A web-based data visualization platform for MATSim

William Charlton, Technische Universität Berlin, Germany

Abstract

There are many tools available for analyzing MATSim results, both open-source and commercial. This research builds a new visualization platform for MATSim outputs that is entirely web-based. After initial experiments with many different web technologies, a client/server platform design emerges which leverages advanced user interface capabilities of modern browsers on the front-end, and relies on back-end server processing for more CPU-intensive tasks. The initial platform is now operational and includes several aggregate-level visualizations including origin/destination flows, transit supply, and emissions levels; as well as one fully disaggregate traffic visualization. These visualizations are general enough to be useful for various projects. Further work is needed to make the visualizations more compelling and the platform more useful for practitioners.

---

1. Introduction

MATSim is an open-source framework for implementing large-scale agent-based transport simulations (Horni, et. al, 2016). MATSim is widely used for transportation research in academic settings, and is gaining momentum as a tool ready for practice in real-world planning contexts.

There are many tools available for analyzing MATSim results, both open-source and commercial. Typically, analysts can choose either the free tool OTFVis (Strippgen, 2016) or the commercial software Via (Rieser, 2016), both of which are desktop software packages requiring installation as well as a fair amount of technical acumen to operate. Alternatives to these tools include the more general-purpose desktop mapping software packages such as QGis or ArcGIS, again both of which require installation and expertise to use, or statistical software packages.

As MATSim moves from the confines of academic research to a more public-facing role, a notable gap is apparent: there are no web-based, interactive tools available for disseminating MATSim results. This creates a challenge for using MATSim in public policy settings: the only people who can meaningfully examine and explore results are those who have extensive technical knowledge and access to the specialized software and large datasets involved.

This research explores one way to fill this gap: building an open web-based visualization platform which is specifically designed to complement MATSim.

2. Motivation

The rapid advancement of Internet browsing technologies in the last five years has enabled the web browser to do things much more "application"-like than ever before: background processing, three-dimensional rendering using GPU acceleration, offline support, and more. The combination of these technologies and their standard implementations on every popular hardware type and operating system now makes the web a very compelling platform.

For MATSim research, the question is: could a web browser really be useful for exploring and delivering results when the datasets are so large? Answering this question is the primary motivation for this research. Essentially, has the web become powerful enough for MATSim?

Currently, analysis of MATSim outputs ends up in research reports, PDF's, screen-recordings, and presentations. An online dashboard of results, which a user could explore and manipulate, would not only be more interactive but might also reveal findings that the original analysts hadn't anticipated.

3. Requirements

The research team at TU Berlin had several "blank slate" discussions before any code was written: meaning, if we could start at the very beginning and design something completely web-based and open, what would the bare minimum requirements be for it to be truly useful? The following requirements emerged from those discussions.

Requirement 1: Web browser-based

Given the above-stated motivation and hypothesis that the modern web platform is ready for large-scale visualization tasks, the most obvious requirement is that the product of this research must work with any modern web browser.

Several specific web technologies developed and made widely available in recent years enable us to perform this research: HTML 5, CSS 3, WebGL, ECMA Script 6, and Web Workers. Briefly, these technologies are:

- **HTML 5** improves and standardizes the "document model" of what constitutes a web page and how it is specified.
- **CSS 3** is a styling language that enables fine-grained styling of individual elements on a page. CSS 3 defines in a consistent, standard way the details of things such as color, size, layout, and animation of page elements.
- **WebGL** provides browser support for the 3D-accelerated graphics capabilities of modern machines.
- **ECMAScript 6** is an updated version of the Javascript scripting language that has been part of the web platform since the early 1990's. Recent versions of Javascript eliminate the more problematic aspects of the language and make it easier for developers to create bug-free, efficient code.
- **Web Workers** are a recent addition to the web platform that allow background thread processing for long-running tasks. Before Web Workers, there was no way to run truly multi-threaded code inside a browser.

A complication in web development is that the major web browser vendors implement these technologies on their own timelines, some much more rapidly than others. Further complicating things is the reality that end users do not always upgrade their browsers frequently (or at all). This creates a landscape where there is a technology adoption curve with a very long tail. Developers of every web site need to make a conscious decision about where to draw the line choosing necessary technologies for their site to operate correctly, knowing that some users with older browsers will either have a sub-optimal experience or no access to the site at all.

For this research, we are deliberately exploring the latest *standard* web technologies, with the expectation that access to these technologies will become more and more common in the future. Thus, we are targeting the most recent versions of modern web browsers as of 2019, including Google Chrome, Mozilla Firefox, and Apple Safari. All three browsers fully support the above-listed technologies, and importantly, all three auto-update automatically, ensuring that most users of those browsers stay current as these technologies evolve.

Requirement 2: Open source

The entire project, including all front-end (browser) and back-end (server) must be fully open-source.

No proprietary or closed licensing schemes were considered, because excellent proprietary visualization packages for MATSim already exist. Creating a competing product would be duplicative and unnecessary, and would not further the research goal of determining whether web-based technology is now advanced enough to work with MATSim outputs. The goal of this research is not to replace existing, proprietary solutions, but rather to complement them.

The software developed as part of this research is licensed entirely with the GNU General Public License v3, commonly known as the "GPL" (Free Software Foundation, 2007).

This matches the license of MATSim itself. Several other open-source licenses were considered, including the MIT License and the Apache Public License, but the benefit of sharing a common license with MATSim outweighs any perceived benefits of switching to other open licenses.

Requirement 3: Use good defaults, with minimal configuration, and be opinionated

Since its inception, the web platform has had a relentless focus on simplicity and smooth user onboarding. Users are accustomed to being immediately familiar with a site – often within seconds of their first interaction. Because of this expectation, it is critical that this research follow current best practices for user interface (UI) and user experience (UX). Specifically, that means using familiar UI paradigms such as navigation bars and breadcrumbs, separating configuration from usage, limiting settings and options to the bare minimum, and being "opinionated", i.e. encouraging a correct way to accomplish a task.

This approach is dissimilar to some data exploration tools (e.g., QGis and Via) where extreme configurability is emphasized. Rather than providing endless options for things such as scales and color ramps, our research focuses on choosing good defaults and determining whether that is sufficient for the software to be useful.

Requirement 4: An extensible platform

Every data visualization use case is different; there is no way to anticipate how the tool will be used. If the platform is too generic, it will be not at all useful. Conversely if only hard-coded visualizations are created for specific projects, it will be relegated to "demo-ware", meaning it is a successful technology demonstration but not actually useful for real users.

To fulfill this requirement the software platform will need to be extensible: basic capabilities and templates will be provided, but a user with some level of coding skill should be able to create new visualizations that are not anticipated by the researchers.

Initial experiments

It is no exaggeration to state that the Javascript code library ecosystem is extremely, enormously large. Thousands of libraries and packages are available on a common centralized Javascript repository known as "NPM", and there are often multiple packages that do similar things. As a developer, one must assess and select from these packages or choose to solve a problem by writing code by hand. Of course these libraries are of varying levels of popularity and quality.

Based on the requirements laid out above, some initial experiments were carried out to assess various approaches before committing to a technology stack.

Visualizing time-dependent data on a geographic map

Two popular web-based Javascript libraries were tested for displaying geographic data; Leaflet and Mapbox GL. A simple test case comprised of MATSim simulation outputs was developed, with the goal of displaying aggregated roadway link volumes by time of day.

Leaflet is very popular and its application programming interface (API) is a bit simpler than that of Mapbox GL. Leaflet uses background map "tiles" at specific zoom levels, and layers data on top of those base maps. With small networks (we tested Cottbus, Germany, a small city of 100,000 inhabitants) Leaflet performed well, but medium-sized and large-sized networks with many elements visible at once suffered from noticeable performance degradation. This was problematic, as this was the simplest use-case envisioned.

Mapbox GL fared much better, apparently better-suited to displaying large datasets with many visible features simultaneously. In addition, Mapbox GL's use of 3D vector graphic mapping instead of preset tiles made for a much more pleasing user experience, with smooth animations between zoom levels and better background processing during page loads. For these reasons, Mapbox GL was chosen as the base map for the remaining geographic visualizations.

Visualizing non-geographic data

There are literally dozens of data visualization libraries available for the web which provide ways to produce charts and plots of varying complexity. Our requirement of using open-source code narrows the field considerably.

After experimenting with several packages including D3, Raphael, Morris and others, the package Vega-Lite (Satyanarayan, 2016) exhibited many of the characteristics desired. Notably, Vega-lite follows a "grammar of graphics" declarative syntax, as popularized by Wilkinson (2005), and this grammar allows concise description of the meaningful components of a graphic.

Dealing with large datasets

MATSim network files are usually small enough to fit in RAM, but MATSim plan files and event files can be much larger than RAM, necessitating careful consideration about how to handle them.

Modern browsers allow access via API to a data storage area that is unique per hostname, i.e. http://mysite.com is allowed some storage on the local machine. Each browser vendor implements this differently, with strict limits on the absolute amount of data as well as on the percentage of free space on the user's machine. It became apparent that this local browser-based storage would not be sufficient for MATSim outputs. A client-server paradigm emerges as a viable alternative, and indeed this is how most websites operate: the browser is just the front-end to the heavier processing and storage tasks that happen on someone else's server.

A companion paper from J. Laudan (citation pending) is being submitted simultaneously, that describes the server component developed for handling large MATSim outputs. The front-end developed for this research uses the same back-end server as described in that paper.

Platform architecture

The client/server architecture depends on a set of back-end services for user authentication and file storage. Those back-end services are not described in this paper; suffice it to mention that the front-end communicates with them to establish what resources a user has access to, and provides an API with which to query and fetch those files and resources.

The front-end architecture has several interacting components:

"Vue" Single Page Application

The primary framework used to build the application is known as "Vue JS" (Vue). Vue is a framework for building interactive user interfaces on the web, and it depends on HTML5, CSS 3, and Javascript. Vue provides many services which allow a web page to behave more like a full-featured application, including state management, routing between different URLs, and componentization of code in a way that encourages reuse and loose coupling.

Vue components each encapsulate three elements required for the modern web: the HTML layout, the javascript code, and the CSS formatting. Components only interact with each other through well-defined pathways of properties and events, improving debuggability.

Build system

The build system of a modern web application is fairly complex and the Javascript ecosystem changes rapidly. After numerous iterations, the current build system comprises a series of

individual tools that all work together to produce the final assets that get sent to a user's browser. Those tools include the Vue command line interface (CLI), the NPM package manager, webpack, babel, and TypeScript.

Visualization plug-ins

One of the main requirements of this research is to produce a system where new visualizations can be produced rapidly and dropped into the existing framework to generate new capabilities.

The Vue component architecture enables this. To create a new visualization, a developer copies an existing "blank" visualization template and gives it a new name, specifies the file inputs and parameters required, and then uses the above-described libraries to modify the code per their needs.

This currently requires ample coding skill in Javascript; it is not a system that is point-and-click like an online data exploration tool.

Results: the current state of the tool

A working instance of the platform is now online and available at https://viz.vsp.tu-berlin.de. Sample datasets are uploaded, and pre-built visualizations are publicly accessible, as a demonstration of the platform's current state. There is also a user login system so that researchers can extend and experiment with the system without exposing data or partially completed work to the public.

Basic user, project, and file management capabilities are operational. This includes grouping files by model run or by other user-defined tags.

Several types of aggregate visualizations are currently operational:

- Origin/destination flows between aggregate areas
- Link flows by time of day
- Transit supply explorer, which displays all transit routes and allows the user to see which routes serve specific stops and links.
- Sankey diagrams, which can be used to depict changes/flows between between scenarios across multiple choices, such as shifts in mode between two scenarios
- Emissions levels on a geographic grid basis

In addition, one disaggregate visualization is available:

- A vehicle flow simulation, showing individual vehicle agents in real-time on the network. The details of this visualization are described in the companion paper by J. Laudan (citation pending).

See the following screenshots in *Figure 1* for examples of the current state of the user interface.

Performance

Even with modern hardware and the latest browsers, it is quite challenging to produce performant, visually pleasing results with disaggregate MATSim data. The vehicle flow simulation depends

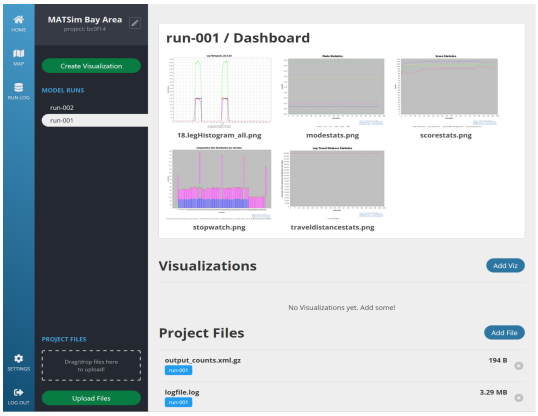

Figure 1a: Project dashboard

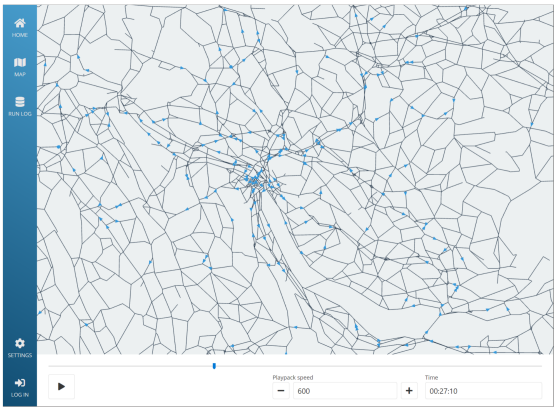

Figure 1b: Vehicle animation

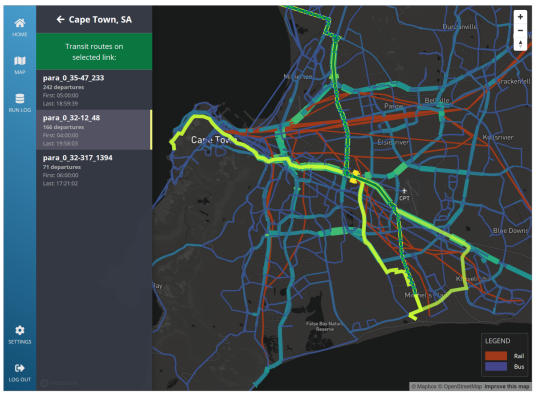

Figure 1c: Transit supply explorer

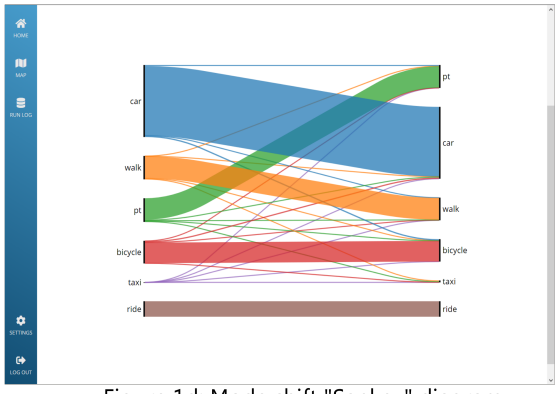

Figure 1d: Mode shift "Sankey" diagram

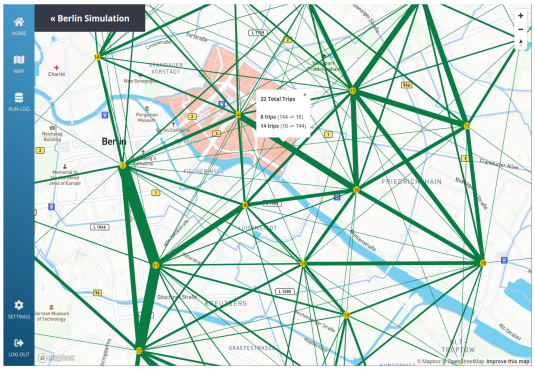

Figure 1e: Origin/destination flow aggregation

Figure 1f: Link volume summary

Figure 1: Sample visualizations, 1a - 1f

heavily on the back-end server to produce and deliver simulation "frames" to the browser in real-time, so that the browser simply has to render the data.

Various levels of aggregation make MATsim data much easier to visualize, as is reflected in the sheer number of visualizations this research was able to produce with aggregate data.

Based on this, further work needs to be done to leverage the back-end server capabilities we now have available.

Conclusions and outlook

Experimenting with the various technologies and getting all of the disparate pieces working together was an enormous task, one which took much longer than anticipated. However, those decisions are now behind us and the platform has become quite stable.

A new visualization can now be generated by the researchers in a matter of days. The researchers are admittedly very familiar with the inner workings of the system, but even so it has been encouraging to see new visualizations go from ideation to rough draft in such a short time.

None of the above-listed visualizations are particularly groundbreaking or visually stunning. All of them could be easily created in other tools. This is a bit disappointing but the open nature of the platform, requiring no software installation by end-users, still has an advantage: it opens up the display of MATSim results to the public and to decisionmakers, even if they do not have access to desktop mapping or travel forecasting software.

Also of note is that the world has not stood still while this platform was under development. Just in the past year, major data visualization efforts from well-funded companies such as Uber and others have been released. There are legitimate questions about how much of this work could be superceded by large, well-funded, professional coding teams.

Despite these concerns, the MATSim visualization framework is operational and is now just beginning to be useful for researchers at the department of its creation. This bodes well for further development in the near future.

All code is available on the MATSim Github site, at https://github.com/matsim-org/viz.

9. References

GNU General Public License (June 29, 2007). Version 3. Free Software Foundation. URL: https://www.gnu.org/licenses/gpl.html

Horni, A., Nagel, K. and Axhausen, K.W. (eds.) 2016 The Multi-Agent Transport Simulation MATSim. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw. License: CC-BY 4.0

Rieser, Marcel (2016): Senozon Via. In Andreas Horni, Kai Nagel, Kay W. Axhausen (Eds.): The Multi-Agent Transport Simulation MATSim: Ubiquity Press, pp. 219–224.

Satyanarayan, A., Moritz, D., Wongsuphasawat, K., and Heer, J. (2016): Vega-Lite: A Grammar of Interactive Graphics. IEEE Transactions on Visualization and Computer Graphics, Volume 23, Issue 1. DOI: https://doi.org/10.1109/TVCG.2016.2599030

Strippgen, David (2016): OTFVis: MATSim's Open-Source Visualizer. In Andreas Horni, Kai Nagel, Kay W. Axhausen (Eds.): The Multi-Agent Transport Simulation MATSim: Ubiquity Press, pp. 225–234.

Wilkinson, Leland (2005): The Grammar of Graphics, Second Edition. Springer Press, Chicago, USA. DOI: https://doi.org/10.1007/0-387-28695-0