Optimizing Traffic Light Cycles

Simon Kurgan David Batarseh

June 1, 2025

Abstract

We model traffic flow using a series of elementary 1 dimensional cellular automata following rule 184. Active cells are tracked as agents and given a destination; agents at determined 'intersections' turn left, right, or go straight dependent on pre-assigned destinations. Intersections are assigned a constant timing cycle as a traffic light. Using this model, we perform simulated annealing over the output space of traffic lights to minimize congestion along lanes, where 'congestion' is defined as the number of time steps in which an agent could not move.

Chapter 1

Introduction

1.1 Problem Description

According to the U.S. Department of Transporation's Federal Highway Administration, one-quarter of traffic fatalities and about one-half of traffic injuries in the U.S. are attributed to intersections. Signalized intersections in particular present a complicated tradeoff between signal timing, safety, and mobility. They represent about one-third of all intersection fatalities, which equated to roughly 4,204 deaths in the year 2022[2]. Optimizing these cycles within safe bounds presents an opportunity to lower congestion and reduce the overall amount of unnecessary time travellers spend at signalized intersections.

Moreoever, traffic signal optimization, wherein existing traffic light cycles are optimized to reduce congestion present a uniquely high benefit-cost opportunity. Using existing infrastructure and bettered data, outdated systems can reduce disorder and congestion.

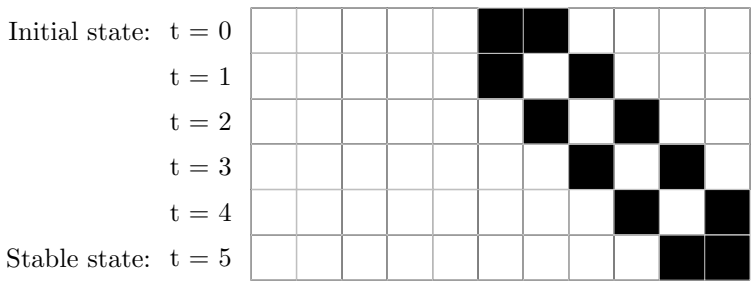
As a direct result, cities, states, and countries can lower transporation-related greenhouse gas emissions by reducing travel times on roads and waiting times at intersections. Bob Pishue from Roads and Bridges, citing the Federal Highway Administration, described that an estimated 5% of travel delays were due to waiting at traffic signals in 2004[3].

Describing the tangible benefits, Pishue further described the implementation of a network-wide signal synchronization system in Los Angeles. Before and after studies by the Los Angeles Department of Transportation found that drivers saved 12% on travel times due to signal upgrades[3].

Through this report, we seek to model a traffic network, explore the constraints that cause congestion, and in turn expirement with traffic light cycles to find an optimal assignment. Our expressed goal is to lower congestion and travel wait time, as described in our assumptions and model.

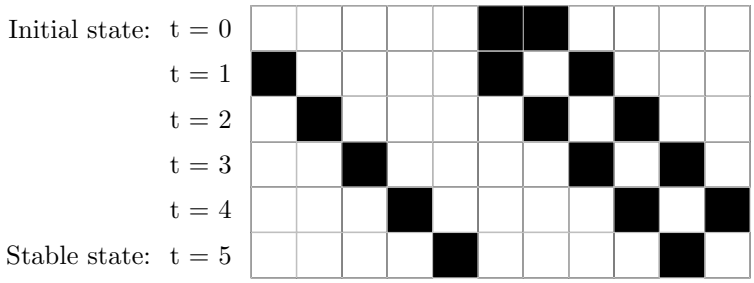
1.2 Rule 184 Definition

Rule 184 is a one dimensional cellular automata that is often used to describe traffic flow[1]. However, it can be loosely understood as "if the space to the right of a one is a zero, move that one to the right". Rule 184 has several properties that are desirable for modeling traffic. Unlike most cellular automata, the number of 'one' cells is constant. Additionally, it is computationally simple and easy to modify. Below is an example of rule 184, given that once the cells reach the right edge of their bounds, they are no longer modified.



We note that our model has a series of seperate systems that are modelled with 184, as opposed to only one. A directed street is one such system. Moreover, our systems can accept a new cell at the back of the system and cells at the front have the capacity to be moved to another subsequent system or to be removed from the network entirely. Furthermore, traditional cellular automata is infinite, whereas our system has strict bounds proportionate to street length.

Below is one such example, where a cell is added to the back at $t = 1$, and the cell reaching the front at $t = 4$ is removed.



Chapter 2

Simplifications

We assume a series of things for our model to function properly, and to simplify the chaotic real-world data present in a road network.

2.1 Traffic Assumptions

- Our model only accomodates single-lane one-way or two-way opposite-flowing streets.
- Intersections can either be with light cycles, or without. The latter represents free uninhibited flow for agents.
- Congestion along a street, and a travel itinerary, is defined as experiencing wait time. Wait time is caused by being inhibited from moving at the defined speed on a roadway by traffic along the same route, or by the inability to merge into desired roadways due to congestion.
- Our model assumes that all cars travel at the exact same speed, i.e. one potential unit per tick. This can be extended to introduce variability.
- We heavily select against traffic buildup; each successive car adds a wait time of the distance from it to the intersection, as opposed to a constant factor. This is done to prevent buildups, which can cause congestion in future intersections, even when one intersection may clear after turning green.

2.2 Assumptions of Agents

- We note that rule 184 is known to generate 'cautious' agents; that is, agents will always be one car space apart, assuming that the agent is moving.
- Agents do not accelerate; they either move or do not move at each timestep dependent on if the state of the roadway and intersections afford it. They can move at most 1 unit per tick.
- Agents have a perfect recollection of their travel plans and move precisely along the shortest path from their defined source and defined destination.
- Agents choose their source and destination at random from a precomputed set gathered from the defined network. This remains unchanged throughout their time in the network.
- Agents turning left do not block cars from the other side of the street from turning left or going straight. For example, if two agents are opposite each other (180 degrees) and approach an intersection, they can both turn in the same direction (one left, one right or vice-versa) without any accident occurring
- When slated to move through an intersection, the agent either leaves the network if arriving at their destination or transfers to the start of a neighboring roadway in 1 tick.
- Agents drive uniformly sized vehicles.

2.3 Assumptions of Streets

- Two-lane roads are a pair of oppositely directed streets.
- One-way roads are defined by a single directed street.

- Streets exist between two intersections, one or both of which may be unaffected by traffic light cycles.
- Streets are defined by their distance and speed. Distance corresponds to the number of average sized cars that can fit on the roadway, and speed is relative to the set tick rate.

2.4 Assumptions of Intersections

- Intersections have no delay in switching light cycles.
- Intersections do not hold custom cycles or states to describe traffic patterns such as protected turns.
- Intersections have only two cycles. The first cycle represents a green light for left-right traffic and a red light for top-bottom traffic. The second cycle represents a red light for left-right traffic and a green light for top-bottom traffic.
- Intersections hold the same values throughout a simulation.

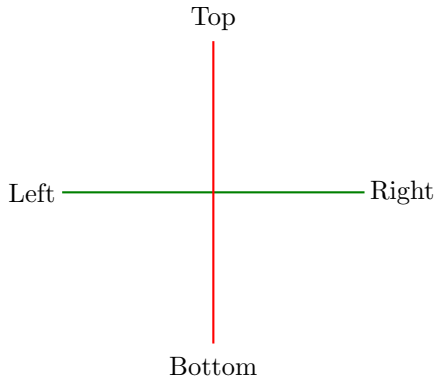


Figure 2.1: Cycle 1 is active (left to right traffic)

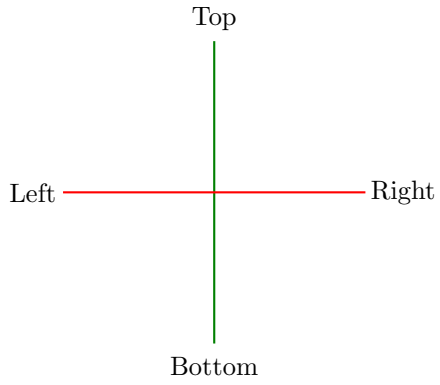
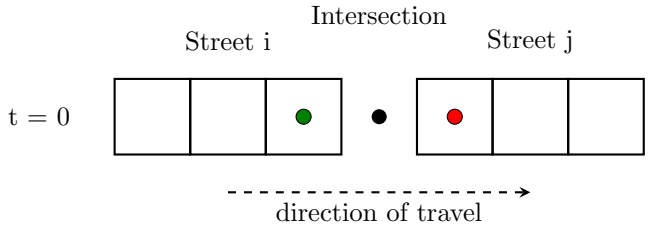


Figure 2.2: Cycle 2 is active (top to bottom traffic)

2.5 Assumptions of Our Network

Due to agents traveling strictly without cycles, we can simplify our graph to remove untraveled edges and process each street in a topological way. We want to reflect consistency in our model throughout a set of trials and using an arbitrary order to process roads in the network introduces variability.

The figure presents an initial state to illustrates how the processing order of two streets can affect the state of the streets. Each array is a street, and if a slot is occupied by an agent it is filled with a colored node. Street i has green agents, whereas street j has red agents for illustrative purposes. An agent exiting from street i travels to street j through an intersection which has no light cycles.



As we are iterating over streets i and j in a simulation with random ordering, we have two cases. In the first, we process Street i first and thereafter look at Street j. In the second, we do the opposite.

If Street i is processed before Street j, the agent intending to travel through the intersection is blocked with the assumption that the traffic state of Street j is current. If Street j is processed before Street i, the both agents iterate forward 1 step as the agent at the front of Street i sees a free slot at the back of Street j.

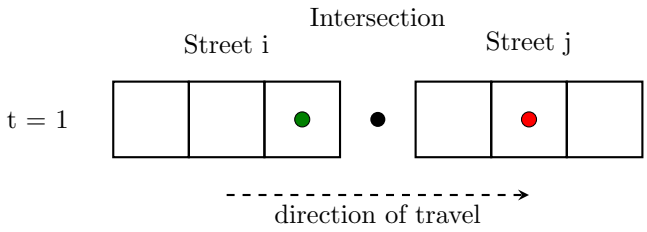


Figure 2.3: Evaluated traffic state after 1 time tick if Street i is processed first and j second.

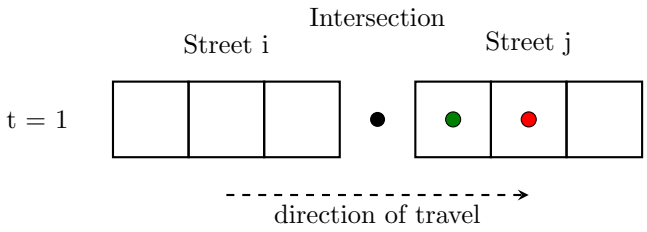


Figure 2.4: Evaluated traffic state after 1 time tick if Street j is processed first and i second.

2.6 The Effect of Simplifications

Agents being cautious is somewhat accurate to real life; for instance, cars do not all stop and start in an instant chain at a green light. Each car tends to wait for the car ahead of it to go, creating a distance between each of them. Our model implicitly affords that due to the order in which streets and agents are processed.

We are excluding a lot of variability and noise by using simpler traffic rules, agents, and intersection definitions. For instance, if were to open our model up to have multiple parallel lanes of same-direction traffic we would have to correctly model the likelihood of cars switching lanes. Moreover, cars and intersections would become more sophisticated as agents attempt to drive strictly in the lanes necessary to execute their desired path.

An agent might for example be stuck at an intersection waiting for the rightmost lane of a street to decongest despite free flowing traffic in other lanes, where as our model reduces this to an operation of strictly one lane.

Perhaps the greatest assumption we are making is in encoding perfect responsiblity for all of our agents. There are no quarrels or accidents and instead impossibly attentive uniform drivers whose implicit intent is to decongest the roadway they are on.

Chapter 3

Modeling

3.1 Modeling Time

We use discrete time events to model time through the simulation. For each cellular automata, we model using 10 mph, a relatively low speed. Each car would be moving at approximately one car length a second, so to match our cellular automata we assume a tick rate of 3600 per hour. We specifically use 10 mph to get that each tick is one second. Modelling multiple speed limits at a time would simply require moving each street at a specified fraction of ticks.

3.2 Modeling Agents

Each cell with a 'one' in our system represents a driver on the road. We model each driver as an agent with the following set of rules:

- Agents must have a destination node and a source node.
- Agents seek to travel on the shortest path from their source to their destination.
- Agents never backtrack, meaning they avoid cycles when travelling.

As such, our representation of the street network in effect represents a directed acyclic graph for each agent, despite having clear cycles.

3.3 Modeling Intersections

An intersection is defined as a node, from which and to which a lane (arc) connects. Intersections are of two types: terminal and intermediary. A intermediary intersection has outgoing lanes, meaning traffic can stem from it. A terminal intersection has no outgoing lanes, and serves as an exit for agents.

Each intersection's traffic lights are modelled using two numbers, $c_1, c_2 \in \{x \in \mathbb{N} \mid x = 5k,\ 3 \leq k \leq 40,\ k \in \mathbb{Z}\}$. The first number c_1 is associated with the number of ticks the light is green in the west-east or left-right direction, and red for the north-south or top-bottom direction. The second value c_2 represents the number of ticks the light is green in the north-south or top-bottom direction and red for the west-east or left-right direction.

Let $s = c_1 + c_2$ and t be the current tick number such that $t > 0$ and $t \in \mathbb{N}$. The current active cycle can be obtained using modulo. Let v be the resulting value from ($t \bmod s$). If $v \neq 0$ and $v \leq c_1$ then we are in c_1. If $v = 0$ or $v > c_1$, then we are in cycle c_2.

Example: Suppose $c_1 = c_2 = 3$. The table below illustrates the found cycle.

t	$v = t \bmod (c_1 + c_2)$	Active Cycle
1	1	c_1
2	2	c_1
3	3	c_1
4	4	c_2
5	5	c_2
6	0	c_2

3.4 Modeling Lanes

Each street and direction is associated with a one-dimensional cellular automata modeled by rule 184. We note that rule 184 is directional, and we denote 'forward' as being the direction in which 1's (cars) move, and the 'front' as being the maximal position in the system. An equivalent definition would be the element at the maximal index of a contiguous array. We also define the 'back' as the minimal position in the system. Index 0 if related to the same analagous contiguous array.

Each street is associated to a source and destination intersection. Implicitly, the source intersection of a lane is intermediary, however the destination can be of either type. If the destination is terminal then a '1' cell is set to a '0' when processed at the front. In effect, the agent has left our network and is no longer impacting traffic.

If the destination is an intermediary intersection, the agent at the front of the current system is slated to be transferred to the back of a lane stemming from the intersection. This lane corresponds to an edge in the shortest path the agent seeks from its source to its destination. In effect, given multiple possible lanes to travel to at an intersection, an agent prefers the path strictly necessary to reach its destination.

Two conditions must be met in order for the agent to transfer. First, the light at the intersection for that direction of travel must be green. Second, the back of the intended lane must be unoccupied. If both are not met, the agent stays in place. If they are, the front '1' cell is set to '0' and the back '0' cell of the receiving lane is set to '1'.

3.5 Our Simulated Case

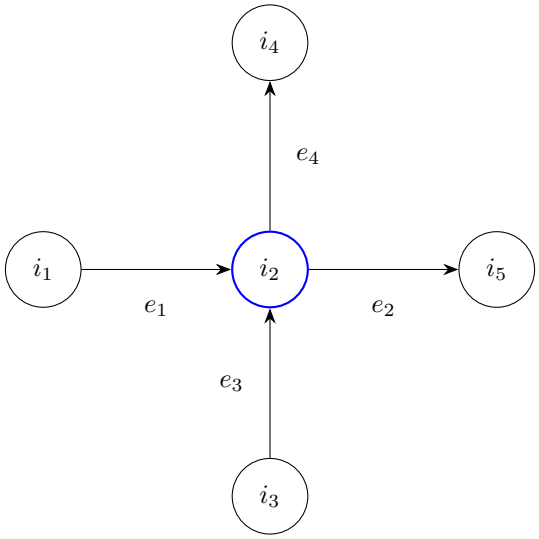
Our base case is two one-way roads with two segments passing through an intersection.

Let i_2 represent an intersection with associated cycle values $c_1 = 3$ and $c_2 = 3$. Let i_1, i_3, i_4, i_5 all have the associated values $c_1 = 0$ and $c_2 = 0$ to represent an unrestricted intersection with no signals.

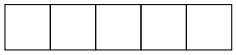
There are 5 intersections, I = { i_1, i_2, i_3, i_4, i_5 }. There are 4 roads such that E = { e_1, e_2, e_3, e_4 }.

Our directed adjacency list representation is as follows:

- i_1: $[i_2]$
- i_2: $[i_4, i_5]$
- i_3: $[i_2]$
- i_4: $\emptyset$
- i_5: $\emptyset$



Each road $e_i \in E$ in this case has a length of 5, and a speed of 1 for simplicity. This is a representation of the automata system in each edge at t = 0. It represents 5 open slots for cars.

t = 0 

Two conditions must be met in order for the agent to transfer. First, the light at the intersection for that direction of travel must be green. Second, the back of the intended lane must be unoccupied. If both are not met, the agent stays in place. If they are, the front '1' cell is set to '0' and the back '0' cell of the receiving lane is set to '1'.
We 'spawn' agents in accordance to a discrete Poisson distribution. Using a sample of 2400 cars going through an intersection on average, which would be a 'high traffic' scenario, we spawn cars with a probability of $2400 * P(x)/3600$ for each lane each tick, where $P(x)$ is the probability that a car going through this intersection comes through that lane, assuming that each tick represents 1 second. For e_1, e_3, we assume that there is a 50% chance that any car going through this intersection begins at either one of e_1, e_3. This gives us a probability of 'spawning' a car per tick of 33.33%.

Chapter 4

Results

We analyze the results from a series of source percentages given a single intersection. Using 2400 cars over an hour as the 'crowded' time for an intersection, with an even 50:50 split, we get the following plot:

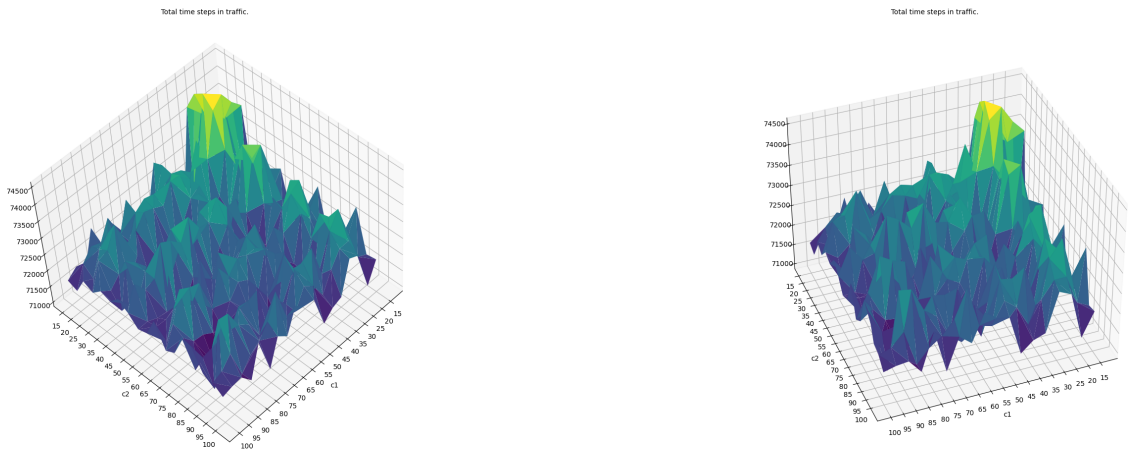


Figure 4.1: Each data point sampled over 5 trials.

By using the dual annealing solution implemented in scipy and allowing all natural number inputs in the range $(5, 120)$, we get an optimal solution of $c_1 = 49$, $c_2 = 52$, with an average wait time of 62976 ticks per hour. We note that the Federal Highway Administration arrives at an average cycle time of 120 seconds when assuming very similar conditions[4]; our model is likely fairly accurate in this scenario.

We achieve interesting results after modifying our model with differing probabilities for each intersecion. At a 55%, 45% split, our model begins to heavily skew towards one side:

Using dual annealing, we get an optimal cycle value of $(102, 18)$, with an average wait time of 66942 ticks. Even more astonishing is at a $60\%, 40\%$ split, we get an optimal cycle value of $(162, 15)$! We believe this is a result of our skewing against multi-car buildups at intersections.

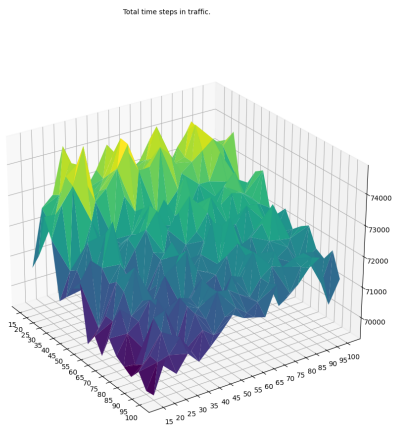


Figure 4.2: Each data point sampled over 5 trials.

Chapter 5

Code & Algorithms

This section documents the code used to represent the modeled components of our network. Comments include supplementary descriptions where necessary to replace length code.

5.1 Data Classes

Lane Class

```
class lane:
    def __init__(self, length):
        self.agents = [False for x in range(length)]
        self.length = length
        self.cumulative_wait_time = 0
        self.destination = None

    def spawn(self, agent):
        # takes in an agent object and spawns it at index 0 if unoccupied

    def process_tick(self, time, network):
        # processes the array of agents from back to front, following predefined automata and
            simulation rules.
        # procesess the handoff of an agent after reaching the front to either an
            intersection, or its removal
```

Agent Class

```
class agent:
    def __init__(self, source, destination, network):
        self.source = source
        self.destination = destination
        self.path = network.shortest_path(source, destination)[1:]
```

Network Class

```
from collections import defaultdict

class network:
    # note: vertex == intersection
    def __init__(self, graph):
        self.graph = graph # intersections -> [ lanes ]
        self.topo_sorted_vertices = self.get_toposorted_vertices()
        self.sources = self.get_sources()
        self.destinations = self.get_destinations()

    def get_sources(self):
        # return all vertices with no inbound arc
```

```
    def get_destinations(self):
        # return terminal vertices

    def get_toposorted_vertices(self):
        # gets toposorted order of network as [ vertices ]
        # to be used carefully and sparingly, as we must only operate on a network with no
            cycles

    def shortest_path(self, source, destination):
        # return shortest path from source to destination as [ vertices ]
```

5.2 Simulation Class [Base Case]

```
def simulate_base_case(network, general_spawn_probability):
    # https://wsdot.public.ms2soft.com/tcds/tcount.asp?offset=0&id=357290207&a=206&sdate
        =2024-10-17&local_id=CS05860&classDate=&speedDate=&gapDate=&count_type=%27VOLUME%27
    totalCars = 2400
    simulationDuration = 3600 # in ticks (seconds), represents an hour

    spawnFactor = totalCars * general_spawn_probability / simulationDuration

    suma = 0
    sumb = 0
    def simulate(tick):
        # for each source and probability, use random to determine if a agent needs to be
            spawned
        # spawn cars at beginning of each tick at source nodes

        for intersection in network.topo_sorted_vertices: # process topologically
            for road_from in network.graph[intersection]:
                print(road_from)
                road_from.process_tick(tick, network)
        return [x, y]

    while currentTick < simulationDuration:
        t = simulate(currentTick)
        suma += t[0]
        sumb += t[1]
        currentTick+=1

    return sum(lane.cumulative_wait_time for lane in lanes)
```

5.3 Averaging and Plotting Class

```
network = basic_network # predefined

points = [[], [], []]
for i in range(2,21):
    for j in range(2, 21):
        temp = 0
        points[0].append(i)
        points[1].append(j)
        for k in range(10):
            m = simulate_case([i * 5, j * 5]) # call simulation_lass
            temp+=m
        points[2].append(m/10)
        print(f"{i}, {j}: {m}")

fig = plt.figure()
ax = fig.add_subplot(projection = '3d')
ax.plot_trisurf(points[0], points[1], points[2], cmap='viridis', linewidth=0, antialiased=
    True)
plt.show()
```

5.4 Complexity Estimates

5.4.1 Toposort

The network class, defined once per problem, makes use of toposort in determine the processing order for different automata systems (lanes), which runs in the order of $O(V + E)$ where V is the number of vertices and E is the number of edges in a directed acyclical graph. It uses $O(V)$ auxilary space for the creation of the resulting stack[5].

5.4.2 Shortest Path

This algorithm uses the property of topological sorting, and lack of negative edge weights, to calculate a single source shortest paths. We opted to avoid an Algorithm like Dijkstra's to reduce the overal complexity of calculating each necessary path. It works by initializing all vertices to an infinite value and sources to 0, thereafter processing the nodes topologically and updating values. Our code extends this algorithm by returning the path. We make note of the optimal step from each vertex in the calculation and travel recursively backwards to return the path passed to the agent. This runs in the order of $O(V + E)$. Auxilary space is O(V + E) due to the set of paths being returned[6].

5.4.3 Simulating Values

The assignment of intersection values to solve the problem has a time complexity dependent on two factors: the number of intersections and the set of potential assignments of c_1, c_2 to each distribution.

Let the number of Intersections be I, and the set of potential assignments to one cycle value c_i be A. The complexity of finding all combinations is $O(|A|^2 * I)$.

Practically, our restriction of c_i to a range with a step size k leads to a constant run time. The set we employed ranged from $[15, 200]$, and with a step size of $k = 5$ had 37 elements. For our base case with just one intersection, this is 37^2 or 1369 operations. Additionally, the dual annealing done by scipy is done in $O(|A|^2 * I)$ time as well.

Bibliography

[1] Fuks, Henryk, and Nino Boccara. "Generalized Deterministic Traffic Rules." *International Journal of Modern Physics C*, 09(01):1–12, February 1998. https://doi.org/10.1142/s0129183198000029.

[2] Federal Highway Administration (FHWA). "About Intersection Safety." *About Intersection Safety — FHWA*. Accessed June 1, 2025. https://highways.dot.gov/safety/intersection-safety/about.

[3] Pishue, Bob. "Better Traffic Signals Save Time ... and Can Help Save the Planet." *Roads and Bridges*, October 25, 2021. https://www.roadsbridges.com/traffic-signals/article/10654415/better-traffic-signals-save-time-and-can-help-save-the-planet.

[4] Federal Highway Administration (FHWA). "Traffic Signal Timing Manual: Chapter 3 - Office of Operations." *Federal Highway Administration*. Accessed June 1, 2025. https://ops.fhwa.dot.gov/publications/fhwahop08024/chapter3.htm#:~:

[5] GeeksforGeeks. "Shortest Path in Directed Acyclic Graph." *GeeksforGeeks*. February 3, 2023. Accessed June 1, 2025. https://www.geeksforgeeks.org/shortest-path-for-directed-acyclic-graphs/.

[6] GeeksforGeeks. "Topological Sorting." *GeeksforGeeks*. April 4, 2025. Accessed June 1, 2025. https://www.geeksforgeeks.org/topological-sorting/.