How could innovative decentralized organisational approaches contribute to addressing humanity's societal, sustainability, and governance challenges? A software demonstrator deliverable.

Marcin Korecki
Omkar Zade

September 26, 2023

1 Introduction

Starting from the paradigm of smart cities, we proposed that the requirement of organizing societies in a more resilient way implies the need for more decentralized solutions, based on digitally-assisted self-organization. This concept is also compatible with sustainability requirements and stronger democratic participation.

Traffic signal control as a methodology of alleviating traffic and improving city life has been a core point of interest for researchers and industry working on smart cities. Proposing new solutions in the domain is challenging as one cannot easily test them in real life. Therefore, most work on new designs is done in simulated environments. Due to that many of the work proposed is difficult to compare with existing methods due to different simulators being used.

As part of work package 2 of the CoCi project we have proposed several improvements and comparisons with existing methodologies for traffic signal control [Kor22, KH22, KDH23]. We have worked with both self-organizing methods as well as machine learning based ones. To make sure that our work reaches a wide audience and offers other researchers a chance to compare with and further improve with designed a demonstrator that can visualize the functioning of some of the methods we have worked with. The visualisations make it possible to validate the functioning of each of the methods beyond the aggregate metrics that are usually used. By observing the qualities of the actual traffic induced by the methods one can reason about the appropriateness of each of the methods.

In the following sections we provide details on the methods and scenarios available in our demonstrator as well as a detailed documentation and instructions for setting up and contributing to the project.

The demonstrator is publicly available at the following address: `https://demonstrator.inn.ac/`.

2 Methods & Scenarios

Our tool demonstrates the performance of some Machine Learning based as well as Self-Organizing approaches to traffic signal control in a variety of scenarios. The methods and scenarios are introduced in this section. For more detailed description we urge the reader to refer to the publications we indicate.

2.1 Methods

The following methods are available in our demonstrator.

- **GuidedLight** A Deep Q-Learning method using length-agnostic state design and heuristic exploration [KH22].
- **PressLight** A Deep Q-Learning method based on intersection pressure minimisation [WCZ+19].
- **Analytic+** A self-organizing analytic method relying on insights from traffic physics [LH08].
- **Demand** A simple adaptive method that always gives green to the flows with the largest number of vehicles.
- **Fixed** A simple baseline cyclically changing lights in a loop.

2.2 Scenarios

The following scenarios are available in our demonstrator (visualised in Figure 1).

- **2x2** A synthetic homogenous network with high traffic demand [ZDZ+19].
- **4x4** A synthetic heterogenous network (varied approaches lengths) and medium traffic demand [Kor22].
- **Hangzhou** Realistic traffic network of the city Hangzhou, China with flow based on real-world data [ZDZ+19].
- **NY48** Realistic traffic network of part of Manhattan, New York City, USA with flow based on real-world data [KDH23, ZDZ+19].

3 Documentation

As a part of the CoCi project, COSS ETHZ has built a software demonstrator to visualize various traffic control algorithms. This repository contains the server and frontend code for the project. This file explains the high level design and setup instructions.

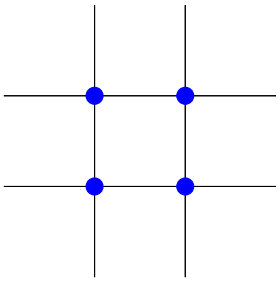
Table of contents:

- Design
- Setting up for development
- Deploying to production
- Example usage
- Live instance

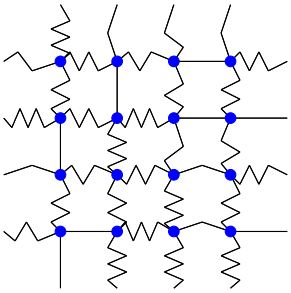
3.1 Design

The app consists of a Python/Flask backend (a simple REST API, located in ‘app.py‘) and a Svelte frontend (locate under ‘svelte-app/‘). Moreover, the simulation files generated using CityFlow [ZDZ+19] are stored on the server under ‘$SERVER_HOME/static/software_demonstrator_coci/‘.

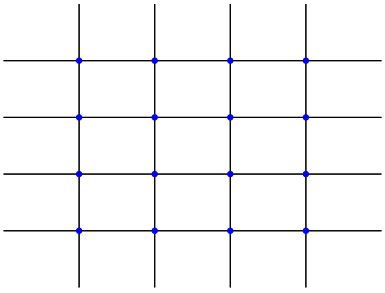
Below we explain the responsibilities of each of these components. A short technical presentation can be found here.



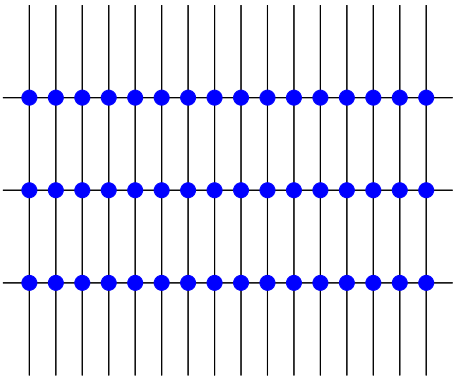
(a) 2x2 scenario road network



(b) 4x4 scenario road network



(c) Hangzhou scenario road network



(d) NY48 scenario road network

Figure 1: Road networks of the scenarios used in the demonstrator. Black lines represent links, blue dots represent intersections.

3.1.1 Simulation files

Each simulation is a parametrized by set of 3 files:

1. A roadnet (.json) definition of the network (aka Scenario)
2. A flow (.txt) definition of the traffic flows with 3600 timesteps (aka Method)
3. A density file which gives the density of vehicles on each road at each timestep

3.1.2 Flask server

The flask server is responsible for serving the simulation files to the frontend. It also provides a REST API for the frontend to

1. Fetch the list of available simulations. For each scenario, currently 5 possible methods are available.
2. Given the scenario/method combination, get the 3 files mentioned above corresponding to the combination.

3.1.3 Svelte frontend

The frontend is responsible for rendering the simulation files. It extends the CityFlow frontend with additional featrues such as:

1. Hosted predefined scenarios and a dropdown to select them
2. Support for multiple simulation runs to let user compare different methods side by side
3. A heatmap according to traffic density

We have used Svelte to modularize the design of frontend – for example:

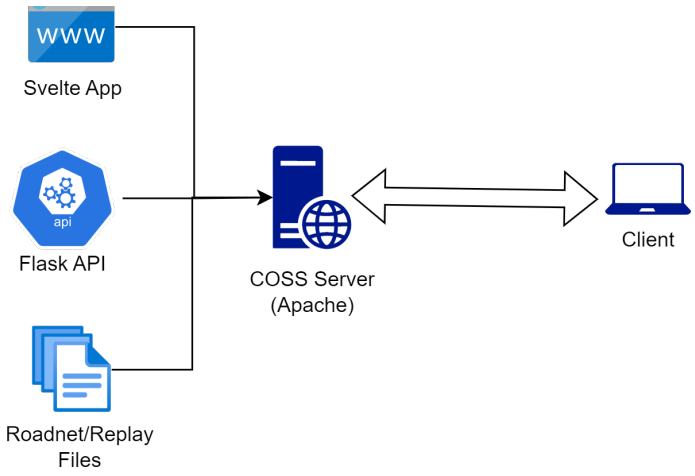


Figure 2: High Level Design

- to encapsulate the simulation in a component, which allows us to spawn multiple of them for side-by-side comparison
- to refactor Navbar, Controller etc into their own components so they can be reused or placed anywhere on the screen etc.

We have also replaced gradually JavaScript with TypeScript for type-safety and better developer experience.

3.2 Setting up for development

3.2.1 Prelimnaries: preparing the simulation files

You can generate the roadnet/flow files yourself using the CityFlow simulator. Alternatively, you can download the files for the scenarios we host from here. The server expects all simulation files to be in present in the 'static/software_demonstrator_coci/' directory, so create this directory and dump the files there. We have not checked-in any simulation files to github due to their large size.

3.2.2 Running the development Flask server

1. Create a virtual environtment and activate it

```
python -m venv env
source env/bin/activate
```

2. Install the app

```
pip install -r requirements.txt
```

3. Run the app

```
env=development flask run

 * Debug mode: off
WARNING: This is a development server.
Do not use it in a production deployment.
Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

3.2.3 Starting the Svelte app

You need ‘node‘ to run the development server. I recommend using Node Version Manager ‘nvm‘ to install/maintain node. The following commands have been tested with ‘node v18.12.1‘. (To see the currently active version use ‘nvm current‘)

Change to ‘svelte-app/‘ directory. Install dependencies with

```
npm install
```

and run bash

```
npm run dev
```

This setup assumes that the flask server is running on port ‘5000‘ on local as above. If this isn't the case, modify the variable `${SERVER_URL}` in utils.ts to appropriate value.

3.3 Deploying to production

Flask is not secure or performant enough to be used as a production server. Hence, we proxy the flask app by a gunicorn server.

[INTERNAL: these instructions are specific to our setup at COSS, and will need to be adapted depending on your infrastructure] The COSS server runs an Apache server to host all it's websites. Hence, we create an apache configuration for our app (example configuration given in demonstrator.inn.ac-le-ssl.conf, and proxy API requests to the gunicorn instance. The ‘ProxyPass /api http://127.0.0.1:8134/‘ directive tells apache to forward all requests to ‘/api‘ to the gunicorn server running on (non-exposed) port ‘8134‘ on the server.

3.4 Example usage

The four sub-figures in Figure 3 demonstrate the starting configuration, a basic simulation run, a simulation run with the traffic density overlay, and a side by side comparison of two different methods on a 2x2 network.

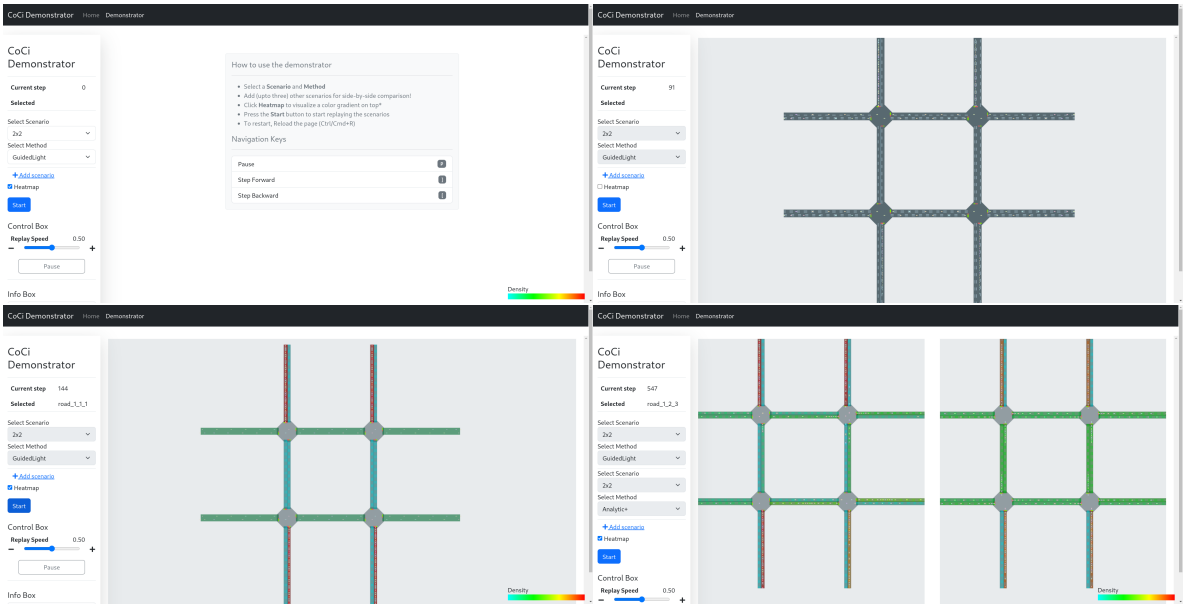


Figure 3: Example usage of the tool

3.5 Live instance

The app is currently live at https://demonstrator.inn.ac/

4 Extending and contributing

First, go over Documentation in the previous section to understand the high level design. Set up the app locally by following the setup instructions in 3.2.

To familiarize yourself with Flask and Svelte I recommend the official tutorials/quickstart guides: flask quickstart, svelte tutorial.

The app is distributed under Apache License. To extend the app for personal use, create a fork of the project. If the change you are making is more generally applicable, create a pull request on this repository. Below we give an overview of the codebase to facilitate extension and contribution.
Table of contents:

4.1 Flask app

The entry point of the flask app is app.py. The only dependency is data.py which contains mapping from scenario/method in the UI dropdown to the corresponding simulation files. `app.py` provides a REST API for the frontend to fetch the list of available scenarios and methods, and to fetch the simulation files for a given scenario/method combination.

4.2 Svelte app

In svelte-app/, at the root level are files generated while creating a new app with SvelteKit and contain various configuration files related to the app - e.g. `package.json` contains the dependencies.

The actual frontend code is located under svelte-app/src/. In svelte, each unique path in the app is called a 'route'. The `+page.svelte` is the entry point for each route. Hence, the landing page (`/`) is defined in ../routes/+page.svelte.

Similarly, the core of the app, i.e. the demonstrator itself (`/demonstrator`) has it's entry point in ../routes/demonstrator/+page.svelte

You will notice that `<Navbar/>` and `<Footer/>` are included as components on the landing page, and `<Navbar/>` is included also in the demonstrator page. These components are defined at the root level in ../routes/navbar.svelte and ../routes/footer.svelte.

The controller logic (i.e. dropdowns, speed, start, pause) is encapsulated in the `<Controller/>` component defined in ../demonstrator/controller.svelte.

In Svelte, a 'store' is used to share state across various components. Components can react when the state of the store changes. The `store` is defined in ../demonstrator/stores.ts.

The simulator itself is defined in the Simulator component. This is mostly a major refactoring and an extension of the CityFlow's script.js based on `PIXI.js`.

On user interaction, the `<Controller/>` updates the state in `store`, which in turn triggers the simulator to update the simulation. The simulator reacts to this update and makes the necessary changes to the simulation.

When simulations run side-by-side, it is important to synchronize them at every time step. The synchronization logic is implemented in `createGlobalCount()` in 'store.ts'

4.3 Adding a new scenario / method

Add the scenario and the supported methods to the dictionaries `roadnet_options` and `replay_options` in data.py and add the corresponding simulation files to `static/software_demonstrator_coci/`. The frontend will automatically pick up the new scenario and display it in the dropdown. No changes to frontend code are necessary. The flask server will need to be restarted after this.

5 Concluding Remarks

In this document we have recorded the software demonstrator for the CoCi project. We detailed its functionalities, capabilities as well as implementation and documentation. The tool allows anyone to compare both novel machine learning and traditional self-organizing methods for traffic signal control. Moreover, as it is an open source project, our documentation allows other researchers and users to build up on the provided codebase to extend the capabilities of the software. The project highlights how self-organizing approaches can be deployed in real life on the example of traffic signal control problem. It enables both citizens and scientists to compare different paradigms of control and educate themselves about the advantages and disadvantages of each methodology.

References

[KDH23] Marcin Korecki, Damian Dailisan, and Dirk Helbing. How well do reinforcement learning approaches cope with disruptions? the case of traffic signal control. *IEEE Access*, 11:36504–36515, 2023.

[KH22] Marcin Korecki and Dirk Helbing. Analytically guided reinforcement learning for green it and fluent traffic. *IEEE Access*, 10:96348–96358, 2022.

[Kor22] Marcin Korecki. Adaptability and sustainability of machine learning approaches to traffic signal control. *Scientific Reports*, 12(1):1–12, 2022.

[LH08] Stefan Lämmer and Dirk Helbing. Self-control of traffic lights and vehicle flows in urban road networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(4):1–36, 2008.

[WCZ+19] Hua Wei, Chacha Chen, Guanjie Zheng, Kan Wu, Vikash Gayah, Kai Xu, and Zhenhui Li. Presslight: Learning Max pressure control to coordinate traffic signals in arterial network. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1:1290–1298, 2019.

[ZDZ+19] Huichu Zhang, Yaoyao Ding, Weinan Zhang, Siyuan Feng, Yichen Zhu, Yong Yu, Zhenhui Li, Chang Liu, Zihan Zhou, and Haiming Jin. CityFlow: A multi-agent reinforcement learning environment for large scale city traffic scenario. *The Web Conference 2019 - Proceedings of the World Wide Web Conference, WWW 2019*, pages 3620–3624, 2019.