Using Reinforcement Learning in Detection Systems to Modernize Traffic Control Algorithms

Research
What is the problem we are trying to solve?

$1160 — "The average cost of the time lost in rush hour traffic is $1160 per person."

5.7 — "Congestion results in 5.7 billion person-hours of delay annually in the United States."

44% — "On-road vehicles are responsible for 44 percent of all carbon dioxide emissions in the United States."

Existing Solutions

Name	Summary	What we do Differently and Drawbacks
Verizon	"Smart roads are possible thanks to intelligent traffic systems that use data... cost-effectively... (and) cellular communications to assess system performance in near real time."	• Primarily uses/focuses on the Internet of Things (IOT) • Difficult to install
Connected Signals	"delivers real-time, predictive traffic signal data. Their engineers have developed sophisticated techniques to predict upcoming signal states."	• Does not change the light; adapts map directions based on light • Does not improve commute time (simulation)
Surtrac	"can do it cost-effectively using cellular communications to assess system performance in near real time."	• Does not use simulations to quicken the duration of the data-collecting phase • Inactive since April 2017

Must improve the rate of cars/hour by a sufficient amount

Software must be able to be run and make decisions in real time

We need an Artificially Intelligent Traffic Controller (AiTC)

Must have the capabilities to direct randomized traffic flow patterns

Must be able to be implemented in a feasible manner

About Traci + SUMO
How did we collect the traffic data needed?

- **SUMO** – **Simulation** of **Urban Mobility**
  - Simulation of vehicles moving through a network of roads
- **TraCI** – **Traffic Control Interface**
  - Python interface to SUMO; Can retrieve **values** of simulated **objects** and **manipulate** its **behavior**
  - Has GUI for visualization

The GUI of a run, with the center depicting a tL. The colors indicate its **Phase** (rrrrgrGGrgrrrrrgrGGrgr)

Current Simulated Street

Improvements From Last Year

Realistic Intersection: Last year, we had a simple **2-Way Intersection** that had no turn lanes and only 4 light phases. This year, we used OpenStreetMap to generate a **real 4-Way Intersection** (Blossom Hill Rd. and Meridian Ave.) that contained 16 light phases (including yellow).

Last Year's Simulated Project

Reinforcement vs Supervised: We decided to switch to the cutting-edge machine learning technique, **reinforcement learning**, due to the complexity of supervised learning on a **larger-scaled street**.

Real Data: We talked to a traffic engineer to ensure that our environment was accurate. Our simulation's traffic flow emulates morning rush hour on this street using probabilities from San Jose's traffic control database.

Traffic Light Phases

Green for...	NL	NT	NR	EL	ET	ER	SL	ST	SR	WL	WT	WR
Traffic from N	GG	GG	g	gG	rr	r	rr	rr	g	rr	rr	g
Traffic from E	rr	rr	r	gr	GG	G	gG	rr	r	rr	rr	g
Traffic from W	rr	rr	g	rr	rr	g	GG	GG	G	rr	rr	r
Traffic from S	rr	rr	r	rr	rr	g	rr	rr	r	GG	GG	gG
Traffic turning left (N → E, S → W)	GG	G	rr	r	r	g	rr	r	g	r	r	g
Traffic straight (N↔S)	rr	G	rr	r	r	g	rr	r	g	G	G	gG
Traffic turning left (E → N, W → S)	rr	r	g	rr	GG	G	rr	r	g	r	r	g
Traffic straight (E↔W)	rr	r	g	gG	GG	G	rr	r	g	rr	r	gG

* All traffic phases also have **matching phases for the yellow light** (Yy replace Gg)!
Ex. grrygrrrrgrryygyyyy

Turns: [r, G, g] + [r, G] Throughs: [r, G]
- r = red
- G = protected green
- g = default green (not priority)

Since East and West left turn lanes are green, the phase is rrrgrGGrgrrrrgrGGrgr

Satellite View

OSM View — JOSM Editor

SUMO Simulated View

OSM and JOSM (street covert/editing) — NetConvert Utility (creates TraCI.xml)

Training + Testing AiTC
How did we create and test our algorithm?

How we trained our network

CITY OF San Jose — CAPITAL OF SILICON VALLEY

About OpenStreetMap and the San Jose Transportation Department

OSM: OpenStreetMap is a free editable **map of the world**, and can be **converted** into SUMO streets (**map.net.xml**). Also contains information about the **intersection system**.

JOSM: JOSM is a free software desktop **editing tool** for **OpenStreetMap** geodata (Java). This is **directly compatible** with **SUMO**.

SJ Department of Transportation: **Oversees transportation** quality and services. We spoke with Ms. Lily Lim-Tsao, **Traffic Operations Manager**. She talked to us about current traffic solutions and gave us realistic **data for our simulated traffic flow**.

Reinforcement learning is a self-teaching/learning system that learns by trial and error; changes actions by trying to maximize its rewards; our reward system looks for the improvement between traffic phases.

19 State Inputs
- **rate**- The cars that have **passed** the traffic light **over** the number of **steps** the tL has **spent** in that phase (1 Value)
- **halt** - # of **halted cars** at the traffic light in each lane (16)
- **phase_time**- # of **steps** a traffic light has spent in the **current phase** (1)
- **curr_phase**- The **current state** the traffic light is showing (1)

Next_State
- Advances state following an action to compute:

Reward
- Computed after agent acts to judge improvements of its decision
- Determined based on the improvement in the amount of cars halted and passing through

* Seed - a reproducible set of random probabilities that controls the amount of cars launched in the program

Reward Function

```
def calc_reward(old_halt, new_halt, passed):
    """
    old_halt = cars halted before action
    new_halt = cars halted after action
    passed = cars passed after action
    """
    reward = 0
    if(passed == 0):
        if(new_halt == 0):
            reward = 0
        else:
            reward = - new_halt
    else:
        if(new_halt == 0):
            reward = 8 * passed
        elif(change > 0):
            reward = 4 * passed
        elif(change == 0):
            reward = passed
        else:
            reward = change
    return reward
```

Reinforcement Q Learning Model Architecture

Key Concepts of Reinforcement Learning

Environment: Simulation or real world where the measured events take place.
Observations: Simulator or sensor data extracted from the environment.
State: Observations of the environment at a particular point in time.
Agent: Takes action based on observed environment to improve future state.
Action: something the agent does to change the environment.

The **agent observes** the current simulation after changing the **environment** and generates the action.

SUMO creates a randomly seeded **environment**, and implements the action.

Every n time steps, the **agent** gets input values from its current **state**.

Action: Agent will choose which traffic light pattern to use based on current traffic **environment**

Weighted Connections = Nodes = ●●●●

19 Inputs — 604 Nodes (Hidden) — 1166 Nodes (Hidden)

Input Layer — Layer 1 — Layer 2 — Output Layer

- **Q-Value = 8 outputs** = categorical model
  - Largest value determines highest probability of producing most successful state

Observations — State — Environment — Next State — Action — Reward* — Agent — Random* — Seed

*When Training

Learner Class in DQN.py

- Exploration Rate is the rate at which our program will output a random value
- Starts with all outputs being randomized; slowly decreases until almost all actions are generated by AI

Epochs passed vs self exploration rate
f(x) = 0.999995^x

Time vs. factor of prediction discounting
f(x) = 0.94^x

Discounted Total Reward
$R_t = \sum_{t=0}^{\infty} \gamma^t r_t = \gamma^0 r_0 + \gamma^1 r_{t+1} + ... + \gamma^{n+1} r_{t+n} + ...$
γ: discount factor

- We create a neural network that allows us to take future rewards into account in our training.
- This network allows us to get a better computation of the reward value based on predicting benefits
  - **Discount Factor** = Self.gamma = 0.94

≈0.89 Y_2 Y_3 Y_4

Q (Replay) Neural Network created and used for additional training

Reward — Next Rewards
1 | -1 | 0 | 1 | 1 | -1 x 0.94 | 0 x 0.94² | 1 x 0.94³

Training Methodology

- Each Epoch contains a maximum of 1204 simulation steps
  - Reward Calculated a max. of 172 times
- Every 7 time steps, a reward is calculated and the phase is updated
  - Longer than the timing of a yellow light
  - Enough data to be useful to the training
  - Phase is long enough to let cars pass
- Reinforcement Learning vs. Supervised Learning
  - Determines labeling of good vs. bad on its own
  - Trains at a slower rate
- Why a Categorical Model Architecture?
  - 8 outputs
    - Since it is categorical, both outputs do not have to equal one
      - More accurate

Training Progress Reported at Each Epoch
Predictor: How many times the agent acts on environment
Map Seed: Repeatable, uniquely generated traffic flow
Passed: Cars passing through junction
End Sim Step: Time when all cars reach end of road (max. 1204)
Lost Time: Avg. time lost for all simulated cars
Reward: $R_t = \sum \gamma^t r_t$ (γ = discount factor; t = 7i)
Cars Left: Cars still on road after time ends
Performance: End Sim Step x Lost Time

By customizing our reinforcement learning model and using multiple outputs, we optimized the capabilities and speed of our model and the traffic light system

Our prototype shows that it is **feasible** and cost-efficient to create an AI Traffic Controller.

We used USB addressable LEDs, arcade buttons, various python libraries, and more to create our **x86 embedded system** that models **Blossom Hill Rd. and Meridian Ave.** As needed, we drilled, soldered, and assembled our various acrylic panels and electronic components.

Microprocessor board around $250

Runs on 5V USB Power Supply

Simple USB **cameras** can be connected to the board to **recognize and count cars** in a specific direction and lane. There are many **open-sourced resources** (like openCV or YOLO AI object detection) for this car object recognition. This can replace **Induction loops are very expensive**, high maintenance, and have limited range. This solution would help reduce cost, labor, and inefficiencies at the intersection.

Tuning + Optimization
What is our design, debug, and test process?

How We Programmed AiTC

- **Failsafe** function for rare unexpected or unresponsive action

Cmd Line: aitc.py -h

Failsafe function

- **Debug** Commands
  - Command Line **Switches/Logs**
  - **Printing** statements with Verbose levels
- Tuning
  - **GUI-mode** for visual confirmation
  - Program uses **constants and procedures** for easy **modification** of key functions
  - Clean simulation to **remove warnings** and make behavior more **realistic**
  - Precise **commenting** for readability

Results + Conclusion
How well does AiTC work? What is next?

Results

We ran a diverse number of simulations. Our figure of merit (score) combines the average time loss and the time needed for all trips to be completed.

$$Score = (avgWait_{FIX} * tripTime_{FIX}) / (avgWait_{AI} * tripTime_{AI})$$

AiTC provides **1.26x (avg)** improvement in quality score compared to current fixed timing traffic lights.

Score Improvement Ratio

300 Simulations of AiTC vs Fixed Traffic Lights

Implementation

In order for AiTC to work, we will need to have a camera and a processing unit that can recognize and count vehicles.

Since many lights have already installed cameras, we can use object recognition systems to count cars waiting at an intersection.

Input from camera → Counts cars → Takes in current state and sent to agent for inference → AI computes phase output action → Traffic light acts upon AI's instructions

Big Idea – Why does this matter?

A common **misconception** is that neural networks can only be beneficial if multiple traffic lights are **linked together, creating an interconnected system.**

Such projects, despite being the panacea of all traffic-related issues, are **expensive** and **time-consuming**.

Our research **illustrates** that even **implementing separate, unconnected neural networks** dramatically reduces traffic in a given area. Installing such versatile programs are both cost-efficient and feasible, serving as a **firm stepping stone** on the arduous journey towards an omniscient traffic light system.

Future Research

- Larger **variety of streets**
  - Train the AI to work on more **uncommon streets**

AI Techniques for winning complex games are applicable to traffic optimization

Freeway exits and entrances

Pedestrian crossings

- Try more advanced AI techniques
  - Adversarial, **Policy-Based**, Combination of multiple?
  - Allow more **flexibility with reward** system
    - Based on end result/outcome
    - Determines reward of each action by itself
- Multiple intersections next to each other to simulate city
  - Train with **connected intersections**
    - Intersections communicate with each other
  - Test **compatibility**
- Incorporate accidents, emergency vehicles, etc.
  - Give **priority** when needed
- Reach out to researchers/Publicize for **feedback**
  - Already **shared** on GitHub

Emergency Vehicles

Bibliography

- "SUMO at a Glance." *SUMO - Simulation of Urban Mobility*, www.sumo.dlr.de/userdoc/Sumo_at_a_Glance.html.
- Morgan, Lee, and Leaf Group. "The Effects of Traffic Congestion." *USA Today*, Gannett Satellite Information Network, traveltips.usatoday.com/effects-traffic-congestion-61043.html.
- "Keras: The Python Deep Learning Library." *Keras Documentation*, keras.io/.
- Gao, Bo. *Traffic Control Test Bed Documentation*, 10 Mar. 2017, intelagent.github.io/tct/post-learning-tract-tls.html.
- Amini, Alexander. "Deep Reinforcement Learning MIT 6.S191." *MIT EECS*, introtodeeplearning.com/materials/2019_6S191_L5.pdf
- "How Do Traffic Signals Work?" *How Do Traffic Signals Work*, www.traffic-signal-design.com/how_do_traffic_signals_work.htm.
- Python: Package TraCI, www.sumo.dlr.de/daily/pydoc/traci.html.