КОМПЬЮТЕРНЫЕ ИССЛЕДОВАНИЯ
И МОДЕЛИРОВАНИЕ 2022 Т. 14 № 2 С. 343–355
DOI: 10.20537/2076-7633-2022-14-2-343-355

Ки&М

МАТЕМАТИЧЕСКИЕ ОСНОВЫ И ЧИСЛЕННЫЕ МЕТОДЫ МОДЕЛИРОВАНИЯ

УДК: 519.8

Ускорение работы двухстадийной модели равновесного распределения потоков по сети

Е. В. Котлярова[a], П. А. Северилов[b], Я. П. Ивченков[c], П. В. Мокров[d], М. О. Чеканов[e], Е. В. Гасникова[f], Ю. И. Шароватова[g]

Национальный исследовательский университет «Московский физико-технический институт»,
Россия, 141701, г. Долгопрудный, Институтский пер., д. 9

E-mail: [a] kotlyarova.ev@phystech.edu, [b] severilov.pa@phystech.edu, [c] ivchenkov.yap@phystech.edu, [d] mokrov.pv@phystech.edu, [e] chekanov.mo@phystech.edu, [f] egasnikov@yandex.ru, [g] julia.i.moroz@gmail.com

Получено 20.01.2022.
Принято к публикации 13.02.2022.

В работе приведены возможные улучшения двухстадийной модели равновесного распределения транспортных потоков, повышающие качество детализации моделирования и скорость вычисления алгоритмов. Модель состоит из двух блоков, первый блок — модель расчета матрицы корреспонденций, второй блок — модель равновесного распределения транспортных потоков по путям. Равновесием в двухстадийной модели транспортных потоков называют неподвижную точку цепочки из этих двух моделей. Более подробно теория и эксперименты по данной модели были описаны в предыдущих работах авторов. В этой статье в первую очередь рассмотрена возможность сокращения вычислительного времени алгоритма расчета кратчайших путей (в модели стабильной динамики, равновесно распределяющей потоки). В исходном варианте эта задача была выполнена с помощью алгоритма Дийкстры, но, так как после каждой итерации блока распределения транспортных потоков, время, требующееся для прохода по ребру, изменяется не на всех ребрах (и если изменяется, то очень незначительно), во многом этот алгоритм был избыточен. Поэтому были проведены эксперименты с более новым методом, учитывающим подобные особенности, и приведен краткий обзор других ускоряющих подходов для будущих исследований. Эксперименты показали, что в некоторых случаях использование выбранного T-SWSF-алгоритма действительно сокращает вычислительное время. Во вторую очередь в блоке восстановления матрицы корреспонденций алгоритм Синхорна был заменен на алгоритм ускоренного Синхорна (или AAM-алгоритм), что, к сожалению, не показало ожидаемых результатов, расчетное время не изменилось. И наконец, в третьем и финальном разделе приведена визуализация результатов экспериментов по добавлению платных дорог в двухстадийную модель, что помогло сократить количество перегруженных ребер в сети. Также во введении кратко описана мотивация данных исследований, приведено описание работы двухстадийной модели, а также на маленьком примере с двумя городами разобрано, как с ее помощью выполняется поиск равновесия.

Ключевые слова: модель расчета матрицы корреспонденций, многостадийная модель, модель равновесного распределения потоков по путям

Исследование Е. В. Гасниковой было выполнено при поддержке Министерства науки и высшего образования Российской Федерации (госзадание), № 075-00337-20-03, номер проекта 0714-2020-0005.

COMPUTER RESEARCH AND MODELING
2022 VOL. 14 NO. 2 P. 343–355
DOI: 10.20537/2076-7633-2022-14-2-343-355

Ки&М

MATHEMATICAL MODELING AND NUMERICAL SIMULATION

UDC: 519.8

Speeding up the two-stage simultaneous traffic assignment model

E. V. Kotliarova[a], P. A. Severilov[b], Ya. P. Ivchenkov[c], P. V. Mokrov[d], M. O. Chekanov[e], E. V. Gasnikova[f], Yu. I. Sharovatova[g]

National Research University Moscow Institute of Physics and Technology,
9 Institutskiy per., Dolgoprudny, Moscow region, 141701, Russia

E-mail: [a] kotlyarova.ev@phystech.edu, [b] severilov.pa@phystech.edu, [c] ivchenkov.yap@phystech.edu, [d] mokrov.pv@phystech.edu, [e] chekanov.mo@phystech.edu, [f] egasnikov@yandex.ru, [g] julia.i.moroz@gmail.com

Received 20.01.2022.
Accepted for publication 13.02.2022.

This article describes possible improvements for the simultaneous multi-stage transport model code for speeding up computations and improving the model detailing. The model consists of two blocks, where the first block is intended to calculate the correspondence matrix, and the second block computes the equilibrium distribution of traffic flows along the routes. The first block uses a matrix of transport costs that calculates a matrix of correspondences. It describes the costs (time in our case) of travel from one area to another. The second block presents how exactly the drivers (agents) are distributed along the possible paths. So, knowing the distribution of the flows along the paths, it is possible to calculate the cost matrix. Equilibrium in a two-stage traffic flow model is a fixed point of a sequence of the two described models. Thus, in this paper we report an attempt to influence the calculation speed of Dijkstra's algorithm part of the model. It is used to calculate the shortest path from one point to another, which should be re-calculated after each iteration of the flow distribution part. We also study and implement the road pricing in the model code, as well as we replace the Sinkhorn algorithm in the calculation of the correspondence matrix part with its faster implementation. In the beginning of the paper, we provide a short theoretical overview of the transport modelling motivation; we discuss current approaches to the modelling and provide an example for demonstration of how the whole cycle of multi-stage transport modelling works.

Keywords: correspondence matrix calculation model, multi stage model, equilibrium distribution model of traffic flow

Citation: *Computer Research and Modeling*, 2022, vol. 14, no. 2, pp. 343–355.

The research of E. V. Gasnikova is supported by the Ministry of Science and Higher Education of the Russian Federation (Goszadaniye) No. 075-00337-20-03, project No. 0714-2020-0005.

Introduction

Careless planning of transport routes in a city can lead to traffic jams. Although in the well-known Braess' paradox [Braess, 1968] it is described that new roads could do more harm than good, many car owners and city planning specialists still do believe that, if one more lane is added to a road, traffic jams will disappear. Fortunately, today city authorities and transport modelling mathematicians are in a unique situation: in recent years the amount of available transport information has increased significantly! Almost all drivers are using the Google Maps or Yandex Maps phone applications during a trip and their GPS-tracks could be easily used for transport modelling and improving the situation with traffic jams.

Let the city road network be represented by a directed graph $G = (V, E)$, where vertices V correspond to intersections or centroids [Sheffi, 1984] and edges E correspond to roads, respectively. Let's define the edges of graph as $e \in E$. Edge travel time t_e, distance, etc., can be considered as the travel demands (weights of the edges). A number of people (vehicles) that are going from the Origin (or source, $O \subseteq V$) to the Destination (or terminal, $D \subseteq V$) or way back called the correspondences. People's behaviour was described by John Glen Wardrop [Wardrop, 1952] (so-called Wardrop principles). According to the first Wardrop principle, agents choose the shortest route and consider their contribution to the change of the general traffic situation to be zero. Then, every agent knows the current time costs (or any other costs) on all edges of the graph (for example, they are using Google/Yandex Maps). According to the second Wardrop principle, the social equilibrium (Wardrop equilibrium) in such a system will be a distribution of agents along the routes in which none of the traffic participants will profitably deviate from the chosen strategy (route). At the equilibrium, the travel time for every agent is minimal!

Thus, finding the equilibrium is very important for city planning. The widely used iterative multi-stage transport modelling [Gasnikov et al., 2013] is quite questionable as the convergence of the method is not guaranteed and there are computational problems with overfitting and rounding errors [Gasnikov, Gasnikova, 2020].

So, in this paper we use another simultaneous model for two-stage modelling previously described and investigated in [Kotliarova et al., 2020]. Let's briefly recall the basics of the model.

1. First, the vectors L_i and W_j are calculated from raw data: $\sum\limits_{j:\,(i,\,j)\in OD} d_{ij} = L_i$, $\sum\limits_{i:\,(i,\,j)\in OD} d_{ij} = W_j$, with normalization $\sum\limits_{i\in O} L_i = \sum\limits_{j\in D} W_j = 1$. Here we know only L_i, $i \in O$, and W_j, $j \in D$, while the exact correspondence matrix d_{ij}, $(i,\ j) \in OD$, is not given.

2. Then the entire correspondence matrix $\{d_{ij}\}_{(i,\,j)\in OD}$ is reconstructed from the vectors L_i, W_j and the cost matrix $\{T_{ij}\}_{(i,\,j)\in OD}$. This was implemented in the code using the entropy model [Wilson, 2012]: the correspondence matrix $d(T)$ is understood as a certain method for calculating the set of correspondences $\{d_{ij}\}_{(i,\,j)\in OD}$ using the known cost matrix $\{T_{ij}\}_{(i,\,j)\in OD}$.

3. The third step transport flows $\{f_e\}_{e\in E}$ and transport costs $\{t_{ij}\}_{(i,\,j)\in OD}$ are calculated from the reconstructed correspondence matrix $\{d_{ij}\}_{(i,\,j)\in OD}$. We used the Beckmann model for numerical experiments. But unlike the previous step, that model makes only one iteration! This is one of the key differences between simultaneous and iterative multi-stage models.

4. The outputs from the previous step are the flows $\{f_e\}_{e\in E}$ and the transport costs $\{t_{ij}\}_{(i,\,j)\in OD}$, but the input in the block restoring the correspondence matrix is the "full" path cost $T(t)$, which is calculated through the Dijkstra algorithm.

5. After that the model is looping between the second and the fourth step: first, there are many iterations of the algorithm for restoring the correspondence matrix; then one iteration of the Beckmann model; calculation of the shortest paths, and so on until the stable point is found.

In order to illustrate the work flow of the algorithm in practice and the double-check the correctness of the code implementation, we have implemented a couple of simple example graphs. Let us demonstrate one of them: suppose there is a town with two districts 1 and 2. Transport flows from one district to another are distributed along graph edges s, l and a circle trajectory, which describes agents travelling inside one district, thus giving us six edges in total. From the task definition we know only the vectors $\sum\limits_{j:\,(i,j)\in OD} d_{ij} = L_i$ and $\sum\limits_{i:\,(i,j)\in OD} d_{ij} = W_j$, while the exact correspondence matrix d_{ij}, $(i, j) \in OD$, is not given. In this particular example the vectors L_i, $i \in O$, and W_j, $j \in D$, show us the total number of people that leave district 1, 2 or arrive at them. Let us set values $W_1 = W_2 = L_1 = L_2 = 4000$ veh/hour. Also, from the task definition we know the free flow time $\bar{t}_e$ and road capacity $\bar{f}_e$ values. They can be found in Fig. 1.

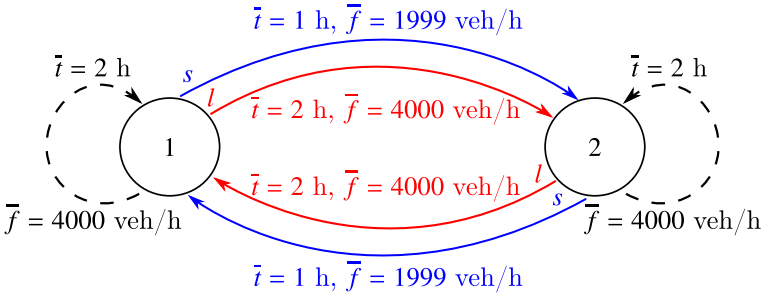

Figure 1. An illustration for the example with two cities. Colors and arrows indicate different oriented graph edges carrying the transport flows: s_{12} and s_{21} (blue), l_{12} and l_{21} (red), circle trajectories (dashed black lines)

From the definition of the vectors L_i and W_j, the next conditions should be satisfied simultaneously:

$$\begin{cases} d_{11} + d_{12} = L_1 = 4000, \\ d_{22} + d_{21} = L_2 = 4000, \\ d_{11} + d_{21} = W_1 = 4000, \\ d_{22} + d_{12} = W_2 = 4000. \end{cases}$$

As can be seen after solving the system of equations, in the equilibrium the values will be as follows: $d_{11} = d_{12} = d_{21} = d_{22} = 2000$ veh/h, thus we have 2000 agents travelling inside district 1 and the same number for district 2, and there are 2000 agents moving from 1 to 2 (and as many moving in the opposite direction). To calculate the distribution of agents along the edges in equilibrium, we need to consider the following principles in the model of stable dynamics: in equilibrium if $f_e \leqslant \bar{f}_e$, $e \in E$, then $t_e = \bar{t}_e$, meaning that if the flow along an edge is smaller than its capacity, all agents are moving with the fastest possible speed (e. g. with minimal time costs); otherwise the costs can be indefinitely high. According to the first Wardrop principle, the agents always choose the shortest path (with minimal time costs), so having two options of travelling along the edges s ($\bar{t}_s = 1$ h) or l ($\bar{t}_l = 2$ h), they will be choosing the s edge, until its capacity is full ($f_s = \bar{f}_s = 1999$). The only agent left cannot choose the same path as the costs would increase (a traffic jam would appear), so he/she chooses the l edge. Another 2000 agents which do not leave the district freely occupy the circle trajectory as its capacity is high enough. Then we can calculate the flow values f_e: trivially, $f_{11} = f_{22} = 2000$ veh/h, $t_{11} = t_{22} = 2$ hours. For the rest of the graph it will be $t^s_{12} = t^s_{21} = \bar{t}^l_{12} = \bar{t}^l_{21} = 2$ hours, $f^s_{12} = f^s_{21} = 1999$ veh/h,

$f_{12}^l = f_{21}^l = 1$ veh/h. As mentioned previously, that solution also could be obtained with that paper's code implementation.

Sinkhorn algorithm updates

The evolutionary reasoning of solving the entropy-linear programming problem for calculating the correspondence matrix is described in [Gasnikov et al., 2016]. It is important to note that the problem of reconstructing the matrix of correspondences can be written as the following optimization problem:

$$\min_{d_{ij} \in Q} f(d_{ij}) := \gamma \sum_{i,\, j=1,\, 1}^{n,\, n} d_{ij} T_{ij} + \sum_{i,\, j=1,\, 1}^{n,\, n} d_{ij} \ln d_{ij}, \tag{1}$$

where Q is defined as (2) and $T_{ij} := T_{ij}(\eta)$ is the cost function that depends on the vector of parameters η:

$$Q = \left\{ d_{ij} \geqslant 0 \colon \sum_{i,\, j=1}^{n,\, n} d_{ij} = N,\ \sum_{j=1}^{n} d_{ij} = L_i,\ \sum_{i=1}^{n} d_{ij} = W_j,\ i,\ j = 1, \ldots, n \right\}. \tag{2}$$

Let us introduce the following normalization: $\sum\limits_{i,\, j=1,\, 1}^{n,\, n} d_{ij} = 1$, the limits can be rewritten as $\sum\limits_{j=1}^{n} d_{ij} = L_i$ and $\sum\limits_{i=1}^{n} d_{ij} = W_j$, where $L_i = \frac{L_i}{N}$ $W_j = \frac{W_j}{N}$ and let's define the set $\widetilde{Q}$:

$$\widetilde{Q} = \left\{ d_{ij} \geqslant 0 \colon \sum_{i,\, j=1}^{n,\, n} d_{ij} = 1,\ \sum_{j=1}^{n} d_{ij} = L_i,\ \sum_{i=1}^{n} d_{ij} = W_j,\ i,\ j = 1, \ldots, n \right\}.$$

The problem (1) can be rewritten in the next equivalent form after normalization:

$$\min_{d_{ij} \in \widetilde{Q}} \gamma \sum_{i,\, j=1,\, 1}^{n,\, n} d_{ij} T_{ij} + \sum_{i,\, j=1,\, 1}^{n,\, n} d_{ij} \ln d_{ij}. \tag{3}$$

Next, let us introduce two blocks of the dual variables as $\lambda^l \in \mathbb{R}^n$ and $\lambda^w \in \mathbb{R}^n$, where λ_i^l is a multiplier of the limit $\sum\limits_{j=1}^{n} d_{ij} = L_i$ and λ_j^w is a multiplier of the limit $\sum\limits_{i=1}^{n} d_{ij} = W_j$. After applying the method of Lagrange multipliers for solving the problem (3), which was described in detail in [Ivanova et al., 2020], the following minimization problem is obtained:

$$\min_{\lambda^l,\, \lambda^w} \varphi(\lambda^l,\, \lambda^w) := \ln\left(\mathbf{1}^T B(\lambda^l,\, \lambda^w)\mathbf{1}\right) - \langle \lambda^l,\, l \rangle - \langle \lambda^w,\, w \rangle, \tag{4}$$

where $B_{ij}(\lambda^l,\, \lambda^w) = \exp(-\gamma T_{ij} + \lambda_i^l + \lambda_j^w)$.

To solve the problem (4) one could use the method described in [Guminov et al., 2020], which is based on the implementation of the Sinkhorn algorithm [Cuturi, 2013], as was done in our previous work [Kotliarova, Yarmoshik, 2020]. Using the normalization part $\sum\limits_{i,\, j=1,\, 1}^{n,\, n} d_{ij} = 1$ is crucial for correct work. The procedure is presented as a pseudo-code, see Algorithm 2 in the Appendix.

A logical way to improve the two-stage model is to replace the Sinkhorn algorithm with a faster method, for example, accelerated alternating minimization (AAM) or Greenhorn [Guminov et al., 2020]. We chose AAM, as previously it has shown better results [Ivanova et al., 2020]. The algorithm is schematically depicted as a pseudo-code (see Algorithm 3 in the Appendix).

The AAM-algorithm was used in numerical experiments [Severilov et al., 2021; Nartsev, Fesiuk, Ibraev, 2021]. The results did not show any significant advantage of the accelerated algorithm as compared to the standard one. As shown in Fig. 2, the restored matrices are almost the same for both methods. Thus, we can assume that the algorithm was implemented in the two attached codes correctly. In Fig. 3 iterations are marked along the horizontal axis and the duality gap estimation is marked on the vertical axis for the simultaneous multi-stage models both with standard and accelerated Sinkhorn algorithm. Clearly, the values are exactly the same.

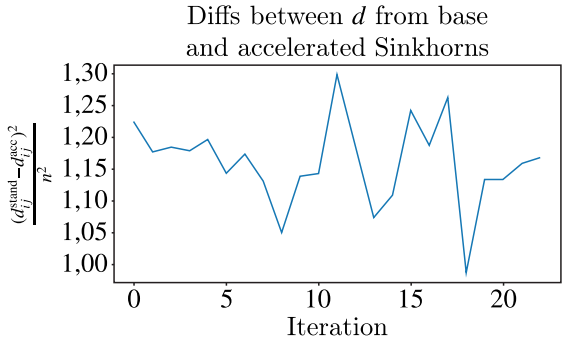

Figure 2. Difference between the correspondence matrix restored by the standard Sinkhorn and the accelerated Sinkhorn algorithm variants $\frac{(d_{ij}^{\text{stand}} - d_{ij}^{\text{acc}})^2}{n^2}$. The restored matrices are almost the same for both methods

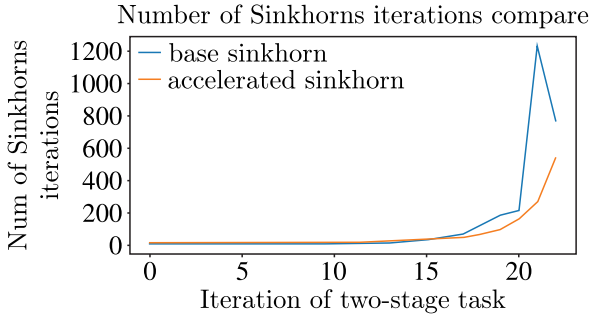

Figure 3. Difference between criterion values of the model based on standard and accelerated Sinkhorn algorithms. Iterations are marked along the horizontal axis and the duality gap estimation is marked on the vertical axis for the simultaneous multi-stage models both with standard (blue) and accelerated (orange) Sinkhorn algorithm. Again, the values are exactly the same

Experiments with dynamic single-source shortest-path algorithms

Another idea to shorten the computation time is to utilize the Dijkstra algorithm for performing the flow reconstruction phase (see Algorithm 4 in Appendix) [Kubentayeva, Gasnikov, 2020]. After every step of the Beckman model it is necessary to update the shortest paths, and every time the Dijkstra algorithm works independently of previous runs. However, the cost-matrix $\{T_{ij}\}_{(i,\,j)\in OD}$ is changing a little, and for some edges $e \in E$ the costs may not change at all. Algorithms that update the trees without a full recomputation from scratch are called dynamic single-source shortest-path algorithms. Such algorithms slightly differ in the type of their output. Some store only distances from the source, while others additionally store the shortest-path tree or the shortest-path subgraph. Some of the algorithms known in the literature are only able to cope with the update of one edge at a time, while others can perform batch updates, i.e. update the shortest-path information after multiple edges have simultaneously changed their weight [Bauer, Wagner, 2009]. Batch updates naturally arise in real-world

applications: traffic jams usually affect a set of edges; updating information in sensor networks usually requires flooding, which is done in intervals big enough so that more than one link in the network has changed. [Bauer, Wagner, 2009]. Below we provide a brief overview of a few studies.

In [Sunite, Deepak, 2018] the authors dynamized the Dijkstra algorithm by using a retroactive priority queue data structure. The retroactive data structure identifies a set of affected vertices step by step and thus helps to accommodate the changes in the least number of computations. So, using a suitable dynamic graph representation and retroactive priority queue, the authors proposed a method to dynamize the Dijkstra algorithm giving the solution of a dynamic single source shortest path problem. In [Bergamini et al., 2014], the authors used the idea of betweenness. Betweenness centrality ranks the importance of nodes by their participation in all shortest paths of the network. But computing exact betweenness values is impractical in large networks. For static networks, an approximation based on randomly sampled paths has been shown to be significantly faster in practice. However, now for dynamic networks, there is no approximation algorithm for betweenness centrality that improves on static recomputation. The authors of the paper [Bergamini et al., 2014] address this deficit by proposing two incremental approximation algorithms (for weighted and unweighted connected graphs) which provides a provable guarantee on the absolute approximation error. The work [Bergamini, Meyerhenke, 2015] of the same authors studies an extension of the former algorithm for semi-dynamic Breadth-First Search (or BFS, from the previously mentioned paper) to batches of both edge insertions and deletions and proposed the first fully-dynamic approximation algorithms (for weighted and unweighted undirected graphs that need not be connected) with a provable guarantee on the maximum approximation error. Their experiments showed that the implemented algorithms can achieve substantial speedups compared to recomputation, up to several orders of magnitude.

For numerical experiments we used the tuned SWSF (T-SWSF) algortihm [Bauer, Wagner, 2009] and implemented it in Python programming language [Aminov et al., 2021]. For a fair comparison they also have implemented the original Dijkstra algorithm in Python by themselves, the implementation can be found in [Aminov et al., 2021]. The T-SWSF algorithm is as follows. Let $G = (V, E)$ be a directed graph with n nodes and m edges and a nonnegative length function $\text{len}\colon V \times V \to R^+ \cup \{\infty\}$. Let $s \in V$ be an arbitrary but fixed source. $d(v)$ denotes the length of the shortest s–v-path in G for any $v \in V$. Let the outdated distance vector $D[\,]$ be given. Then it can be said that we relax an edge (u, v) when we check if $D[v] > D[u] + \text{len}\,(u, v)$. We say we relax and update an edge (u, v) when we set $D[v] := \min\{D[v], D[u] + \text{len}(u, v)\}$. An edge (u, v) is said to be consistent if $D[v] = \text{len}\,(u, v) + D[u]$ and underconsistent if $D[v] > \text{len}(u, v) + D[u]$. The consistent value $\text{con}(v)$ of a node v is

$$\begin{cases} \min\limits_{(u,\,v)\in E}\{D[u] + \text{len}\,(u,\,v)\}, & v \neq s, \\ 0, & v = s. \end{cases} \tag{5}$$

A node is said to be consistent if $D[v] = \text{con}(v)$ and overconsistent if $D[v] > \text{con}(v)$.

The untuned version of an SWSF algorithm is the following: for each node v, a label $d[v]$ is given. Initially, $d[\,] = D[\,]$. We say we adjust an inconsistent node v when we set $d[v] := \text{con}(v)$ and insert v with priority $\min\,(D[v], d[v])$ in Q. In case v is already in Q, we only update its priority. We adjust a consistent node v when we remove it from Q. If v is not in Q, nothing needs to be done. Initially, we adjust each node which is the target of an edge in U. The structure of the main phase is described below. While Q is not empty, the algorithm performs as follows: extract and delete the minimum node w from Q; if $d[w] < D[w]$, set $D[w] := d[w]$ and adjust each outgoing neighbor of w. If $d[w] > D[w]$, set $D[w] := \infty$ and adjust w and each of its outgoing neighbors.

The tuned SWSF algorithm is similar, but it requires less computational effort. It relaxes fewer incoming edges: when we adjust an outgoing neighbor v of a node w with $d[w] < D[w]$, we compute $\text{con}(v)$ by $\min\{d[w] + \text{len}(v, w), d[v]\}$. The same strategy works in the initialization phase when we

compute con for a node n that is the target node of an edge with a decreased edge weight and the same strategy holds in the initialization phase for target nodes of edges with increased weight. The full pseudo-code of the algorithm can be seen in [Bauer, Wagner, 2009].

Experiments with the T-SWSF algorithm were done for the Stable Dynamic model only (not for simultaneous multi-stage transport model), the Anaheim city data were used [Transportation Networks for Research Core Team]. Experimental studies show that for arbitrary graph perturbations the T-SWSF performs worse than Dijkstra, however, if the ratio of the number of perturbed edges to the number of all edges is sufficiently small, the T-SWSF works faster than the recomputation of the shortest paths and distances from scratch using the Dijkstra algorithm. The ratio constant equal to $r^* = \frac{1}{20}$ was used, but probably it is not the optimal one (and depends on implementation). The graph perturbation situation with $r \leqslant r^*$ is referred to as sparse graph perturbation. In the Stable Dynamic model when using Universal Method of Similar Triangles (USTM) and Universal gradient method (UGM) solvers [Kubentayeva, Gasnikov, 2020], the sparse graph perturbation situations occur frequently (every third or even every second graph update is actually sparse), and therefore the utilization of T-SWSF in such cases seems profitable. In order to validate the profitability of T-SWSF when solving the stable dynamic model using USTM and UGM, the times needed to achieve the fixed duality gaps ε-s were compared (following the original research [Kubentayeva, Gasnikov, 2020]). Results are shown in Fig. 4.

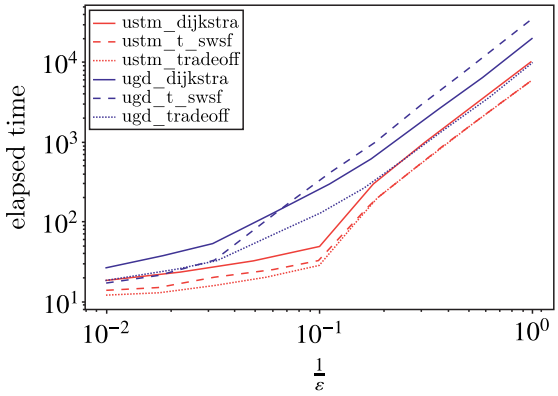

Figure 4. For validation of the profitability of T-SWSF when solving the stable dynamic model using USTM and UGM, the times needed to achieve the fixed duality gaps ε-s were compared. The -dijkstra suffix means that the SSSP problems are solved by recomputing all paths from scratch using Dijkstra algorithm. The -t-swsf suffix means that the T-SWSF algorithm is aplied for arbitrary graph perturbations (even for $r > r^*$). The -tradeoff means that Dijkstra is in use if $r > r^*$ and T-SWSF if $r \leqslant r^*$. Numerical experiments show that the T-SWSF can improve the time complexity in the cases considered

Except using the known algorithms, there is an original implementation [Zakharov, 2021] of the same assumption, namely, that a big part of the shortest paths remains the same as in the previous iterations. Also, it is supposed that the number of neighbors for any vertex is limited by a small constant, which is a usual case for transport graphs.

The proposed Algorithm 1 consists of several steps.

The first step complexity is $O(|V|)$, and the second $O(|V| + |E|)$. The complexity of the third step is $O(k \log k)$, where k is the number of vertices with changed shortest paths. And the final step complexity is $O(|V|)$. Thus, the total complexity is $O(|V| + |E| + k \log k)$, where k is the number of vertices with changed shortest paths. So, if the number of such vertices is small, we get a linear complexity instead of standard $O(|E| + |V| \log |V|)$. However, the general idea behind the algorithm looks reasonable,the real improvements were not obtained on the Anaheim data set (the algorithm was implemented in Python, and the Dijkstra algorithm was also implemented by the same author in Python).

Algorithm 1. Dijkstra++ algorithm

1: **Input:** Graph $G = (V, E, \text{len})$
2: **for** vertex v in V **do**
3: Recompute *shortestDistances*, based on the previous iteration tree order of the Dijkstra algorithm.
4: Find all vertices v satisfied: $shortestDistances[v] > shortestDistances[v_{pred}] + edgeDistance(v_{pred}, v)$, where v_{pred} is a predecessor of v.
5: Run Dijkstra algorithm starting with a set of vertices v, obtained in step 2. Any vertex could be added to the set of unvisited vertices only if the new distance to this vertex were less than *shortestDistances*[*vertex*], computed in step 1. Thus, only the vertices with the changed shortest paths would be evaluated during this step.
6: Tree order update. To make this in linear time, the iteration goes through the old tree order and only the vertices that are not visited during step 3 are taken into account. Then this tree order extended with tree order of visited vertices, obtained during step 3. In the tree order created in this way for every vertex its predecessor is located earlier than this vertex.
7: **end for**
8: **Output:** Graph $G = (V, E, \text{len})$.

Road pricing implementation

A common problem with design of transport networks is that the equilibrium achieved in them does not minimize the total travel time (social optimum) and, in particular, can lead to traffic jams. However, it was shown [Gasnikov, Gasnikova, 2020; Gasnikov et al., 2013] that adding a member to the cost function (see (6)) leads to social optimum in the network. Physically, this additional member is a monetary fee (in the time equivalent) for travelling along the corresponding edge. The convenience of such fees is that they are calculated from the characteristics of the edge only without the need to track the drivers' routes. We implemented such a road pricing system to demonstrate its work.

$$\overline{\tau}_e(f_e) = \tau_e(f_e) + \underbrace{f_e\tau'_e(f_e)}_{taxes}, \quad e \in E. \tag{6}$$

In this implementation we use the BPR-function as the $\tau_e(f_e)$: $\tau_e(f_e) = \bar{t}_e\left(1 + \gamma \cdot \left(\frac{f_e}{\bar{f}_e}\right)^{1/\mu}\right)$. Thus, $\overline{\tau}_e(f_e) = \tau_e(f_e) + f_e\tau'_e(f_e) = \bar{t}_e\left(1 + \gamma\frac{\mu+1}{\mu}\left(\frac{f_e}{\bar{f}_e}\right)^{1/\mu}\right)$. Further, if the social optimum f^{opt} is fixed, then penalties can be chosen to be constant $f_e^{opt}\tau'_e(f_e^{opt})$, $e \in E$. The only condition to keep it in effect is that the transport system must be maintained for given correspondences d_w and cost functions $\tau_e(f_e)$. Otherwise, the resulting equilibrium may no longer correspond to the social optimum.

After implementation of the tax system [Chekanov, 2021], the authors vizualized flows for better understanding of whether it helps or not, Figure 5.

Conclusions and open problems

In this paper, multiple improvements were proposed to the existing simultaneous multi-stage model code [Kotliarova et al., 2020; Kotliarova, Yarmoshik, 2020]; its purpose was to speed up the calculations of the model parts and improve the transport modeling detailing. The ideas of the projects were proposed by A. V. Gasnikov and E. V. Kotliarova. The first idea was to replace the Sinkhorn algorithm in the entropy model by the AAM-algorithm [Guminov et al., 2020], but, unfortunately, that didn't show any advantages as compared to the standard variant in the numerical

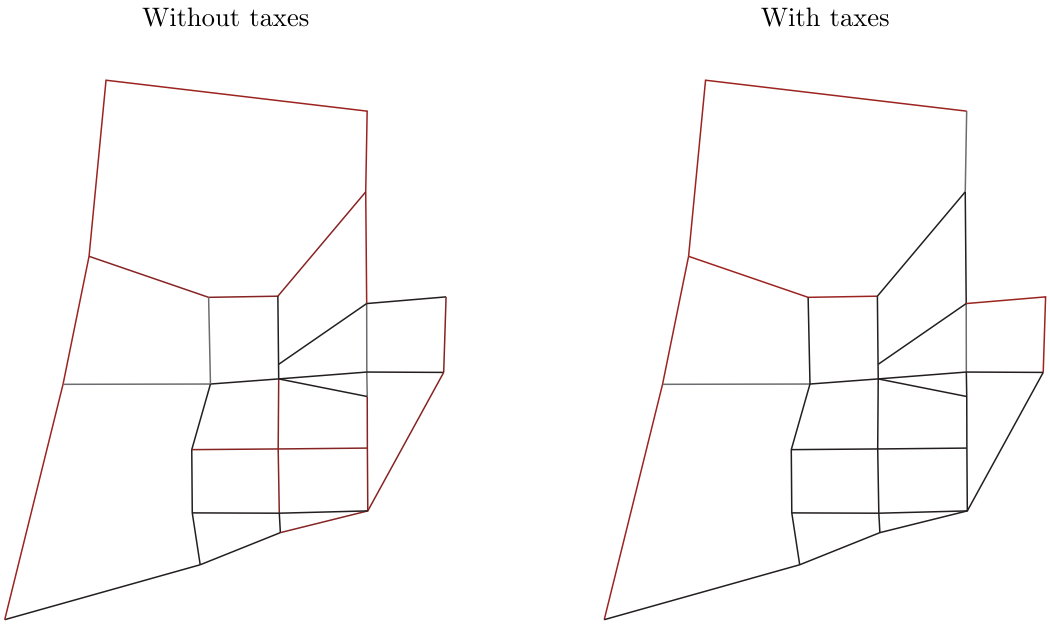

Figure 5. Transportation network of Sioux Falls city before adoption of the taxes (left) and after (right). Red color represents the ratio of actual flow f_e on an edge to its maximum capacity $\overline{f}_e$. It can be seen that, after taxes implementation, flows on particularly crowded roads have decreased

experiments [Severilov et al., 2021]. Secondly, the replacement of Dijkstra's algorithm for more modern and efficient algorithm T-SWSF was done in the Stable Dynamic model, and that results in relatively shorter computation times [Aminov et al., 2021]. Moreover, we attached a short review of the future research directions in the corresponding paragraph [Sunite, Deepak, 2018; Bergamini et al., 2014; Bergamini, Meyerhenke, 2015]. Thirdly, the toll highways were added to the model and their efficiency was estimated [Chekanov, 2021]. All studies were done as projects by master (sixth year) students attending the transport modeling course provided by the Phystech School of Applied Mathematics and Informatics, Moscow Institute of Physics and Technology (MIPT, Dolgoprudny, Russia). These projects were supervised by E. V. Kotliarova and A. V. Gasnikov.

References

Aminov T. V., Panchenko S. K., Mokrov P. V., Grishanov A. V. [Electronic resource]: https://github.com/PetrMokrov/TransportNet/ (accessed: 03.01.2022).

Bauer R., Wagner. D. Batch dynamic single-source shortest-path algorithms: an experimental study // 8th Int. Symp. on Experimental Algorithms (SEA '09.) — 2009. — Vol. 5526. — P. 51–62.

Bergamini E. et al. Approximating Betweenness Centrality in Large Evolving Networks // arXiv preprint. — 2014. — https://arxiv.org/pdf/1409.6241

Bergamini E., Meyerhenke H. Fully-dynamic Approximation of Betweenness Centrality // Algorithms – ESA 2015. Lecture Notes in Computer Science. — 2015. — Vol. 9294. — P. 155–166.

Braess D. Ueber ein Paradoxen der Verkehrsplanung // Unternehmensforschung. — 1968. — Vol. 12. — P. 258–268.

Checkanov M. [Electronic resource]: https://github.com/AlonsoQuijano/TransportNet-1/tree/tax_integration (accessed: 12.01.2022).

Cuturi M. Sinkhorn distances: Lightspeed computation of optimal transport // Advances in neural information processing systems. — 2013. — P. 2292–2300.

Gasnikov A. V., Gasnikova E. V., Mendel' M. A., Chepurchenko K. V. Evolyutsionnye vyvody entropiinoi modeli rascheta matritsy korrespondentsii [Evolutionary interpretations of entropy model for correspondence matrix calculation] // Mathematical Models and Computer Simulations. — 2016. — Vol. 28, No. 4. — P. 111–124 (in Russian).

Гасников А. В., Гасникова Е. В., Мендель, М. А., Чепурченко К. В. Эволюционные выводы энтропийной модели расчета матрицы корреспонденций // Математическое моделирование. — 2016. — Т. 28, № 4. — С. 111–124.

Gasnikov A. V., Gasnikova E. V. Modeli ravnovesnogo raspredeleniya transportnyh potokov v bol'shih setyah [Models of equilibrium flow distribution in large networks]. — Moscow: MIPT, 2020 (in Russian).

Гасников А. В., Гасникова Е. В. Модели равновесного распределения транспортных потоков в больших сетях. — М.: МФТИ, 2020.

Gasnikov A. V., Klenov S. L., Nurminskij E. A., Holodov Ya. A., Shamraj N. B. Vvedenie v matematicheskoe modelirovanie transportnykh potokov [Introduction to the mathematical modeling of traffic flows] / ed. A. V. Gasnikov. — Moscow: MCCME, 2013 (in Russian).

Гасников А. В., Кленов С. Л., Нурминский Е. А., Холодов Я. А., Шамрай Н. Б. Введение в математическое моделирование транспортных потоков / под ред. А. В. Гасникова, с приложениями М. Л. Бланка, К. В. Воронцова и Ю. В. Чеховича, Е. В. Гасниковой, А. А. Замятина и В. А. Малышева, А. В. Колесникова, Ю. Е. Нестерова и С. В. Шпирко, А. М. Райгородского, с предисловием руководителя департамента транспорта г. Москвы М. С. Ликсутова. — М.: МЦНМО, 2013. — 427 с. — 2-е изд.

Guminov S., Dvurechensky P., Tupitsa N., Gasnikov A. Accelerated alternating minimization, accelerated Sinkhorn's algorithm and accelerated iterative Bregman projections // arXiv preprint. — 2020. — https://arxiv.org/pdf/1906.03622

Ivanova A. C. et al. Kalibrovka parametrov modeli rascheta matricy korrespondencij dlya g. Moskvy [Calibration of model parameters for calculating correspondence matrix for Moscow] // COMPUTER. — 2020. — Vol. 12, No. 5. — P. 961–978 (in Russian).

Иванова А. С. и др. Калибровка параметров модели расчета матрицы корреспонденций для г. Москвы // COMPUTER. — 2020. — Т. 12, № 5. — С. 961–978.

Kotliarova E. V., Gasnikov A. V., Gasnikova E. V., Yarmoshik D. V. Poisk ravnovesij v dvuhstadijnyh modelyah raspredeleniya transportnyh potokov po seti [Finding equilibrium in two-stage traffic assignment model] // COMPUTER. — 2021. — Vol. 13, No. 2. — P. 365–379 (in Russian).

Котлярова Е. В., Гасников А. В., Гасникова Е. В., Ярмошик Д. В. Поиск равновесий в двухстадийных моделях распределения транспортных потоков по сети // COMPUTER. — 2021. — Т. 13, № 2. — С. 365–379.

Kotliarova E. V., Yarmoshik D. V. [Electronic resource]: https://github.com/tamamolis/TransportNet (accessed: 16.02.2021).

Kubentayeva M., Gasnikov A. Finding equilibria in the traffic assignment problem with primal-dual gradient methods for Stable Dynamics model and Beckmann model // arXiv preprint. — 2020. — https://arxiv.org/pdf/2008.02418

Nartsev D. Yu., Fesiuk M. A., Ibraev T. V. [Electronic resource]: https://github.com/Meshemi/TransportNet_fast_sinkhorn (accessed: 02.01.2022).

Severilov P. A., Ivchenko Ya. P., Sotnikov A. D., Revar V. I., Emelianov A. V. [Electronic resource]: https://github.com/severilov/transport_flows_proj (accessed: 02.01.2022).

Sheffi Y. Urban transportation networks. — Prentice-Hall, 1984.

Sunite G., Deepak G. Dynamizing Dijkstra: A solution to dynamic shortest path problem through retroactive priority queue // Journal of King Saud University – Computer and Information Sciences. — 2021. — Vol. 33, I. 3. — P. 364–373.

Transportation Networks for Research. — [Electronic resource]: https://github.com/bstabler/TransportationNetworks (accessed: 10.01.2022).

Wardrop J. G. Some Theoretical Aspects of Road Traffic Research // Proceedings of the Institution of Civil Engineers. — 1952. — Vol. 1, No. 3. — P. 325–362.

Wilson A. G. Entropy in urban and regional modeling. — Routledge, 2012.

Zakharov A. [Electronic resource]: https://github.com/A1ekse1/TransportNet/ (accessed: 11.01.2022).

Appendices

Sinkhorn algorithm

Algorithm 2. Sinkhorn Algorithm

Input: $x^0 = [[\lambda^l]^0, [\lambda^w]^0] = (0, \ldots, 0) \in \mathbb{R}^{2n}$ — starting point.
for $k \geqslant 0$ **do**
  if $k \bmod 2 = 0$ **then**
    Compute

$$[\lambda^l]^{k+1} = [\lambda^l]^k + \ln(l) - \ln\left(B([\lambda^l]^k, [\lambda^w]^k)\mathbf{1}\right),$$
$$[\lambda^w]^{k+1} = [\lambda^w]^k.$$

  else
    Compute

$$[\lambda^l]^{k+1} = [\lambda^l]^k,$$
$$[\lambda^w]^{k+1} = [\lambda^w]^k + \ln(w) - \ln\left(B^T([\lambda^l]^k, [\lambda^w]^k)\mathbf{1}\right).$$

  end if
end for
Output: $x^k = [[\lambda^l]^k, [\lambda^w]^k] \in \mathbb{P}^{2n}$.

Accelerated alternating minimization algorithm (AAM)

Algorithm 3. Accelerated alternating minimization algorithm (AAM)

Input: $x^0 := [[x^l]^0, [x^w]^0] = (0, \ldots, 0) \in \mathbb{R}^{2n}$ — starting point, $L_0 = 1$, $a_0 = 0$.
repeat
  Set $y^0 := [[y^l]^0, [y^w]^0] = x^0$.
  Set $v^0 := [[v^l]^0, [v^w]^0] = x^0$.
  $L_{k+1} = \frac{L_k}{2}$
  while True **do**
    Set $a_{k+1} = \frac{1}{2L_{k+1}} + \sqrt{\frac{1}{4L_{k+1}^2} + a_k^2 \frac{L_k}{L_{k+1}}}$
    Set $\tau_k = \frac{1}{a_{k+1}L_{k+1}}$
    Set $y^k = \tau_k v^k + (1 - \tau_k)x^k$
    Choose $i_k = \underset{i \in \{1,2\}}{\operatorname{argmax}} \|\nabla_i \varphi(y^k)\|_2^2$
    if $i_k = 1$ **then**
      Compute

$$[x^l]^{k+1} = [y^l]^k + \ln(l) - \ln\left(B([y^l]^k, [y^w]^k)\mathbf{1}\right),$$
$$[x^w]^{k+1} = [y^w]^k.$$

    else
      Compute

$$[x^l]^{k+1} = [y^l]^k,$$
$$[x^w]^{k+1} = [y^w]^k + \ln(w) - \ln\left(B^T([y^l]^k, [y^w]^k)\mathbf{1}\right).$$

end if
Set $v^{k+1} = v^k - a_{k+1}\nabla\varphi(y^k)$
if $\varphi(x^{k+1}) \leqslant \varphi(y^k) - \frac{\|\nabla\varphi(y^k)\|_2^2}{2L_{k+1}}$ **then**
Set $\widehat{d}^{k+1} = \frac{a_{k+1}d^k(y^k)+L_k a_k^2\widehat{d}^k}{L_{k+1}a_{k+1}^2}$
break
end if
Set $L_{k+1} = 2L_{k+1}$.
end while
until $|f(\widehat{d}^{k+1}) + \varphi(x^{k+1})| \leqslant \varepsilon_f$, $\|\widehat{d}^{k+1}\mathbf{1} - l\|_2 \leqslant \varepsilon_{eq}$, $\|(\widehat{d}^{k+1})^T\mathbf{1} - w\|_2 \leqslant \varepsilon_{eq}$
Output: $\widehat{d}^{k+1}$, x^{k+1}.

Flows reconstruction algorithm

Algorithm 4. Flows reconstruction

Input: times t
$f := 0_{|E|}$ {flows on edges}.
for origin o in O **do**
Get the shortest-path tree τ_0 from o to all destinations in D with weights t
traversal-order := TopologicalSort(τ_0) {sorting from furthest to closest vertices}
$f_{out} := 0_{|V|}$ {total output flow from each vertex}
$f_{out}[v] := d_w$ for $w = (o, v) \in OD$
for v in traversal-order **do**
Get predecessor p of v in τ_0
$e := (p, v)$
$f[e] := f[e] + f_{out}[v]$
$f_{out}[p] := f_{out}[p] + f_{out}[v]$
end for
end for
Output: flows f.