, 12 logical cores), corresponding to computation time savings of nearly 77 %.

The remaining two parallel variants (variants 2 and 3) utilize two versions of parallel flows shift rather than a sequential one used in the main parallel variant. Nevertheless, they show insufficient convergence, especially for the large road traffic networks, which makes them unusable. However, they demonstrate that the shift of flows in bushes is the vital part of the B algorithm ensuring its fast convergence and cannot be parallelized easily.

In our future work, we will explore the possibilities to further improve the speed of the B algorithm. We will consider further options how to parallelize the shift of flows in bushes. We will also consider the adaptation of the main parallel variant for the distributed/parallel computing environment, where the computation runs as a set of multithreaded processes on multiple multi-core computers. This enables to utilize a large number of cores, but brings an additional overhead in the form of the inter-process communication via the message passing. Since the number of threads interactions per iteration in the main parallel variant of the B algorithm is relatively low, this approach can be viable, but only for very large road traffic networks (probably larger then the large road traffic network used in our tests).

We will also focus on possible optimization of individual parts of the B algorithm not related to parallelization to further reduce the computation time. For example, during the implementation of the sequential B algorithm, the replacement of the library power function with an iteration-based implementation reduced the computation time by ca. 70 %.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

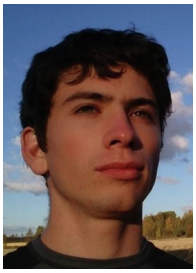
This work was supported by projects PoliVisu (Policy Development based on Advanced Geospatial Data Analytics and Visualisation, H2020-SC6-CO-CREATION-2017, grant agreement No. 769608) and TRAFFO (Innovative Approaches to Mathematical Traffic Modelling for Sustainable Development of Cities and Regions, The Technology Agency of the Czech Republic, program DOPRAVA 2020+, grant agreement No. CK01000096).

References

[1] Wardrop JG. Road Paper. Some Theoretical Aspects of Road Traffic Research. In: Proceedings of the Institute of Civil Engineers. p. 325–62.
[2] Dial RB. A path-based user-equilibrium traffic assignment algorithm that obviates path storage and enumeration. Transport Res Part B Methodolog 2006;40(10):917–36.
[3] Lotito PA. Issues in the implementation of the DSD algorithm for the traffic assignment problem. Eur J Oper Res 2006;175(3):1577–87.
[4] de Dios Ortúzar J, Willumsen LG. Modelling Transport. John Wiley & Sons; 2011.
[5] Perederieieva O, Ehrgott M, Raith A, Wang JYT. A framework for and empirical study of algorithms for traffic assignment. Comput Oper Res 2015;54:90–107.
[6] Jafari E, Pandey V, Boyles SD. Static traffic assignment: a decentralized approach. Proceedings of the 95nd Annual Meeting of Transportation Research Board, 2016.
[7] Jayakrishnan R, Tsai WK, Prashker JN, Rajadhyaksha S. A Faster Path-based Algorithm for Traffic Assignment. Transp Res Rec 1994;1443.
[8] Chen RJ, Meyer RR. Parallel optimization for traffic assignment. Math Program 1988;42:327–45.
[9] Damberg O, Migdalas A. Distributed Disaggregate Simplicial Decomposition — A Parallel Algorithm for Traffic Assignment. In: Network Optimization, Lecture Notes in Economics and Mathematical Systems. p. 172–93.
[10] Karakitsiou A, Mavrommati A, Migdalas A. Efficient minimization over products of simplices and its application to nonlinear multicommodity network problems. Oper Res Int Journal 2004;4(2):99–118.
[11] V. Buchhold, P. Sanders, and D. Wagner, "Real-time Traffic Assignment Using Engineered Customizable Contraction Hierarchies," ACM Journal of Experimental Algorithmics, Vol. 24, No. 2, 2019.
[12] Chen X, Liu Z, Kim I. A parallel computing framework for solving user equilibrium problem on computer clusters. Transportmetrica A: Transport Science 2020;16(3):550–73.
[13] Jafari E, Pandey V, Boyles SD. A decomposition approach to the static traffic assignment problem. Transport Res Part B: Methodolog 2017;105:270–96.
[14] Nie Y. A class of bush-based algorithms for the traffic assignment problem. Transport Res Part B Methodolog 2010;44(1):73–89.
[15] Boyce D, Asce M, Ralevic-Dekic B, Bar-Gera H. Convergence of Traffic Assignments: How Much is Enough? J Transp Eng 2004;130(1):49–55.
[16] Smith MJ. The existence, uniqueness and stability of traffic equilibria. Transport Res Part B Methodolog 1979;13(4):295–304.
[17] Dafermos S. "Traffic Equilibrium and Variational Inequalities. Transport Sci 1980;14(1):42–54.
[18] Bar-Gera H. Origin-Based Algorithm for the Traffic Assignment Problem. Transport Sci 2002;36(4):398–417.
[19] Gentile G. Solving a Dynamic User Equilibrium model based on splitting rates with Gradient Projection algorithms. Transport Res Part B Methodol 2016;92:120–47.

[20] Mitradjieva M, Lindberg PO. The Stiff Is Moving—Conjugate Direction Frank-Wolfe Methods with Applications to Traffic Assignment. Transport Sci May 2013;47(2):280–93. doi: https://doi.org/10.1287/trsc.1120.0409.
[21] Lee DH, Nie Y, Chen A. A conjugate gradient projection algorithm for the traffic assignment problem. Math Comput Modell 2003;37(7–8):863–78. doi: https://doi.org/10.1016/S0895-7177(03)00090-6.
[22] Cheng L, Xu X, Qiu S. Constrained newton methods for transport network equilibrium analysis. Tsinghua Sci Technol 2009;14(6):765–75. doi: https://doi.org/10.1016/S1007-0214(09)70147-6.
[23] Xie J, (Marco) Nie Y, Liu X. A Greedy Path-Based Algorithm for Traffic Assignment. Transp Res Rec 2018;2672(48):36–44. doi: https://doi.org/10.1177/0361198118774236.
[24] Xie J, Xie C. Origin-Based Algorithms for Traffic Assignment: Algorithmic Structure, Complexity Analysis, and Convergence Performance. Transp Res Rec 2015;2498(1):46–55. doi: https://doi.org/10.3141/2498-06.
[25] Xie J, (Marco) Nie Y. A New Algorithm for Achieving Proportionality in User Equilibrium Traffic Assignment. Transportation Science 2019;53(2):566–84. doi: https://doi.org/10.1287/trsc.2018.0845.
[26] Transportation Networks for Research Core Team. Transportation Networks for Research. https://github.com/bstabler/TransportationNetworks. Accessed 2021-04-29.

Tomas Potuzak was born in 1983 in Sušice, Czech Republic, Europe. He went to University of West Bohemia (UWB) where he studied software engineering and obtained his degree in 2006. Then, he entered Ph.D. studies at the Department of Computer Science and Engineering (DCSE) at the same university and has worked on issues of distributed simulation of road traffic. He obtained his Ph.D. in 2009. He is now a senior lecturer at the DCSE UWB. His research is focused on the issues of road traffic simulations and software testing.



Frantisek Kolovsky went to the secondary school of Civil engineering (Geodesy). He graduated at the University of West Bohemia (UWB) where he studied Geomatics. Then, he started to study Ph.D. at the Department of Geomatics at the same university. His research is focused on time-dependent shortest path search and transportation modeling, especially on dynamic traffic assignment.