User Manual for the C-BiTA Project

September 8, 2025

This document is intended to help readers quickly understand and make effective use of our algorithms. It is organized into four sections:

- **Section 1** provides a step-by-step guide for viewing, editing, compiling, and running the algorithm source code.
- **Section 2** offers a step-by-step guide for interactively invoking the algorithm via the console to solve problems rapidly.
- **Section 3** outlines the procedure for reproducing the results reported in the paper.
- **Section 4** gives a concise overview of the key classes and methods implemented in the SBA and FW algorithms.

For the hardware and software requirements and environment:

- Operating System: Windows 10 (64-bit)
- Compiler/IDE: Microsoft Visual Studio 2022 (v143), ISO C++14 standard
- CPU: Experiments were conducted on an Intel(R) Xeon(R) Gold 5222 CPU @ 3.80GHz. A modern 64-bit processor with at least 4 cores is recommended.
- RAM: The workstation used was equipped with 64 GB RAM. For typical problem instances, at least 8 GB RAM is recommended.
- Additional Software: Python 3.x with matplotlib (for reproducing figures).

1 How to Run the Source Code

- Open the solution file located at '`..\C-BiTA\Source\Solution\mySolution.sln`' using Visual Studio 2022.
- Locate the `main` function in *trafficDriver.cpp*. The *trafficDriver.cpp* file is included in the source files of the `trafficDriver` solution within `mySolution`.
- Run the SBA (or FW) algorithm by invoking the `TestSBA()` (or `TestT2theta()`) function. An example of the call is: `TestSBA(''..\\..\\..\\data\\TestNetwork\\sf\\'',sf);`
- All networks tested in our experiments are provided in '`..\C-BiTA\data\TestNetwork`'. If users want to test their own networks, additional networks can be downloaded from *https:\\github.com\bstabler\TransportationNetworks* and placed in the `data` folder. To ensure the proper execution of the algorithm, each network must at least include the following three network files:
  - `name_trips.tntp`: A file for OD demand.
  - `name_tol.tntp`: A file for constant link tolls.
  - `name_net.tntp`: A file for network link parameters.
- Use `Ctrl + F5` to compile and execute the code.
- Output files will be generated in the same folder as the inputs. The first two types of files are produced by both algorithms, while the last three are specific to SBA:

- `name_alg.ite`: Iterative convergence results, including iteration number (*Iter*), objective function value (*OFV*), relative gap (*ConvIndc*), and elapsed time (*Time*).
- `name_alg.lfp`: Final link flow pattern, reported in the column *Flow*.
- `name_alg.pfp`: Final path flow pattern, showing the geometry of used paths and their associated flows.
- `name_alg.info`: Final number of efficient paths for each OD pair, along with the partitioning of the VOT distribution.
- `name_alg.tnum`: Number of shortest path trees per iteration (one line per iteration, one number per origin).

2 How to Run the .exe Console

In addition to the source code compilation method, we also offer a way to quickly utilize and test the algorithm via an interactive console. The following steps detail the process:

Open and run the execution file: '`..\C-BiTA\exe concole\trafficDriver.exe`'. Follow the instructions and input the network file path, network name, and the algorithm name. If you place the C-BiTA folder in the C drive, you'll be able to find the sf network files at '`C:\C-BiTA\data\TestNetwork\sf\`'. Accordingly, an appropriate interactive process using the exe console is as follows:

> Please input the path of your network files: (For example, D:\Data\network\sf\)
>
> C:\C-BiTA\data\TestNetwork\sf\
>
> Please input the network name: (For example sf, Anaheim, Barcelona, Winnipeg, chicagosketch, chi)
>
> sf
>
> Please input the algorithm name you want to test: (SBA or FW)
>
> SBA

The executables compiled with Visual Studio 2022 (v143 toolset) depend on the Microsoft Visual C++ Redistributable for Visual Studio 2022. On most Windows 10/11 systems this package is already installed. If, however, you encounter an error message indicating that a Microsoft Visual C++ runtime DLL (e.g., 'MSVCP140.dll', 'VCRUNTIME140.dll', or 'ucrtbase.dll') is missing, please download and install the latest supported Visual C++ Redistributable from the official Microsoft website: *https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist*

3 How to Reproduce the Numerical Experiments

All raw results that are used in the numerical experiments of the paper are provided in the folder '`..\C-BiTA\results\raw_results`'. The subfolders correspond to different sections of the paper as follows:

- `convergence_test`: results for Section 6.1,
- `num_of_tolled_links`: results for Section 6.2,
- `sens_toll_mag`, `sens_congestion`, `sens_dist`: results for Sections 6.3.1, 6.3.2, and 6.3.3, respectively.

In addition, Python scripts with the same names (located under '`..\C-BiTA\results`') can be used to process the raw results and generate the figures included in the paper. The figures will be automatically saved in the corresponding subfolder under '`..\C-BiTA\results\figures`'.

All input files for reproducing the experiments through the source code are stored in '`..\C-BiTA\data\TestNetwork`'. Each subfolder (named after the corresponding network) contains the input data used in Section 6.1. The `cs` subfolder also contains the input files for the experiments in Sections 6.2–6.3, organized in its subdirectories.

To reproduce these experiments through the source code:

- Open the solution file located at '`..\C-BiTA\Source\mySolution.sln`' using Visual Studio 2022.

- Locate the `main` function in *trafficDriver.cpp*.
- Uncomment the line following the `base_file_path` string variable that corresponds to your target experiment (note that scripts are organized according to the paper's sections), then compile and execute using `Ctrl + F5`.
- The output results will be generated in the same folder as the input files and can be opened, inspected, renamed, or relocated as required.

4 Structure and Classes

4.1 Implementation Structure

The network classes are defined as `TNM_SNET` class in the `trafficNet` solution (*'TNM_NET.h'*). The main network elements include:

- `TNM_SLINK`: It is a basic class used to represent network links. Derived classes like `TNM_BPRLK` supports evaluating link travel time using a BPR-type link performance function. Key attributes include `tail` node, `head` node, `volume`, `fft` (free flow time), `capacity`, `toll`, `cost` (travel time), etc.
- `TNM_SNODE`: This class represents network nodes which include attributes like `forwStar` (in-node links), `backStar` (out-node links), `pathElem` (shortest path element to the node), etc.
- `TNM_SORIGIN`: All origin-destination (O-D) pairs are organized using origins as roots, i.e., each origin stores a vector of destinations associated with positive demand.
- `TNM_SDEST`: The `dest` (destination node), `assDemand` (O-D pair demand), `origin` (the corresponding origin node), `pathSet` (paths used by each O-D pair), `pathSetbound` (path time equivalence of money (TEM) boundaries) are stored in `TNM_SDEST` object.
- `TNM_SPATH`: The `path` (the collection of links that make up the path) and some path cost evaluation functions are defined in the `TNM_SPATH` class.
- `SCANLIST`: `Scanlist` is introduced into TNM to accommodate different labeling shortest path problem algorithms.

The `buffer` attributes of each class are used to store data related to the object required for algorithm calculations.

The traffic assignment algorithms are derived based on the algorithm class `TNM_TAP` in file *'TNM_Algorithm.h'*. The general attributes and processes related to the execution of the traffic assignment algorithms like convergence accuracy, the maximum iteration and objective function value will be defined here. In the test functions in *'trafficDriver.cpp'*, we instantiate it and specify some algorithm and network attributes first. And then build the network using `Build()`. The `Solve()` process executes `Preprocess()`, `Initialization()`, `Mainloop()`, `Postprocess()` successively and check `Terminate()` when necessary. Then using `Report()` to output the iteration and equilibrium results.

As for the continuous bi-criteria traffic assignment problem, we define `VOT_DIST` class to capture the properties of continuous distributions like their probability density function (PDF) and cumulative distribution function (CDF). Except for the distribution named *'rational'* in sensitive analysis, the numerical experiments are executed using the distribution named *'uni'* in this research. The new algorithm classes `TAP_SBA` and `TAP_T2theta` in file *'TNM_Algorithm.h'* derives from class `TNM_TAP`. The attributes and functions of these two classes are shown in the next section. The idea is not to list every data members and functions. Rather, only those considered both important for programming and not straightforward for understanding will be described.

4.2 Class Description

The attributes and functions of `TAP_SBA` class are shown in Table 1.

Table 1: Attributes and functions of `TAP_SBA` class.

Attributes	Description

votdist:	The distribution object.
theta_min:	The minimum value of TEM, i.e., $\underline{\theta}$.
theta_max:	The maximum value of TEM, i.e., $\overline{\theta}$.
theta_low:	Tree lower boundary θ^{lb} in the parametric shortest path algorithm (Algorithm 4).
Dset:	$\mathcal{D}$ in Algorithm 4.
Hset:	$\mathcal{H}$ in Algorithm 4.
theta_up:	Tree upper boundary θ^{ub} in Algorithm 4.
path k buffer:	[0] Path toll π_k^w. [1] Column generation status, new added or not. [2] Path boundary θ_{k-1}^w. [3] Path boundary θ_k^w.
link a buffer:	[0] link theta boundary θ_a in Algorithm 4. [2] The descent direction of link volume in the FW algorithm. [3] The descent direction of link moment (the aggregation of TEM and link volume) in the FW algorithm. [6] Shortest path tree label, in tree or not. [7] The link volume solution of Algorithm 4 in the FW algorithm. [8] The link moment solution of Algorithm 4 in the FW algorithm.
node buffer:	[0] Time to node D^i, D^j in Algorithm 4. [1] Toll to node E^i, E^j in Algorithm 4. [2] Node is added to Dset or not in Algorithm 4. [4] The boundary that has retrieved paths on each destination (node) in Algorithm 4. [8] A counter used to calculate the number of shortest path trees generated from the origin (node).

Functions	Description
Initialize:	The initialization process of the SBA algorithm.
MainLoop:	The main loop process of the SBA algorithm.
SBA_GAP:	The calculation of the relative gap of the SBA algorithm (by path aggregation).
SBA_obj:	The calculation of the objective function value of the SBA algorithm (by path aggregation).
init_pathElemVector:	Initializing the "pathElemVector" to generate multiple trees when needed.
Generalized_cost_spp:	Using the shortest path label correcting algorithm to build the shortest path tree under given parameters.
MPA_path_scrach:	The parametric shortest path algorithm if we do not use the PTU tree replacement strategy.
Dial_MPA_path:	The process of the parametric shortest path algorithm (Algorithm 4) from the perspective of path and is mainly used in SBA.
Dial_LTA_path:	Parametric shortest path tree lazy assignment algorithm.
Dial_MIN_path:	The first time to do the parametric shortest path tree replacement.
Dial_PTU_path:	Parametric shortest path tree replacement.
Dial_MPA_link:	The process of the parametric shortest path algorithm (Algorithm 4) from the perspective of links, and is used for the all-or-nothing assignment in FW.
Dial_LTA_link:	Parametric shortest path lazy assignment from the perspective of links, and is used for the all-or-nothing assignment in FW.
T2_sortTopo:	Sorting nodes by topology order.
quickSort1:	A recursive function for sorting nodes.
revert_SPath:	Constructing path objects for the shortest path.
revert_SPath_lasttree:	Constructing the shortest path in the tree gotten from the previous step.
add_to_pathset_path:	When a new path is generated, this function is used to find whether the path is already included in the path set. If not, the new path is inserted in the appropriate sort position.
update_pathlinkflow:	Adding volumes to each link of the path.

`map_node_dest`:	Finding the destination object that corresponding to a node object when specifying the origin.
`Set_addpathbound`:	Setting TEM boundaries(path `buffer` [2], [3]) for paths belongs to an O-D pair from the information in `pathSetbound`.
`check_money`:	This function is used to check whether the paths of this OD pair are arranged in reverse order according to the toll value.
`boundadjusting_OD`:	It represents the process of Algorithm 2.
`adjust_one_bound`:	Using newton method to update a single boundary and making projection.
`SBA_innerGap_OD`:	The calculation of Δ_v in Eq. (17).
`deletenousepath_OD`:	Finding whether there are paths with no flow in the OD pair, and delete those paths.
`adaptive_inner`:	The adaptive inner loop procedure used in this paper.
`adaptive_inner2`:	Another adaptive inner loop procedure that can be used to test convergence.

The attributes and functions of `TAP_T2theta` class are shown in Table 2. Note that class `TAP_SBA` is its parent class.

Table 2: Attributes and functions of `TAP_T2theta` class.

Attributes	Description
`lineSearchAccuracy`	The line search termination interval length for step size.
Functions	**Description**
`Initialize`:	The initialization process of the FW algorithm.
`MainLoop`:	The main loop process of the FW algorithm.
`T2theta_GAP`:	The calculation of the relative gap of the FW algorithm (by link aggregation).
`T2theta_Objective`:	The calculation of the objective function value of the FW algorithm (by link aggregation).
`T2theta_lineSearch`:	Searching for an optimal step size in the descending direction in FW algorithm.
`T2theta_updateSolution`:	Calculating a new solution of link volumes and moments and updating the link travel time.