Chapter 6

Smart Traffic: Intelligent Traffic Light Control

Joseph Field[1], Rogier Brussee[2], Evert-Jan Bakker[3], Jeremy Budd[4], Rémi de Verclos[5], Steven Fleuren[6], Stephanie Gonzalez Riedel[6], Emma Keizer[3], Hans Stigter[3], Guus ten Broeke[3], Mark van den Bergh[7]

[1]Oxford University, UK
[2]University of Applied Sciences Utrecht, The Netherlands
[3]Wageningen University, The Netherlands
[4]University of Nottingham, UK
[5]Radboud University, The Netherlands
[6]Utrecht University, The Netherlands
[7]Leiden University, The Netherlands

Abstract:
The Study group participants for the Intelligent Traffic Light Control assignment were tasked with investigating the use of autonomous traffic light controls for a network of road intersections such as the road map of a city. Smart Traffic, the software used and designed by our client Sweco, is designed to control a single intersection. Our task involved optimising the traffic flow without the need for global (e.g. city wide) governance of all traffic lights. This would both fit with the existing approach in Smart Traffic, avoids bad experiences in the past, and mathematically avoids a huge explosion of the state space. Unfortunately, the Smart Traffic software was not available for use and study, and this somewhat limited the approaches we considered. Instead, the software was treated as a black-box, for which we only knew an idealised version of the cost function that the software tried to optimise. We therefore focused on finding a method to have multiple instances of this control coupled to each other in a loose and comparatively simple way. The methodology of Smart Traffic meant that each intersection would act in a 'greedy' fashion, working solely with local information and without any regard for neighbouring intersections. We were advised to devise simple ways in which the software could be adapted, such that each intersection would use only limited information about its neighbours. In particular we should avoid direct communication of optimisation strategies between neighbouring intersections. Indeed, this would be akin to using a global optimisation scheme for the entire network. The approach that we settled upon, was that each intersection implements a (time-dependent) cost function like for the existing Smart Traffic software, but modified to take into account the expected wait-time of downstream traffic. This balances local wait-times with future expected wait-times at neighbouring intersections as communicated by their nearest neighbours. It should result in intersections being less eager to release traffic in a direction where cars would, in the near future, be waiting a long time, and would be sending cars at a reduced rate in the direction of an already congested intersection. At the same time, each instance remains in control of the traffic at a single intersection.

KEYWORDS: *Smart Traffic, Optimisation, Networks*

6.1 Introduction

6.1.1 Company Background

Sweco is a European engineering consultancy company. One of the products that it is further developing is *Smart Traffic*: a real time traffic data driven software that can be used for the efficient control of traffic lights. Using this information, Smart Traffic is able to create detailed predictions for upcoming traffic conditions, and is able to control traffic lights in order to optimise future traffic flow. The main sources of the traffic data used by Smart Traffic are loop detectors, which are embedded in most roads of urban areas, and data such as Floating Car Data, which details the location of moving vehicles in a road network using GPS and cell based location-data from mobile phones carried by the drivers.

A particular feature of Smart Traffic is that it balances the movement of individual road users with the overall mean traffic flow, and avoids long waiting times of individual cars, i.e. it weights individual car "latency" with intersection "throughput", such that mean square vehicle wait-times at four-way intersections can be reduced by up to 40%.

6.1.2 Problem Description

The implementation of Smart Traffic for a single intersection was introduced by Sweco with great success. At the scale of a single intersection, traffic predictions can be made such that total wait-time of cars can be greatly reduced. It is driven by minimising a cost function that is essentially the sum of squares of the waiting times of all cars *at the intersection* (see Section 6.2.2). In particular, the optimisation problem being solved is purely local to the intersection. Unfortunately, local optimisations may result in catastrophic consequences for traffic flow on the whole road network. In fact, the actual deployed software needs to take ad hoc measures to avoid such (rare) situations. Indeed, the group was told that a 2×2 grid of four-way intersections, each controlled by Smart Traffic, would generally tend to a state of gridlock. This was shown by van Hout in a master student's thesis at Eindhoven University of Technology that Sweco worked with prior to SWI 2019 [3]. He investigated the general utility of Smart traffic, as well as the use of Smart Traffic to control coupled intersections. His thesis was the primary source for the group in understanding how traffic models were used in Smart Traffic, and what type of cost function was optimised to provide efficient traffic flow.

Our problem was to find a method to couple multiple instances of the existing Smart Traffic software, such that each intersection was optimised using *full* local information but only using *limited* neighbouring information, without compromising the global behaviour of the network.

6.2 Mathematical Model

We first review the work presented in [3], outlining the main simplifying assumptions before discussing the model itself.

6.2.1 Assumptions

The following simplifying assumptions are made for the single-intersection model. They will be made for the multi-intersection model as well, unless specifically indicated, in order to focus on the problem of coupling different intersections in a simpler idealised context.

Assumption 1: *Traffic is comprised of a single vehicle type.*

We assume that all vehicles are behaving indistinguishably (or are statistically governed by a single simple law such as Poissonian behaviour), i.e., we assume that all vehicles are of the same size, travel at the same speed, and have the same reaction times. For simplicity and ease of reading we will often call a vehicle a car.

Assumption 2: *Vehicles either travel at their desired speed, or are stationary.*

Rather than model car acceleration, the time needed for cars to reach their desired speed is instead attributed to a delay time, τ. Thus, velocities are treated as step functions.

Assumption 3: *Driver reaction time is zero.*

Similar to the effect of acceleration, any effects due to the reaction of the car driver are accounted for by the delay time, τ.

Assumption 4: *Traffic outflow occurs at a maximum flow rate μ.*

Cars can only leave an intersection, once the corresponding light is green, and the delay time, τ, has passed. After this time, cars leave each open lane of an intersection at a constant rate of μ cars per unit time.

Assumption 5: *Traffic lights are either green or red, we ignore yellow.*

When faced with a yellow light, drivers will either continue to drive through the intersection (effectively extending the green light interval), or will stop early (effectively extending the red light interval). Thus yellow could be treated as a (time dependent) probabilistic mix of red and green but for simplicity we simply ignore it.

Assumption 6: *Vehicles arriving at a red light 'stack' on already-present vehicles.*

This assumption allows us to forgo the need to keep track of specific positions of cars in the model. Instead, any vehicles that are waiting together are treated as a 'platoon', that will travel together to the next intersection. In larger networks this needs some modification as cars divide themselves over the network.

Most of these assumptions are used to reduce the amount of book-keeping that would be needed for a more involved model, i.e. a model that allowed for different vehicle characteristics, or specific modelling of delay effects. However, from the results presented in [3], the reduced model captures much of the behaviour of a complex one.

6.2.2 Model

The traffic model that we discuss is found in [3], with notation being taken from [5]. To fix notation wedefine

λ_p	Mean arrival rate for cars approaching lane i
$n_p(t)$	Number of stationary cars in lane i at time t
ℓ_p	Distance to lane i from the previous intersection/boundary
v	Desired vehicle speed (assumed equal for all vehicles)
$w_p(t^*, \Delta t)$	Total waiting time for vehicles in lane i during the interval $[t^*, t^* + \Delta t]$
$Q_p^{\text{arr}}(t)$	Number of cars arriving at lane i
$Q_p^{\text{dep}}(t)$	Number of cars departing from lane i

Let us consider the arrival times of cars on lane i, assuming that the cars are entering the network from the boundary. They are determined by function $Q_p(t)$, that can be modelled by a Poisson process with rate λ_p. We assume that we are able to observe incoming traffic well ahead of its arrival, by sensors that are placed at a distance ℓ_p ahead of the intersection. With the assumption that vehicles travel at a constant speed v, it is clear that the approaching vehicle will reach the intersection $T_p := \ell_p/v$ seconds after it is observed. Thus, we define the intersection arrival function

$$Q_p^{\text{arr}}(t) := Q_p(t - T_p). \tag{6.1}$$

We note that in [3] it is assumed that traffic light schedules are fixed at least 30 seconds ahead of time. This means that for each intersection $T_p \geq 30$, otherwise it would not be possible to incorporate vehicle arrival information into future schedules. The total number of vehicles to have arrived at lane i by time t is given by

$$N_p^{\text{arr}}(t) = \int_{-\infty}^{t} Q_p^{\text{arr}}(s)\mathrm{d}s. \tag{6.2}$$

Similarly, the total number of vehicles to have departed lane i by time t is

$$N_p^{\text{dep}}(t) = \int_{-\infty}^{t} Q_p^{\text{dep}}(s)\mathrm{d}s, \tag{6.3}$$

where $Q_p^{\text{dep}}(t)$ is determined by the controls of Smart Traffic. The number of stationary vehicles in lane i at time t is therefore $n_p(t) = N_p^{\text{arr}}(t) - N_p^{\text{dep}}(t)$ (see Fig. 6.1).

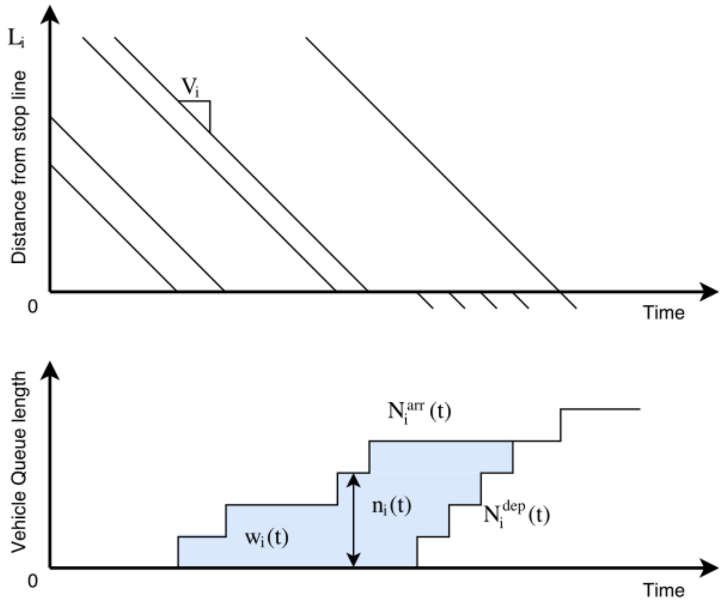

Figure 6.1: (Top) Process of vehicles arriving at an intersection. (Bottom) Cumulative wait-time for vehicles in an intersection. [Image taken from [3]].

6.2.2.1 Cost function

Smart Traffic is built on the concept of minimising the wait-time of traffic until the last time the lights changed. The total wait-time for traffic in lane p over the time interval $[t^*, t^* + \Delta t]$ is given by

$$\langle w_p \rangle(t^*, \Delta t) = \int_{t^*}^{t^*+\Delta t} n_p(t)\mathrm{d}t. \tag{6.4}$$

However, Smart Traffic insists on not only minimising waiting times for the whole intersection but for individual vehicles, such that we want to minimise the *combined* cost of the wait-time for all vehicles over *all* lanes of an intersection. More specifically, the goal is to avoid having one lane waiting for a long time even if it would decrease the total waiting time for the whole intersection. This can be implemented by minimising a *strictly* convex cost function function that depends on the waiting times of individual cars, or less accurate but with reduced book keeping, on the individual lanes, that penalises excessively long waiting times of a car or of a lane[8]. One can argue that the precise cost function could take into account experimental psychology measuring the stress of drivers, but the simplest approach is to minimise the square of the waiting times of cars in the lanes. This still ensures that, as an extreme example, if one lane has a high traffic throughput, while a crossing lane has only one car waiting, eventually the *squared* cost of the wait-time of the single-car will outweigh the combined effect of multiple cars in the other lane. Hence the algorithm will eventually allow the cars to travel through the intersection.

We consider a fixed lane p_0 as a FIFO (First In, First Out) queue, and compute the squared waiting time in terms of the occupation number $n(t)$ (here as below we will temporarily suppress the lane p_0 from the notation). Assume we start at $t = t_0$ with an empty lane and that cars arrive at $t_0 < t_1^{(\mathrm{a})} < t_2^{(\mathrm{a})} < \ldots < t_{N^{(\mathrm{a})}}^{(\mathrm{a})}$ and leave at $t_1^{(\mathrm{d})} < t_2^{(\mathrm{d})}, \ldots t_{N^{(\mathrm{d})}}^{(\mathrm{d})} < t^*$. Car number i can not leave before it arrives so $t_i^{(\mathrm{a})} \leq t_i^{(\mathrm{d})}$ (i.e. a lane is a FIFO queue). Moreover we cannot have more cars leaving then arriving so $N^{(\mathrm{a})} \geq N^{(\mathrm{d})}$. The squared waiting time from $t = t_0$ when there were no cars waiting, up to time t^*, is given by

$$\begin{aligned} \langle w_i^2 \rangle (t_0, t^*) &= \sum_{i=1}^{N^{(\mathrm{d})}} (t_i^{(\mathrm{d})} - t_i^{(\mathrm{a})})^2 + \sum_{j=N^{(\mathrm{d})}+1}^{N^{(\mathrm{a})}} (t^* - t_j^{(\mathrm{a})})^2 && (6.5) \\ &= \langle w^2 \rangle (t_0, t_{N^{(\mathrm{d})}}^{(\mathrm{a})}) + \int_{t_{N^{(\mathrm{d})}}^{(\mathrm{a})}+\epsilon}^{t^*} (t^* - t)^2 \mathrm{d}n(t) && (6.6) \\ &= \langle w^2 \rangle (t_0, t_{N^{(\mathrm{d})}}^{(\mathrm{a})}) + \int_{t=t_{N^{(\mathrm{d})}}^{(\mathrm{a})}+\epsilon}^{t^*} (t^* - t)^2 \mathrm{d}(n(t) - n(t_{N^{(\mathrm{d})}}^{(\mathrm{a})} + \epsilon)), && (6.7) \end{aligned}$$

where $\epsilon > 0$ is so small, that $t_{N^{(\mathrm{d})}}^{(\mathrm{d})} + \epsilon < t_{N^{(\mathrm{d})}+1}^{(\mathrm{a})} < t^*$ (assuming such an arrival time $< t^*$ after the last departure time $< t^*$ exists: the integral is zero for any ϵ such that $t_{N^{(\mathrm{d})}}^{(\mathrm{d})} + \epsilon < t^*$ if $N^{(\mathrm{a})} = N^{(\mathrm{d})}$ as it should) Technically, the integral is a Stieltjes integral, but no harm is done pretending it is the limit of smooth approximations of the step function $n(t)$.

We can now do partial integration with the boundary terms vanishing because of the

[8]In computer science this is called balancing throughput and latency. It ensures e.g. that a video stream which requires that IP packets get through with low latency does not stop when someone using the same WIFI access point downloads a long file for which it is optimal that as many IP packets get through at once as possible.

rewrite in the last line. This gives

$$\langle w^2\rangle(t_0,t^*) - \langle w^2\rangle(t_0,t^{(\mathrm{a})}_{N^{(\mathrm{d})}}) = -\int_{t^{(\mathrm{a})}_{N^{(\mathrm{d})}}+\epsilon}^{t^*} (n(t) - n(t^{(\mathrm{a})}_{N^{(d}} + \epsilon)\mathrm{d}(t^*-t)^2 \tag{6.8}$$

$$= -\int_{t^{(\mathrm{a})}_{N^{(\mathrm{d})}}}^{t^*} (n(t) - n(t^{(\mathrm{a})}_{N^{(\mathrm{d})}}+)\mathrm{d}(t^*-t)^2 \tag{6.9}$$

$$= 2\int_{t^{(\mathrm{a})}_{N^{(\mathrm{d})}}}^{t^*} (t^*-t)(n(t) - n(t^{(\mathrm{a})}_{N^{(\mathrm{d})}}+))\mathrm{d}t, \tag{6.10}$$

where $n(t^{(\mathrm{a})}_{N^{(\mathrm{d})}}+)$ is a shorthand for $n(t^{(\mathrm{a})}_{N^{(\mathrm{d})}}+\epsilon)$ with ϵ as above (i.e. the number of cars in the lane right after the last car that left before t^*). Thus, the additional square waiting time absorbed in the interval $[t^*, t^*+\Delta t)$ is

$$\langle w^2\rangle(t^*+\Delta t, t_0) - w^2\rangle(t^*,t_0) = 2\int_{t^*}^{t^*+\Delta t} (t^*-t)n(t)\mathrm{d}t \tag{6.11}$$

$$+\,2\int_{t_{N^{(\mathrm{d})}}}^{t^*+\Delta t} (\Delta t)(n(t) - n(t^{(\mathrm{a})}_{N^{(\mathrm{d})}})\mathrm{d}t. \tag{6.12}$$

Under the integral in the first term, $-\Delta t < (t-t^*) < 0$, so is negligible (and negative) for $\Delta t \ll (t^* - t^{(\mathrm{a})}_{N^{(\mathrm{d})}})$. Hence

$$\langle w^2\rangle(t^*+\Delta t, t_0) - \langle w^2\rangle(t^*,t_0) = 2(t^* - t_{)}(n(t^*) - n(t_{N^{(\mathrm{d})}})\Delta t + O((\Delta t)^2)$$

and the per unit time increase of square time is

$$\frac{\mathrm{d}}{\mathrm{d}t^*}\langle w^2\rangle(t^*,t_0) = 2(t^* - t_{N^{(\mathrm{d})}})\left(n(t^*) - n(t_{N^{(\mathrm{d})}})\right).$$

In [3], the optimisation performed by Smart Traffic is not explicitly given. We therefore take as the cost function of lane p at time t^*, the additional squared waiting time in the linear approximation in Δt derived above[9] , up to the irrelevant factor 2 (we keep the factor Δt for clarity) .

$$C_p(t^*, t^*+\Delta t) = \left(t^* - t^{(\mathrm{a})}_{p,N_p^{(\mathrm{d})}}\right)\left(n(t^*) - n(t^{(\mathrm{a})}_{p,N_p^{(\mathrm{d})}})\right)\Delta t. \tag{6.13}$$

The total cost for a particular light setting of all lanes of an intersection X with lanes $p \xrightarrow{\frownie} X$ that get red is then

$$C^X_{\frownie}(t^*, t^*+\Delta t) := \sum_{p\xrightarrow{\frownie}X} C_p(t^*) = \sum_{p\xrightarrow{\frownie}X}\left(t^* - t^{(\mathrm{a})}_{p,N_p^{(\mathrm{d})}}\right)\left(n(t^*) - n(t^{(\mathrm{a})}_{p,N_p^{(\mathrm{d})}})\right)\Delta t. \tag{6.14}$$

[9]This is equivalent to considering a cost per unit time

For a traffic light schedule to be updated at time t, Smart Traffic will compute the configuration with minimal cost C for the interval $[t^*, t^* + \Delta t]$, where $t^* = t + 30$, and $\Delta t \ll 30$. Such that Smart Traffic appends an interval of Δt onto the existing schedule, and updates it predictions on the number of cars arriving and leaving in the interval $[t, t*]$ based on the new information available at the sensor loops. Under our simplifying assumptions, however, cars behave predictable given a choice of traffic light settings.

6.2.3 Model Extension

To extend the utility of Smart Traffic to a network, we take the situation at other intersections into account. This necessarily implies some communication between neighbouring intersections. It was discussed with Sweco what type of information could be shared between intersections. They strongly advised against a global control mechanism for all lights in the city network, i.e. that all intersections are controlled by a single big server that takes into account the situation in the whole city, or alternatively, that each smaller server for an intersection has to take into account the algorithmic "decisions" of the smaller servers for all the other intersections. It was decided, however, that intersections, could, share information regarding the expected wait-times for cars that it will receive, to upstream neighbours. The simplest kind of information passed to a neighbouring intersection would be the running average of observed wait-times for a car approaching from the intersection's direction, or only slightly more difficult some Karman filter type extrapolation and noise filtering.

For example, if an intersection has only one car waiting then uncoupled Smart Traffic would generally let it leave immediately. However, if the downstream intersection is able to communicate that wait-times for traffic will be high by the time the car arrives, it may be more beneficial for the first intersection to keep the car waiting for a little longer until downstream traffic is alleviated.

Note that there is a subtlety here: when a car leaves an intersection it is unknown which lanes a car will choose at the next intersection, i.e. in which direction a car will continue its trip. Moreover, the street network of the city may allow cars to turn up at different intersections. One can, however, estimate the probability for a car departing at time t_1 at intersection X on lane p, to arrive at intersection Y at lane q on time t_2 from previous behaviour[10].

Thus, if, at time t^*, each intersection X has expected wait-times for downstream traffic, we want to balance the cost between keeping cars waiting *now* against them perhaps waiting longer further downstream. To implement this we want define a new

[10]Aggregated gps data from car drivers would be particularly useful here

cost function C^X of a particular lights setting of the intersection X that also takes into account some future waiting time cost for the lanes $p \overset{\smiley}{\longrightarrow} X$ that get a green.

$$C^X(t^*, t^* + \Delta t) = C^X_{\frownie}(t^*, \Delta t) + C^X_{\smiley}(t^*, t^* + \Delta t). \tag{6.15}$$

The cost function for the green lanes should take into account the chance to arrive at a lane of a different intersection, the additional squared waiting times at that downstream lane, some discounting for events the farther away they are in the future, and for good measure some fudge factors that allows for tuning the importance of waiting at different intersections allows for a bit of tuning. This gives

$$C^X_{\smiley}(t^*, t^* + \Delta t) = (\Delta t) \sum_{p \overset{\smiley}{\longrightarrow} X} n^X_p(t^*) \sum_{Y \neq X} a^Y_{X,p} \int_{t^*}^{\infty} \mathrm{d}t D(t, t^*) \sum_{q \to Y} p(Y, q; t|X, p; t^*) W^Y_q(t), \tag{6.16}$$

where the sum is over all lanes of the intersection X and each intersection $Y \neq X$ in the network, each given a weight a^X_Y, over all lanes q of that intersection. The integral $\int_{t_0}^{t_1} p(Y, q; t|X, p; t^*)\mathrm{d}t$ is the chance that a car leaving from lane p of intersection X at time t^* will arrive at lane q of intersection Y, between t_0 and t_1, and $W^Y_q(t)$ is the estimated waiting time for lane q. This sum is very general but a little unwieldy, so we can make some simplifying assumptions.

- We assume that all lanes of an intersection have equal weight as far as costs are concerned and is non zero only if a lane directs to a neighbouring intersection Y[11] i.e.
$$a^Y_{X,p} = \begin{cases} a & \text{if } p \to X \to Y \\ 0 & \text{otherwise} \end{cases} \tag{6.17}$$

- We assume that the discount function has a finite time window and/or a fairly steep descent with respect to planning period (30 seconds) so waiting time estimates are not too critical far in the future, e.g. $D(t, t*) = \exp((t - t^*)/\theta)$ with θ 1min.

- the probability to arrive at a lane only depends on the time difference $t - t^*$ and the distance $d(X, Y)$ between two (directly neighbouring) intersections provided lane p on X directs to lane q on Y. More over each lane q of an intersection Y reachable from lane p (through X) is equally probable. Hence
$$p(Y, q; t|X, p; t^*)\mathrm{d}t = \begin{cases} \frac{1}{(\#p \to q \to Y)} p_{d(X,Y)}(t - t^*)\mathrm{d}t & \text{if } p \to X \to q \to Y, \\ 0, \end{cases} \tag{6.18}$$

[11]These time independent parameters may be good candidates for using machine learning for the most satisfying results

where $p_{d(X,Y)}(t-t^*)\mathrm{d}t$ is some 1 parameter family of probability distributions over time. In fact simplifying even further we can make the assumption of cars moving in platoon at speed v after a delay τ and use the degenerate delta function probability distribution

$$p_{d(X,Y)}(t-t^*)\mathrm{d}t = \delta(t-t^*-d(X,Y)/v-\tau)\mathrm{d}t. \tag{6.19}$$

With these assumptions the cost function simplifies to

$$\begin{aligned} &C^X_{\smiley}(t^*, t^*+\Delta t) \\ &\quad = a\Delta t \sum_{p\overset{\smiley}{\longrightarrow}X} n^X_p(t^*) \sum_{p\to q\to Y} \int_{t^*}^{\infty} \mathrm{d}t e^{-(t-t^*)/\theta} W^Y_q(t) \frac{p_{d(X,Y)}(t-t^*)}{(\#p\to q\to Y)}, \end{aligned} \tag{6.20}$$

or simplifying even further using the degenerate distribution over time

$$C^X_{\smiley}(t^*, t^*+\Delta t) = a\Delta t \sum_{p\overset{\smiley}{\longrightarrow}X} n^X_p(t^*) \sum_{p\to q\to Y} e^{-(d(X,Y)/v+\tau)/\theta} \frac{W^Y_q(t^*+d(X,Y)/v+\tau)}{(\#p\to q\to Y)} \tag{6.21}$$

Finally to make a decision on a plan for traffic lights for the next T = 30 seconds and take into account account the rules for changing the lights. Toy rules would be

- lights stay green for at least 1 second,
- lights stay red for at least 3 seconds,
- an intersection must block an opposite lane for at least 10 seconds before a crossng lane can get green

Under these rules we have to minimise total squared waiting time[12]

$$C^X(t,T) = \sum_{n=0}^{T/(\Delta t)-1} C(t+n\Delta t, t+(n+1)\Delta t). \tag{6.22}$$

Where $\Delta t = 0.1$ second should be amply good enough resolution.

6.2.3.1 Matlab **Implementation**

For a four-way intersection, with each road having three lanes (see Fig. 6.2) there are many light configurations that conflict with one another; for example, in the figure shown Lanes 2 and 7 cannot both be relieved at the same time.

[12]This is essentially a poor numerical integration and using Simpson's or the trapezoidal rule and subtracting half the boundary values might be appropriate

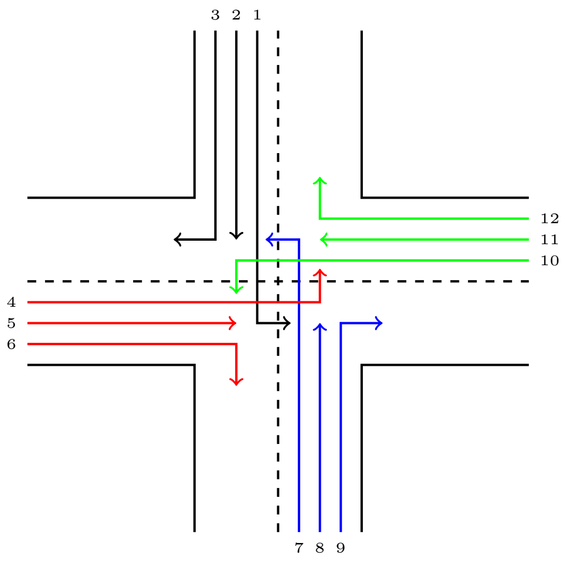

Figure 6.2: Four-way intersection with twelve lanes.

There are roughly 100 light configurations that do not cause conflict, which can be described by a binary vector of length 12, with green lights corresponding to a 1, and red lights corresponding to a 0. For a single intersection, we define the optimisation space to be

$$\mathcal{S} := \{\mathbf{v} \in \mathbb{B}^{12} \ : \ \mathbf{v} \text{ causes no conflicts}\}, \tag{6.23}$$

where $\mathbb{B}$ is shorthand for the Booleans $\mathbb{Z}_2$. Given that each intersection will be governed by separate instances of Smart Traffic, the optimisation space for the entire network will therefore be a product of $\mathcal{S}$, which, for a network of M intersections, we will define as

$$\mathcal{S}^M := \{\mathbf{v} \in \mathbb{B}^{12M} \ : \ \mathbf{v} \text{ causes no conflicts}\}. \tag{6.24}$$

Describing the action space of the coupled network in this way allows us to build a model using simple linear algebra. We define $\mathbf{A}$ to be the adjacency graph of the network, $\mathbf{I}$ to be the identity matrix (the same size as A), and $\mathbf{n}(t)$ to be the vector holding the number vehicles in each lane at time t. Note that the multiplication $\mathbf{An}$ describes the instantaneous travelling of all vehicles to their desired destination, while $\mathbf{In}$ describes the event that all vehicles remain stationary.

For a particular action $\mathbf{v} \in \mathcal{S}^M$, we create the diagonal matrix $\mathbf{V} := \text{diag}(\mathbf{v})$, such that $\mathbf{AV}$ defines an augmented adjacency matrix corresponding *only* to those lanes which allow traffic through. Similarly, the matrix $\widetilde{\mathbf{V}} := \mathbf{V} - \mathbf{I}$ defines all lanes which are not opened. Therefore, defining the matrix $\mathbf{B} = \mathbf{AV} + \widetilde{\mathbf{V}}$, the multiplication $\mathbf{Bn}$ describes the instantaneous state transition for all vehicles in the model.

For implementation purposes, as we have assumed that vehicle data is perfectly accurate, we only need to look at discrete events that occur in the model, i.e. new vehicles entering the model from the boundary, and cars arriving at or leaving an intersection. Given that we assume that all vehicles travel at a fixed speed, and all road lengths are known, we can therefore predict all future events.

By using an event-based model, we therefore need to keep track of vehicles that are stationary at any point, and vehicles that are moving between intersections. This is done in such a way that a car released from an intersection can be described by a Heaviside function for arrival at the downstream intersection. This means that rather than using $\mathbf{A}$ to describe the *instantaneous* travel of a vehicle to its destination, we can instead make it a time-dependent Heaviside matrix $\widetilde{\mathbf{A}}(t)$ which keeps track of when vehicles will be arriving downstream, such that we can compute wait-time effects from cars arriving between successive schedules.

Let us assume that the model has some configuration of stationary and moving vehicles, as well as knowledge of vehicles approaching from the network boundary. Every Δt seconds, Smart Traffic will amend the network schedule by optimising traffic flow over the interval $[t^*, t^* + \Delta t]$. As we have previously described, the total wait-time for traffic at a single lane during this interval is merely the integral given by Eq. 6.4. In vector form, we have

$$\begin{aligned} \mathbf{w}(t^*, \Delta t) :=& \int_{t^*}^{t^*+\Delta t} \mathbf{n}(t)\mathrm{d}t, && (6.25) \\ =& \int_{t^*}^{t^*+\Delta t} \widetilde{\mathbf{V}}\mathbf{n}(t^*) + \widetilde{\mathbf{A}}(t)\mathbf{1}\mathrm{d}t && (6.26) \\ =& \Delta t\widetilde{\mathbf{V}}\mathbf{n}(t^*) + \int_{t^*}^{t^*+\Delta t} \widetilde{\mathbf{A}}(t)\mathbf{1}\mathrm{d}t. && (6.27) \end{aligned}$$

Therefore, for each entry in the vector $\mathbf{w}$, the wait-time is comprised of all cars that are not released at time t^*, plus all cars that arrive before $t^* + \Delta t$. Given that we want to minimise the squared wait-times, this is equivalent to minimising the 2-norm of $\mathbf{w}$.

In the extended model, we also want to balance the cost of sending cars downstream, assuming that we have knowledge of expected future wait-times, which we hold in a diagonal expectation matrix $\mathbf{E}(t^*)$. Therefore, the expected cost of releasing vehicles due to an action $\mathbf{v}$ is merely

$$\mathbf{e}(t^*) := \mathbf{E}(t^*)\mathbf{A}\mathbf{V}\mathbf{n}(t^*). \quad (6.28)$$

Thus, every Δt seconds we must solve the following optimisation problem:

$$\min_{\mathbf{v}\in\mathcal{S}^M} \{\|\mathbf{w}(t^*, \Delta t)\|_2 + \|\mathbf{e}(t^*)\|_1\}. \quad (6.29)$$

Due to the fact that we do not explicitly couple the intersections together, the above problem is equal to n instances of the smaller problem

$$\min_{\mathbf{v}_j \in \mathcal{S}} \{ \|\mathbf{w}_j(t^*, \Delta t)\|_2 + \|\mathbf{e}_j(t^*)\|_1 \}, \tag{6.30}$$

where the subscript $j = 1, \ldots, M$ corresponds to each particular intersection.

The benefit of framing the problem in this way is that we are able to explicitly model *all* events in the coupled model, yet the optimisation scheme is completely separable. Thus, the computational complexity grows linearly with M. While the implementation seems very simple, it was not possible to get a working version of the model in Matlab in only three days. While Matlab is built for matrix operations, the utility of a Heaviside matrix was problematic, and made computation times extremely high for even simple integrals like those in Eq. 6.27. Therefore, we were not able to implement an optimisation scheme that had any chance of terminating in a reasonable time. Given that we need to optimise every Δt seconds, this meant that no useful results could be obtained, as the model that was implemented was not efficient enough to replicate the workings of Smart Traffic.

6.3 Other Approaches

6.3.1 Markov Decision Process

In some interpretation of the Smart Traffic model of Sweco, it seems that the mathematical framework of Markov decision processes fits the problem well. In general, a Markov decision process, or MDP for short, consists of the following. We start with a countable state space S. Time is discrete, indexed, say, by N. In time step t, the process is in some state $s \in S$. In every state, there is a set of actions $A(s)$ to choose from. If some action $a \in A(s)$ is chosen, there is an immediate reward of $r_s^t(a)$ and the process transitions to state j with probability $p_{sj}^t(a)$. The reward may be negative, indicating a cost instead. If the reward and transition probabilities are independent of the time t, the model is called stationary. For a more detailed description of the model, as well as an explanation of some optimization methods in this framework, see the lecture notes on MDPs by Kallenberg [4] or the book on reinforcement learning by Barto and Sutton [6].

For the traffic model, one could choose as time steps the intervals between the times where the calculation of an optimal policy is started, e.g., Δt. The state space, unfortunately, is hard to define. Ideally, one would only have to keep track of the amount of cars waiting at every traffic light, that is, $n_p(t)$ for every lane p. However, cars that are currently travelling between intersections also need to be taken care of. Theoretically, it is possible for every car to keep track of its exact location and speed. However, this would cause the state space to explode in size. Therefore, one could propose to only count the number of cars present in each lane of each intersection, and to count the number of cars travelling between intersections, disregarding their exact whereabouts.

As action set in every state, we can take the set of all different configurations of green lights for the intersection, or a choice between continuing the current green scheme or changing it. The reward for every action is somehow dependent on the expected waiting times of the cars in the system, being a summation of the $w_i(t, \Delta t)$ over the lanes i, given the action chosen. One could choose, for example, the cost function described in Section 6.2.2. Finally, the probability of transitioning from state s to state j given some action a is largely deterministic, apart from cars that might enter the system with some probability at the borders. Moreover, having discarded the exact location of the cars travelling between intersections, their movement may also taken to be stochastic.

As we assume the computations always being done for a finite horizon, e.g., 30 seconds, one might use recursive methods to solve for the optimal policy, such as dynamic programming. Starting at the horizon, we move backwards in time, trying to find in time step $t-1$ an action which optimizes $r_s(a) + \sum_j p_{sj}(a)x_j^t$, where x_j^t is the reward we obtain when moving optimally from time step t up until the horizon. However, the size of the state space may in practice be a complicating factor for computing an optimal policy exactly. One might therefore resort to approximate methods, some of which are also described in the referenced works.

6.3.2 Model Predictive Control

As an alternative to Markov Decision support we mention the framework of model predictive control (MPC) that utilizes Pontryagin's Minimimum Principle as a guide to optimality. Formulations are available for both discrete and continuous time models [2]. Given a traffic model in (continuous-time) state space format, i.e.

$$\frac{dx}{dt} = f(x(t), u(t)), \quad x(t_0) = x_0 \tag{6.31}$$

with $x(t)$ the state vector (e.g. traffic volumes) and $u(t)$ the manipulated variables (e.g. traffic light settings). Since traffic lights are switched on/off, one should include this in the underlying equations and choose for a hybrid type of system description where both continuous and discrete time variables are possible [1]. If the cost is now formulated as in 6.4 for a time horizon Δt, we can formulate the following Hamiltonian

$$\mathcal{H}(x(t), u(t)) = n_p(t) + \lambda^T(t) \cdot f(x(t), u(t)) \tag{6.32}$$

with $n_p(t)$ the number of stationary cars at time t, and $\lambda(t)$ the co-state or *influence* function. Pontryagin's minimum principle states that amongst all choices for $u(t)$, the one that minimizes the Hamiltonian $\mathcal{H}(x,u)$ is the optimal choice. It is only in rare cases that this optimal strategy can be stated in terms of $x(t)$ as a feed-back law. More common is to solve the optimization problem as a two-point-boundary-value problem (TPBV) with the state $x(t)$ satisfying 6.31 at $t = t_0$, and the co-state $\lambda(t)$ satisfying

$$\frac{d\lambda}{dt} = -\frac{\partial \mathcal{H}}{\partial x}, \quad \lambda(t_f) = \phi_x(t_f) \tag{6.33}$$

where ϕ reflects the terminal cost at time $t = t_f$, e.g. the number of cars that are stationary at $t = t_f$. Efficient software is available to solve the TPBV problem. Typically, the first run of the software takes some time but once an optimal strategy has been found it is computationally efficient to *adapt* such a strategy every 30 seconds or so, when new data of the current traffic situation becomes available. Unfortunately, software of the current traffic system applied by SWECO was not available and therefore the optimization problem could not be encoded en solved in Matlab under the time-constraint of several days (the duration of the SWI workshop)..

6.3.3 Predicting behaviour of neighbouring intersections

In order to accurately predict the future state of the network, it is useful to predict the behaviour of neighbouring intersections. This will give us a better idea of how much traffic will be arriving at our intersection, and of how long cars we send to one direction will have to wait down the road. Let S be the state space as in Section 6.3.1, and let A_i denote the set of all possible actions of intersection I_i (here an action is a configuration of traffic lights at intersection I_i, so A_i does not depend on the current state). For simplicity, we restrict ourselves to two intersections, I_1 and I_2, that are connected by a road with no intersection in between I_1 and I_2. For larger networks it is possible to apply the described method on each neighbour separately. We will take the point of view where we want to predict the future actions of I_2 to plan the actions of I_1. For each intersection, Smart Traffic keeps track of and updates a planning P_i of actions that the intersections will perform. This planning will be updated at times $t_0, t_1, \ldots$, where $t_i = t_0 + i\Delta t$. At time t_i, the actions that will happen at times $t_i, \ldots, t_{i+k}$ have already been determined for some fixed k, and the method has to decide which action to take at time t_{i+k+1}. We assume that the traffic light configuration will change at most once during each time step. Different intersections do not have access to each other's planning. This means that at time t_i we would like to predict all the actions the neighbouring intersections perform between t_i and t_{i+k+1}. A general procedure to do this is given in Algorithm 2. One way

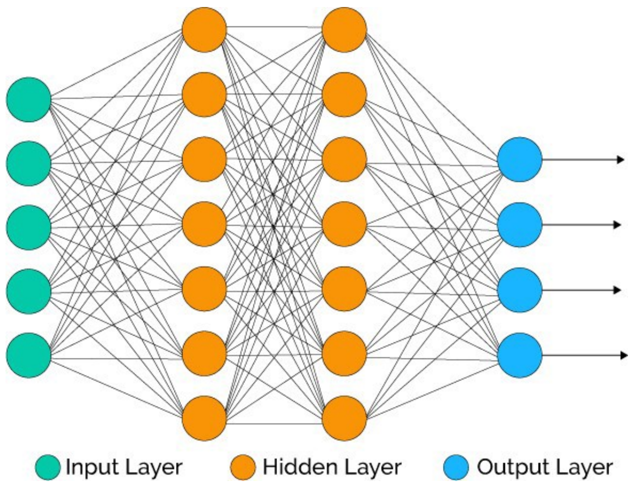

Figure 6.3: A small neural network.

to predict each distribution of actions is by using a neural network, we will discuss this method in the next section.

Algorithm 2:

```
   input  : State $s_{t_i}$
   output: Distributions of actions $\tilde{a}_i, \ldots, \tilde{a}_{i+k+1}$
1  $\tilde{s}_{t_i} \leftarrow s_{t_i}$;
2  for $j \in \{i, i+1, \ldots, i+k+1\}$ do
3      Predict $\tilde{a}_j$ given $\tilde{s}_{t_j}$;
4      Compute expected state $\tilde{s}_{t_{j+1}}$;
5  endfor
```

6.3.3.1 Neural networks

In this section we give the basic ideas behind neural networks, and how we can use them for our problem. We will not go in-depth and often omit explicit formulas, so it will probably be wise to do some more research on the subject before attempting to write an implementation. Luckily, many resources are available, see for instance `https://www.edx.org/course/deep-learning-explained-3`. A neural network consists of a large number of nodes, divided into different layers. Each node in one layer is connected to every node in the neighbouring layers. See Figure 6.3 for an illustration. Each connection has an associated weight. When we give each node in the input layer a value, we can use the weights to compute values for the nodes in the next layer, and repeat this process until the values of the nodes in the output layer are computed. To apply a neural network to our problem, we let the input nodes correspond to the network state s (e.g. have a node for each lane that corresponds to the current waiting time in that node), and each output node corresponds to an action $a \in A_2$. The idea is to set the weights of the connections in such a way that when the input is set according to some state s, the values of likely action will be high and the values of unlikely actions will be low. The process of adjusting the weights to achieve this goal is referred to as training the network. The usual technique for doing this is called back-propagation. This method works as follows: given some state s, we use the current neural network to get a value for each action. We then observe which action a actually takes place. The output value corresponding to a should have been high, and all the others should have been low. So the network can be improved by increasing the weights of connections to the node of a that helped increase the value corresponding the a, and lowering the weights of connections to the other nodes that helped increase their values. Then we can compute what the values of the nodes in the second to last layer should have been, and adjust the weights of the connections between the third to last and the second to last layer based on that. Repeat this process until every connection is updated. Note that this is a data driven approach, since in order to apply backpropagation we need to know which action will be taken given the state we use as input. It will be necessary to collect training data, both before the neural network is implemented to train a initial version, and while the network is active such that it can adapt to changing circumstances (such as changes to the systems that control the neighbours).

6.4 Conclusions and Recommendations

We have suggested several routes to a solution to the complex problem of controlling a network of intersections, given the strategy currently employed by Sweco. Much to our regret we have not been able to thoroughly test our hypotheses and models because of time-constraints and the limited availability of currently used software. Yet, some good suggestions were made on how to tackle this problem from various angles. Most important is to have the current strategy become less greedy in finding a solution for the current situation by including an *expected waiting time* for the downstream intersection into the problem. This could be implemented via, for example, model predictive control in which the goal function is an essential part of the control strategy that is found on the basis of Pontryagin's Minimum Principle. For numerical solution a receding horizon optimal control problem needs to be solved on-line with the expected waiting time of the down-stream traffic included in the goal function. This is certainly within reach since advanced software packages are available (in Matlab) that are capable of solving these kind of problems. Other solutions, such as Markov decision processes and neural networks were also suggested as a means to find an optimal strategy.

Bibliography

[1] Branicky, M.S., Introduction to Hybrid Systems, Chapter in the handbook of networked and embedded control systems, Birkhäuser Boston, 2005

[2] Bryson, A.E., Dynamic Optimization, Addison Wesley, 1999

[3] van Hout, T.C.G., Smart Traffic: Anticipative flexible traffic light control and network stability, Master's thesis, Eindhoven University of Technology, 2017.

[4] Kallenberg, L.C.M., Markov Decision Processes (Manual), Leiden University, 2009

[5] Lämmer, S. and Helbing, D., Self-control of traffic lights and vehicle flows in urban road networks, Journal of Statistical Mechanics: Theory and Experiment, 2008, IOP Publishing

[6] Barto, A.G. and Sutton, R.S., Reinforcement Learning: An Introduction, MIT Press, Cambridge (MA), second edition, 2018