Chicago Loop Traffic Simulation

December 8th, 2024

Jack Cogswell, Justas Jakubonis, and Evan Jones

MSDS 460: Decision Analytics - Fall 2024

Northwestern University

Abstract

This paper models a simple traffic simulation on a grid of roads and intersections using a discrete event system specification (DEVS) in both Python and Go in order to assess the language suitability for a more elaborate, thorough, and geographically expansive simulation. A benchmark of the two languages concludes that while Go exceeds Python in execution time, it has a higher peak memory usage. Go's advantage in execution time appears to be more strongly exhibited as the scale of the simulation increases.

Introduction

With inefficiencies in contemporary traffic systems, the sheer volume of commuters and unforeseen events lead to massive congestion that can last hours throughout the day. A micro-level solution may be to reroute through a lower congested path, but making these decisions independently leads to expanded congestion. In bad enough cases, congestion will ripple throughout the whole system leading to gridlock in which everything comes to a stand still. In worst case scenarios all this leads to risky driving practices, and ultimately, car accidents which further impact the system.

According to the latest INRIX traffic report (Pishue 2024) the US economy as a whole lost $70.4 billion dollars in productivity in 2023, while on an individual level each driver lost an average of 42 additional hours to congestion. This paper aims to show the feasibility of simulating and predicting traffic patterns for metropolitan areas, in hopes that this information can be used to react to congestion in real-time.

Literature Review

Behavioral simulations provide a useful means for modeling traffic flow in both microscopic and macroscopic approaches. Specifically, the Discrete Event System Specification (DEVS) applies a framework for scaling traffic simulation experiments based on an object oriented design (Chi, Lee, and Kim 1995). Chi's implementation of DEVS relies on defining a system entity structure (SES) based on declaring the objects or entities that need to follow both an internal transition function, and react to external events as time advances in the simulation. Those entities follow a procedural model base (MB) to execute the simulation. This paper draws inspiration from their application of this framework to arterial traffic, but their methods prove successful in modeling highway traffic as well.

Himite (2021) demonstrates an object oriented simulation using pygame which visualizes a variety of different traffic structures at a micro scale that we simplified and adapted for our macro-level application. The article was accompanied by the trafficSimulator.py package available to be installed via pip; however, the current build of the python package is significantly different from the one described in the article, and the package proved to be less easily manipulated to account for simulation variables than a script built without using the package. Still, a future simulation may wish to revisit the implementation by Himite to investigate how other features not in our code might be simulated in the future.

Design & Methodology

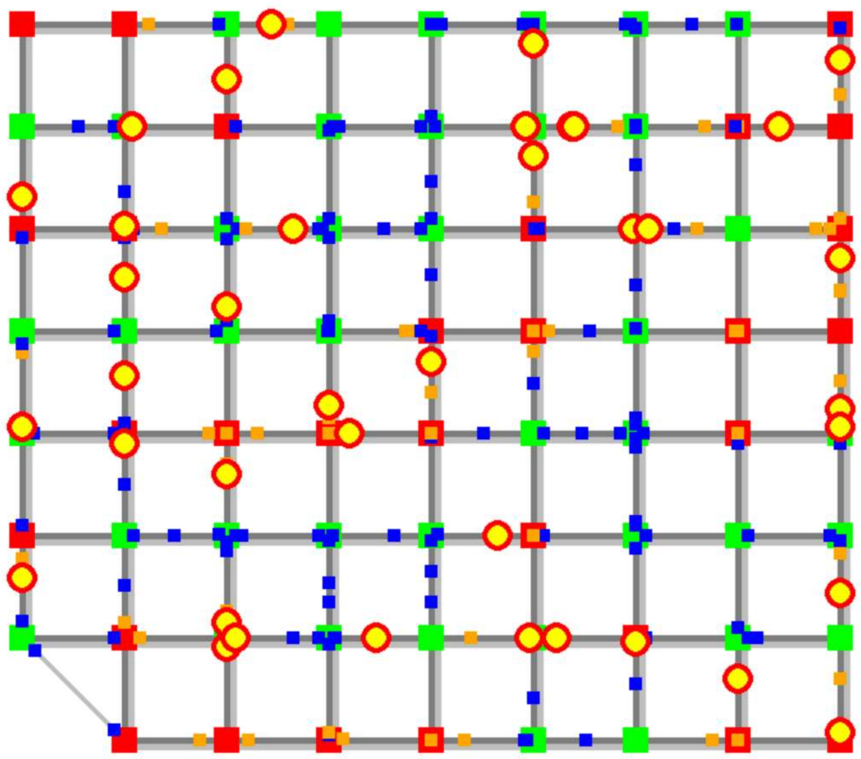
Scope: The location we selected as our subject of study was a portion of the Loop neighborhood in downtown Chicago, bounded in the north and west by Wacker Dr, which curves at the northwest corner of the boundary, in the south by Jackson Dr, and in the east by Michigan Ave (Appendix 1, Appendix 2). The major roads and intersections made up the nodes and links to a network graph that can be visualized as a 2-dimensional grid. For simplicity, the roads in our boundary were modeled with one lane of traffic going in each direction. Each intersection is signaled, with pre-set durations for each light, all randomly initialized. Roads may also contain accidents that are randomly spawned. Vehicles are randomly placed on the grid, and have a randomly generated destination to reach. To travel through the network, the simulation implements Dijkstra's algorithm.

The traffic elements present in the location to be modeled include the vehicles, intersections (as traffic lights), road segments, and accidents (demonstrated as a variable of the road) with each containing various states, inputs, and outputs. A key part of the design of the simulation was determining what we did and didn't want to include as features. For each feature we wish to model, we also needed to consider our ability to scale further as part of our benchmarking test. For this reason, we focused only on uniform vehicles, roads, and intersections, with only the vehicles being scaled between a density (i.e how many vehicles are simulated at one time) of 500 vehicles and 1000 vehicles and the frequency of accidents.

For benchmarking, we developed highly comparable code in both Python and Go, marking the ease of development between the two scripting languages. Once functional prototypes were available, we made an evaluation of whether we wanted to keep the exact function of the Go code, or of the Python code, with the script of the other language being transitioned to exactly resemble the kept code in order to have a sufficiently similar comparison for the benchmarking on execution time to be valid.

Implementation: We used an object oriented framework with the objects and features defined in Table 1 of the Appendix. An internal transition function defines how state changes over time without any input, while an external transition function defines how state changes when an event occurs Finally, the output function generates outcomes just before internal transitions occur. These variables and functions allowed us to simulate appropriate changes in the entities being modeled as they interacted with each other within the simulation. This would also be the point at which we would add additional types of objects, variables, and functions in the event that we wished to increase the precision of our simulation.

Figure 1: Simplified traffic grid simulation



Results

Once the proof of concept had been demonstrated with the two languages, we determined that Go was able to create our preferred simulation. We then adapted the Python code to more accurately resemble the Go code results, finishing with a pair of simulations as comparable to each other as possible. While they are simple models of a much more complex system, the models show that it is possible to simulate traffic patterns by varying multiple factors within the traffic system. Values pertaining to the scope of this project include the number of cars in the system, the rate of unforeseen causes of congestion (in this case accidents), and the changing of traffic signals. The models are also adaptable to any city by overlaying traffic simulations over each city's unique infrastructure and defining their own traffic conditions; as adaptable as the model is, the concept could be applied anywhere in the world.

Figure 2: Python vs. Go Execution Times (seconds)

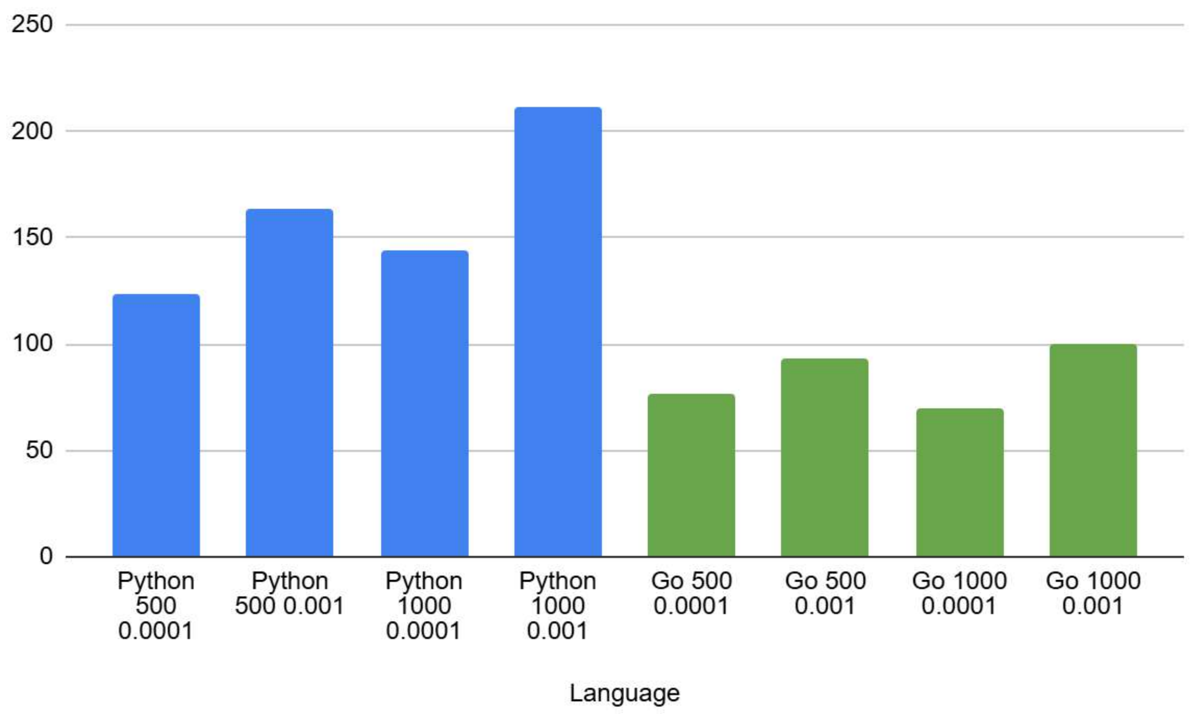
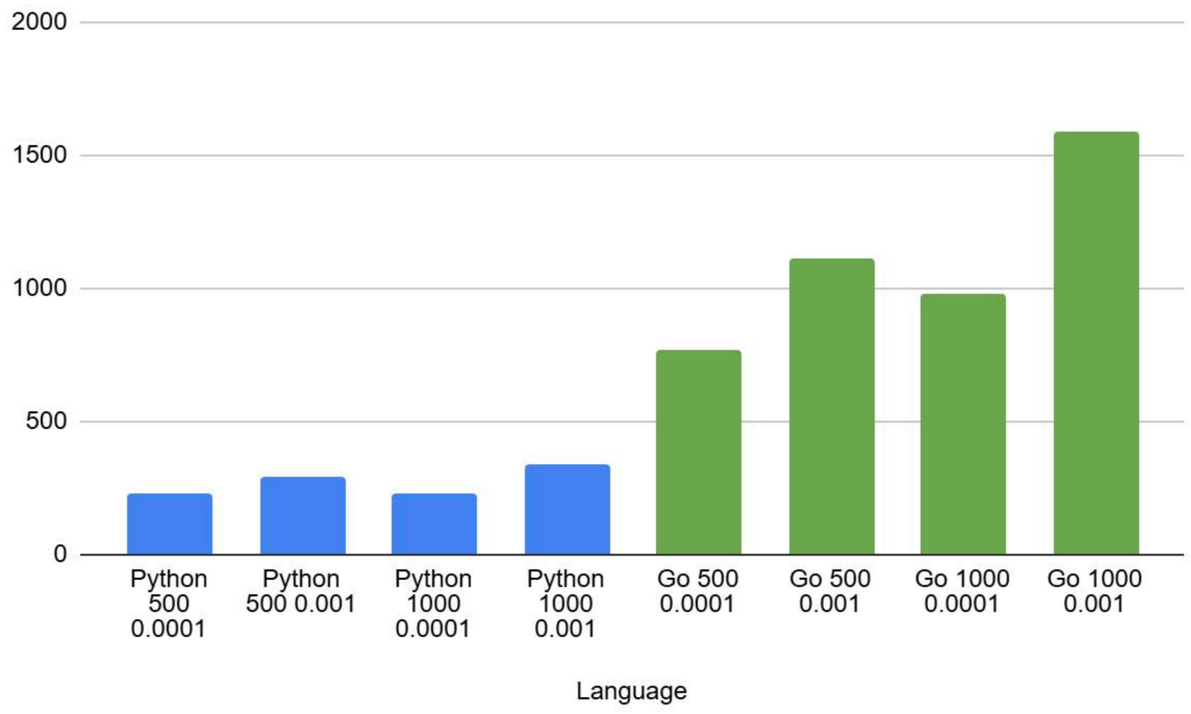


Figure 3: Python vs. Go Peak Memory (MB)



Memory appears to tell a different story. Go appears to be using more memory to conduct the simulation, this may be directly related to how ebiten, Go’s 2d game engine, operates under the hood. While ebiten is very similar to Pygame, it’s not a direct replica.

Conclusions and Insight

In conclusion, the Go simulation proves to be a viable method and tool for further studying traffic systems and how to best manage traffic flows throughout metropolitan areas. For additional development, and more meaningful application of traffic simulations, more work must be accomplished in better modelling the underlying physical infrastructure. A major challenge in further developing this system is the accurate simulation of driver behavior, helping to account for responses such as rubbernecking and lane changes. Completely life-accurate behavior models are not necessary for adequate modelling (Henclewood D et al, 2017), as there would be diminishing returns on further and further behavior accuracy. Ideally traffic flow will be controlled by traffic signal changes to minimize the effect of driver behavior. This type of modelling can then further be used to stress test any future traffic construction plans in hopes of optimizing traffic flow during the construction phase.

References

Pishue, Bob. (2024, June). “New Normal” of 10am to 4pm Hybrid Work While Congestion Grows Globally. Onward. https://inrix.com/blog/2023-global-traffic-scorecard

Himite, Bilal. “Simulating Traffic Flow in Python” Sep 5, 2021. Medium - Towards Data Science. https://towardsdatascience.com/simulating-traffic-flow-in-python-ee1eab4dd20f Retrieved 11/30/2024

Chi, Sung-Do, Ja-ok Lee, and Young-Kwang Kim. “Discrete Event Modeling and Simulation for Traffic Flow Analysis” 1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century, Vancouver, BC, Canada, 1995, pp. 783-788 vol.1, doi: 10.1109/ICSMC.1995.537860. https://ieeexplore.ieee.org/document/537860

Henclewood D, Suh W, Rodgers MO, Fujimoto R, Hunter MP. A calibration procedure for increasing the accuracy of microscopic traffic simulation models. SIMULATION. 2017;93(1):35-47. doi:10.1177/0037549716673723

AI Disclosure

Use of GenAI assisted the brainstorming through comprehension of notes taken on DEVS and formulating initial data structures. The conversation used here is linked below.

https://huggingface.co/chat/conversation/6747ed46081f33b605991aa0

Appendix

Table 1: Objects & Features

Entity	**Vehicles**	**Intersections**	**Road Segments**
States	Position, speed, destination, route, status	Signal status, queue length	Traffic density, capacity, presence of incidents, speed limits
Inputs	Signal changes, interactions with other vehicles, road conditions	Arrival of vehicles, sensor input, timing offset	entry/exit of vehicles, incidents
Outputs	Movement to next segment, status updates	Signal changes, releasing vehicles	Traffic condition, incidents
Internal Transition Function	Keep moving forward	Change signal states	Update traffic density
External Transition Function	Respond to signals, incidents	React to sensors	Update when incidents occur / clear
Output Function	Proceed along route	Change traffic signal	Report incidents
Time Advance Function	Time until movement	Signal duration	Time between incidents

Appendix 1: Links (Streets)

https://github.com/ejones77/460-term-project/blob/main/go_traffic/links.csv

Appendix 2: Nodes (Intersections)

https://github.com/ejones77/460-term-project/blob/main/go_traffic/nodes.csv