Multi-Junction Traffic Controller Algorithm

Mihir Dhanakshirur

1. Introduction:

Traffic is a field which has long been excluded from the rapid developments seen in Artificial Intelligence. Traffic problems are notorious for being extremely dynamic and difficult to model. This exclusion has resulted in the persistence of static traffic controllers at most junctions of metropolitan cities. Controlling a traffic light consists of designing rules to decide which phase to apply over time. Some cities have employed adaptive traffic controllers like SCOOT [1], SCATS [2], PRODYN [3] and RHODES [4]. These systems obtain traffic measures and decide optimal control variables such as phases, splits, cycle times and offsets.

Various traffic control methods using backpressure algorithm are present in the literature. Most of them are based on the original back-pressure control used in wireless communication using real-time queue estimation by Tassiulas and Ephrimides [5]. A backpressure controller was also devised by Varaiya [6] and Wongpiromsarn [7]. These algorithms place an extra condition of knowledge of routing rates and measure of number of vehicles queuing at every node of the network for every possible routing decision, which in practice is difficult to obtain. The algorithm presented by Gregoire [8] discards this assumption and only requires information about vehicles queued at a junction for every possible turn to an upstream node; this can easily be obtained by camera sensors within proximity to the junction.

All these algorithms focus almost exclusively on just minimizing queue length, but in the process end up stalling vehicles belonging to minimal queue length for enormous amounts of time. This algorithm aims to bridge the gap by minimizing both average queue length and waiting-time in the network by introduce a waiting-time component to the backpressure algorithm.

This paper is organized into five sections. Section 2 models the network, section 3 presents the algorithm, section 4 talks about the simulation model, section 5 consists of results of the algorithm and section 6 contains conclusions and future work.

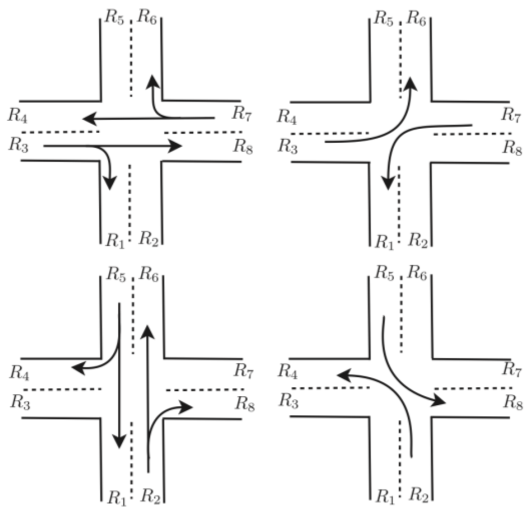
2. Model:

The traffic network *N* can be modelled as directed graph having N links and L junctions. Then the network can succinctly be written as *N* = (*L,J*) where $L = \{L_1, \ldots, L_N\}$ and $J = \{J_1, \ldots, J_L\}$. For each junction J_i there exists a controller which chooses the optimal phase $p_i \in P_i$ the set of all possible phases for that junction at that time step. The traffic signal system operates in slotted time $t \in \mathbb{N}^0$.

Only the vehicles present at the node at the beginning of time slot t can travel from one node to another in time slot t. Every controller maintains queues for each link, and this is updated every time step. For each $a \in \{1, \ldots, N\}$ we let $Q_a(t) \in \mathbb{N}^0$ denote the number of vehicles on L_a. At the beginning of each time slot, the traffic signal controller determines the phase for each junction to be activated during this time slot. The algorithm presented in this paper tries to solve the following problem:

Design a traffic signal controller that determines the phase $p_i \in P_i$ for each junction J_i to be activated during each time slot $t \in \mathbb{N}^0$ such that the network throughput is maximized.

The controller may choose from one of the four below phases:



3. Algorithm:

The backpressure algorithm used in traffic controllers already present in literature [9] is given below:

```
Algorithm 2 BP control
Require:
    Queues lengths $Q_a(t)$,
    Pressure functions $P_a(Q_a)$,
    Loop detectors variables $d_{ab}(t)$.
    function BP
        for $i \in \mathcal{J}$ do
            for $a \in \mathcal{I}(J_i) \cup \mathcal{O}(J_i)$ do
                $\Pi_a(t) \leftarrow P_a[Q_a(t)]$
            end for
            for $a \in \mathcal{I}(J_i), b \in \mathcal{O}(J_i)$ do
                $W_{ab}(t) \leftarrow d_{ab}(t) \max(\Pi_a(t) - \Pi_b(t), 0)$
            end for
            $p_i^\star(t) \leftarrow \arg\max_{p_i \in \mathcal{P}_i} \sum_{a \in \mathcal{I}(J_i), b \in \mathcal{O}(J_i)} W_{ab}(t)\mu_{ab}(p_i)$
        end for
        return Phase $p^\star(t)$ to apply in time slot $t$
    end function
```

The already existing algorithm comes with its own set of advantages. The algorithm has been shown to be throughput optimal for arrival rates up to 0.8 /sec. The algorithm grows in complexity with $O(N)$ where *N* is the number of links/queues in the network, which fares much better than other reinforcement learning approaches to the traffic control problem; reinforcement learning techniques like q-learning usually suffer from dimensionality issues. This enables the algorithm to be implemented in a distributed manner with only next-hop junction information being required. This algorithm performs exceptionally well for large volumes of traffic, especially in cases where arrival rates are at the boundary of the capacity region.

Efficiency of the backpressure algorithm begins to drop when compared to other algorithms in terms of overall waiting time. Problems arise for low volumes of traffic or when vehicles are not equally distributed between the links entering the junction. The algorithm will ensure that queue lengths are kept at a minimum and vehicles in larger queues are almost always given phase assignment priority, but vehicles part of smaller queues end up waiting for arbitrarily large amounts of time.

To account for this, a quantity called "thresh" is measured at each junction for each link (simultaneously with queue length). As soon as the "time-since-last-green" or waiting time for a particular lane exceeds the "thresh" value, the controller immediately activates the corresponding lane completely ignoring the optimal phase decided by backpressure. For low-volume traffic, it makes more sense to have a lower "thresh" value, while for high volume traffic, a greater "thresh" value is required; priority should be given to managing queue lengths in this case. The choice between following waiting-times or backpressure measurements is a tradeoff. Sometimes a lower "thresh" value may come at the cost of allowing larger queue lengths.

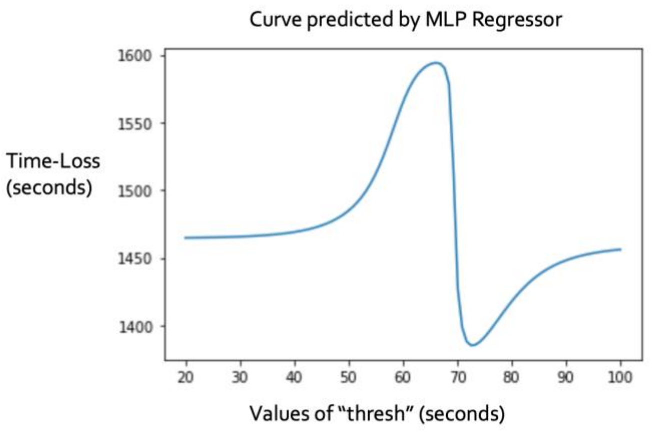
Linear regression is performed to achieve a "thresh" value which minimizes the overall time-loss. Taking time-loss as the performance metric by which efficiency of the network is evaluated, "thresh" values are varied over a time interval. Linear regression using a MLP regressor is performed on the data set with "thresh" values as inputs and time-loss as the desired-output. An optimal "thresh" value which minimizes the time-loss in the networks is then chosen and used for further performance evaluations like stability.

Regressor	MLP Regressor
Hidden Layers	3
Hidden Neurons per Layer	25
Activation Function	Logistic/sigmoid
Solver	Lbfgs (limited-memory Broyden-Flethcher-Goldfarb-Shanno)
Iterations	5000

The modified backpressure control algorithm (accounting for waiting time) is given below:

```
Algorithm Modified BP Control
Require
    Queues lengths $Q_a(t)$
    Pressure functions $P_a(Q_a(t))$
    Time since last green $T_a(t)$
    Lane to phase functions $L(T_a(t))$
Function Modified BP
    For $i \in J$ do
        For $a \in I(J_i) \cup O(J_i)$ do
            $\Pi_a(t) \leftarrow P_a(Q_a(t))$
        End for
        For $a \in I(J_i), b \in O(J_i)$ do
            $W_{ab}(t) \leftarrow d_{ab}(t) \max(\Pi_a(t) - \Pi_b(t), 0)$
End for
        If $T_a(t) > thresh$ do
            $p_i^*(t) \leftarrow L(T_a(t))$
        Else do
            $p_i^*(t) \leftarrow argmax_{p_i \in P_i} \sum_{a \in I(J_i), b \in O(J_i)} W_{ab}(t)\mu_{ab}(t)$
    End for
```

An example graph generated by the MLP regressor is shown below. The minima can be easily obtained analytically.



4. Simulation:

To test the performance of the modified backpressure algorithm, simulations were run on SUMO, a GUI used to model real-world traffic networks. A four-junction network with each road having three lanes was modeled. Standard values of max-speed, acceleration, and deceleration (part of standard definition of vehicle class in SUMO) were used. Vehicles were assigned random trips and started and ended at the fringes of the network.

The simulation was run for 1000-time steps with "thresh" values changed after each simulation. Once the optimal "thresh" value was obtained from the neural network, it was fed to the junction controller and the simulation was run once again, this time to obtain measures such as time-loss, total queue length and waiting time.

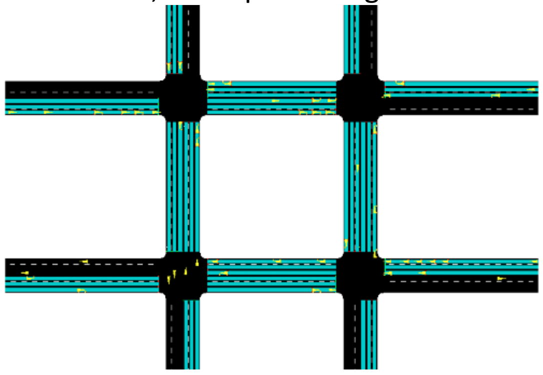


Figure 1 Image of 4 junction simulator

Lane-area-detectors were used on each lane to detect the number of vehicles queued. This queue length was later used in the calculation of back-pressure. The modified backpressure algorithm ran iteratively after every 10-time steps, equal to the duration of each phase, to allow enough vehicles to pass through the junction.

5. Results:

From the graphs obtained via the simulator, it was inferred that the network remained stable for arrival rates up to 0.9 per sec and beyond which total queue length became unbounded.

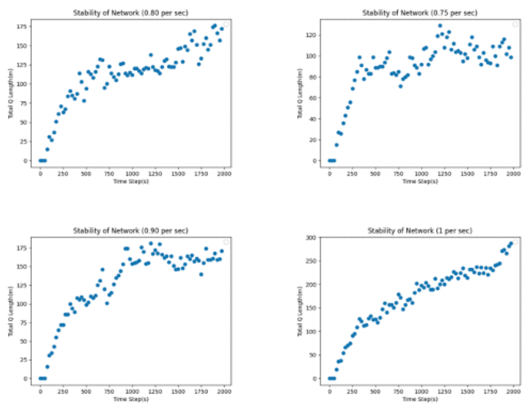


Figure 2 Stability of Modified Algorithm for various arrival rates

This result is indicative of the validity of the algorithm since other algorithms also claim a similar maximum arrival rate to ensure stability (see graphs below)

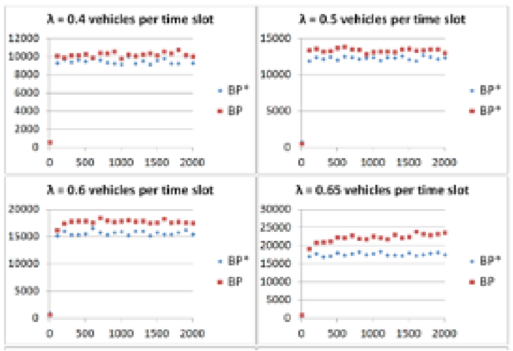


Figure 3 Stability of other algorithms for various arrival rates

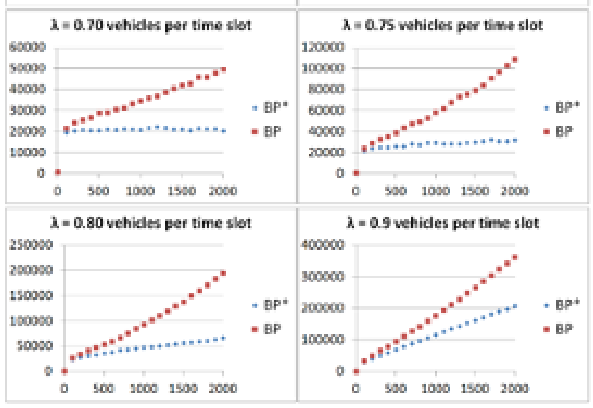


Figure 4 Stability of other algorithms for various arrival rates

Comparisons in terms of jam length and average queue length was made between the modified backpressure algorithm and a static traffic controller.

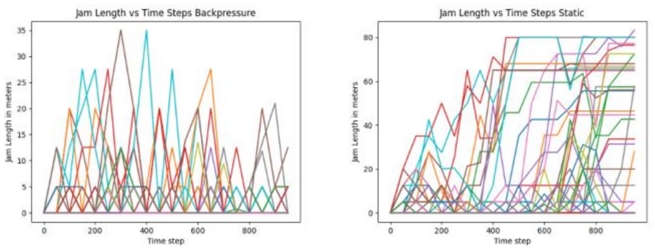


Figure 5 Jam Length for modified backpressure (left) and static controller (right)

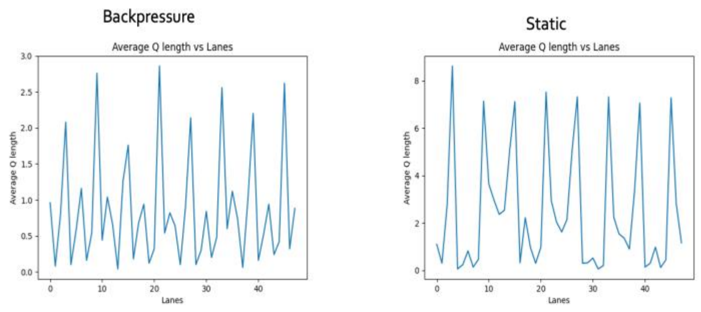


Figure 6 Average queue length for modified backpressure (left) and static controller (right)

The modified backpressure algorithm outperformed the static controller in every traffic measure with reduction up to 220% in jam length, reduction up to 300% in average queue length, albeit in a smaller network.

Section 6: Conclusions and Future Work:

The results of the simulator provide conclusive evidence that the algorithm performs well from a time-loss minimizing perspective and at the same time maintains the same level of stability that other algorithms have. The distributed nature of the algorithm and the linear complexity makes it easy to implement at a larger scale with no significant downside. In the future simulations can be run on larger networks and the performance and scalability can be evaluated. Boundary cases involving extremely high volume of vehicles on certain lanes at a junction and minimal volumes at the other is also unexplored. Finally, since the algorithm is time invariant, any major fluctuation in the flow dynamics of the network requires that the neural network be retrained to obtain the new optimal "thresh" value. This delay can be avoided by introducing unsupervised learning, specifically reinforcement learning so that "thresh" values can be obtained dynamically.

Reference:

[1] THE SCOOT ON-LINE TRAFFIC SIGNAL OPTIMISATION TECHNIQUE by Hunt et al.

[2] P. R. Lowrie, "SCATS, Sydney Co-Ordinated Adaptive Traffic System: A Traffic Responsive Method of Controlling Urban Traffic,"

[3] The PRODYN real time traffic algorithm by Henry et al.

[4] The RHODES prototype: a description and some results by Head and Mirchindani

[5] Tassiulas, L. and Ephremides, A. (1992). Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multi-hop radio networks

[6] Varaiya, P. (2013). The max-pressure controller for arbitrary networks of signalized intersections

[7] Wongpiromsarn, T., Uthaicharoenpong, T., Wang, Y.,Frazzoli, E., and Wang, D. (2012). Distributed traffic signal control for maximum network throughput

[8] Gregoire, J., Frazzoli, E., de La Fortelle, A., and Wongpiromsarn, T. (2013a). Capacity-aware back-pressure traffic signal control

[9] Back-pressure traffic signal control with unknown routing rates by Jean Gregoire, Emilio Frazzoli, Arnaud de La Fortelle, Tichakorn Wongpiromsarn

Appendix:

https://github.com/mihirdshirur/Project-Code