Better Junction Control with Bus Priority

Problem presented by

Bruce Slattery

Halcrow

Executive Summary

The problem was to design a traffic light controller for a set of neighbouring junctions, which gives priority to incoming buses while ensuring a degree of fairness to the general traffic.

The team has developed three complementary approaches, that present different strengths and weaknesses and might be applicable in different junction configurations or traffic conditions:

1. A continuous-variable, discrete-time optimisation approach for determining the fraction of green time to give to each arm of a junction during the next traffic light cycle, in order to minimise total weighted squared vehicle waiting times, with more weight on buses than on cars.
2. A piece-wise linear ordinary differential equation model of queue length dynamics on a junction arm, based on flux of vehicles into and out of that arm.
3. A discrete-variable, discrete-time Markov Decision Process approach. The state of the system is comprised of vehicle queue lengths and the junction's current stage. The action is to stay in the current stage or move to the next stage. An optimal policy minimises long run expected discounted weighted delay.

Version 1.0
May 7, 2013
iii+22 pages

Report author

Helge Aufderheide, Alessandro Colombo, Torran Elson,
John Lees-Miller, Sean Lim, Kat Rock, Matt Saxton,
Piotr Świerczyński

Contributors

Helge Aufderheide (University of Bristol)
Alessandro Colombo (Politecnico di Milano)
Torran Elson (University of Bristol)
John Lees-Miller (University of Bristol)
Sean Lim (University of Oxford)
Kat Rock (University of Warwick)
Matt Saxton (University of Oxford)
Piotr Świerczyński (University of Warsaw)

ESGI91 was jointly organised by
University of Bristol
Knowledge Transfer Network for Industrial Mathematics

and was supported by
Oxford Centre for Collaborative Applied Mathematics
Warwick Complexity Centre

Contents

1 Introduction

(1.1) The original problem was to design a traffic light controller, for a set of neighbouring junctions, which gives priority to incoming buses while ensuring a degree of fairness to the general traffic. This is required if the controller is to work under heavy traffic conditions without generating too much delay for regular cars. Given the complexity of the problem, the team focussed on a single junction.

(1.2) The team has developed three competing approaches, that present different strengths and weaknesses and might be applicable in different configurations or traffic conditions. The degree to which the three proposed solutions can be extended to work efficiently in neighbouring junctions is discussed separately for each approach.

2 Problem statement

(2.1) Bus priority schemes aim to reduce the time that buses spend waiting at traffic lights. This can make buses faster and encourage people to use public transport instead of their cars. The idea is simply that buses tell the traffic control system their positions using the existing real time data feeds, and the system tries to give buses green lights when they approach junctions.

(2.2) Modern traffic control systems measure traffic and try to optimise their actions based on current conditions. However, they were not designed with bus priority schemes in mind. Bus priority is often implemented by adding a crude ‘exception’ rule for buses, which causes the system to give an approaching bus a green light regardless of other traffic. This often leads to poor performance, which leads operators to disable bus priority at peak times, when it would be most useful. A better system would feature full integration of all vehicles into the defined traffic model to enable the harmonious transition of buses through a region or network without artificial interventions.

(2.3) The problem is: *how do we build a better traffic light control system that can fully integrate and handle bus priority properly?* There are many factors to consider, including:

- **Delay, throughput and fairness.** Bus priority requires the control system to trade off delay to buses against delay to other vehicles. In addition to low delay, high throughput and fairness between different inputs are also desirable. How should these objectives be combined and made harmonious?
- **Sensors.** Advances in technology have dramatically increased the amount and quality of data available to the controller. Typically, inductive loop sensors measure flows on the inputs (and sometimes

outputs) for all vehicles, and buses report high frequency (say every 10s) positional data (Automatic Vehicle Location). If similar positional data were available for all vehicles, how much could be gained by using these data?

- **Human factors.** Drivers are used to traffic lights that follow a predictable sequence of 'stages' — they anticipate their green light based on who had the last green light. The controller can in principle set the lights in any way that does not create a conflict, which may lead to increased efficiency (how much?) but also increased incidence of driver error. It is desirable to use the traditional stage sequence, as this increases the marketability of the end solution.
- **Network Effects.** Buses typically run down corridors with multiple junctions and may intersect with other bus corridors. How much might be gained by centralised coordinated control compared with current decentralised control?

(2.4) The evaluation data available for the problem include several road network fragments, origin-destination flows for cars, schedules for buses, and several real life datasets that show the performance of existing control systems (e.g. replications of MOVA and SCOOT, which are widely used in practice) on these examples.

3 Lumped-element Discrete Time Model

3.1 Parameterizing the Problem

(3.1) This approach utilizes a lumped-element discrete time model to optimally control the traffic flow through a junction. In simple terms, this means that current traffic conditions on each arm are represented by a single variable –the queue length– and these variables are updated at discrete time steps. The control law is designed to give priority to incoming buses, all the while ensuring a degree of fairness for the remaining traffic. This is achieved by defining a common metric for buses and regular traffic, which weights bus delays more heavily than other delays. By using a common metric, rather than employing exceptions as in the standard MOVA implementations, buses are given priority under regular traffic conditions, but if an arm is subject to excessively heavy traffic, this is automatically taken into account in the assignment of green times. The metric, which is essentially a weighted distance of the current state from the ideal state of zero delays on all arms of the junction, constitutes the cost function that is minimized by the optimal controller.

(3.2) Control of the junction is considered over one full cycle, T_c, the time taken for each arm to recieve a green light. The cycle length T_c, the sequence in which arms are allocated greentime, and the amount of time the lights are red between green signals are considered fixed, while the controller is allowed to chose the time when each arm receives a green light. As the redtime R is fixed for a given junction, the greentime T_g is given by $T_g = T_c - R$; since T_c is fixed, the total length of the greentime in this approach is not optimized. However, it is a logical extension to add this to the analysis, indeed, the ODE approach considers this explicitly.

(3.3) Consider a junction with n incoming lanes. It is assumed that the traffic never backs up from the subsequent junctions, meaning that the number of vehicles that are allowed to exit the junction in a given cycle time is only limited by the junction capacity. Moreover, the destination of the vehicles that enter the junction is ignored. On the i-th arm the number of vehicles at the discrete time t_j (NB this is an integer label denoting how many cycles have passed), $n_i(t_j)$, is given by the number $n_i(t_{j-1})$ of vehicles in the queue at the end of the previous cycle, the number $v_i(t_j)$ of vehicles entering the queue over a full cycle, and the number $Out_i(t_j)$ of vehicles that left the queue in the given greentime. The update of the queue length on arm i is thus given by the following equation.

$$n_i(t_j) = n_i(t_{j-1}) + v_i(t_j) - Out_i(t_j) \tag{1}$$

The number of vehicles that leave the junction Out_i is a function of the length of the queue. If the queue is sufficiently long, not all the vehicles can exit the queue when the lights turn green. The arm is saturated. Otherwise, when the queue is sufficiently short, all the vehicles can leave the queue in the greentime and the arm is unsaturated.

(3.4) The saturation flow S_i is the maximum number of vehicles that can leave the junction per unit greentime. It is a (constant) property of the junction determined from the width of the approach lane, the angle vehicles turn through on leaving the junction, etc. If the product of the saturation flow and the fraction of greentime of the arm i over the total green time, denoted α_i, is less than the length of the queue not all vehicles exit the queue. Otherwise, when the arm is unsaturated, all the vehicles leave the queue.

$$Out_i(t_j) := \begin{cases} S_i(t_j) \cdot \alpha_i(t_j) & \text{Saturated} \\ n_i(t_{j-1}) + v_i(t_j) & \text{Unsaturated} \end{cases} \tag{2}$$

The average delay on the i-th arm at discrete time t_j, denoted $\tau_i(t_j)$, is given by the ratio of the number of vehicles in the queue $n_i(t_j)$ and the number of vehicles that leave the queue $Out_i(t_j)$.

$$\tau_i(t_j) := \frac{n_i(t_j)}{Out_i(t_j)} \tag{3}$$

To gain some understanding of the delay measure it is worth considering a number of cases. If the queue is not saturated, the average delay time is zero. This in effect says that an undersaturated flow implies that no extra delay is expected. If the flow is saturated then the delay will show what fraction of a cycle you must wait, over and above the time spent stopped in the undersaturated case.

(3.5) Using this delay measure, a cost function can be defined. This is denoted L and given by:

$$\begin{aligned} L(t_j) &:= \tau^T(t_j)\Pi(t_j)\tau(t_j) \qquad \text{where } \Pi := diag(\omega_1, \omega_2, ..., \omega_n) \\ &= \sum_{i=1}^{n} \omega_i \tau_i^2|_{t=t_j} \end{aligned} \tag{4}$$

The diagonal matrix Π defines a metric, that is, a weighted distance between the current delays and the ideal case where $\tau_i = 0$ for all arms i. It depends whether a bus is in a given queue at time t_j. The diagonal elements are the relative weights, or priority, given to the different arms. If no buses are on the i-th arm, $\omega_i = 1$. If a bus is on the i-th arm, the weight is parameterized by p. The choice of p is a design parameter which can be varied. A larger p will give higher priority to buses with respect to regular traffic, at the cost of potentially generating longer delays on arms with no buses.

(3.6) L is a measure of the total delay time. As we have seen in the description above, the only variables under control of the algorithm are durations of the green time on each arm. Let us call α the vector of the normalised green times, that is, the duration of each green time divided by the total green time. At each time step, our algorithm attempts to chose α such that the cost function (4) decreases as much as possible, under the assumption that the incoming flow, measured during the current time step, remains constant in the future. This is equivalent to minimizing the function $L(t_{j+1})$, where the vector $\tau(t_{j+1})$ is computed by projecting the current state forward by one time step using (1). This is a fairly standard minimization problem, which can be solved using off-the-shelf optimization algorithms. The algorithm is shown schematically in Figure 1.

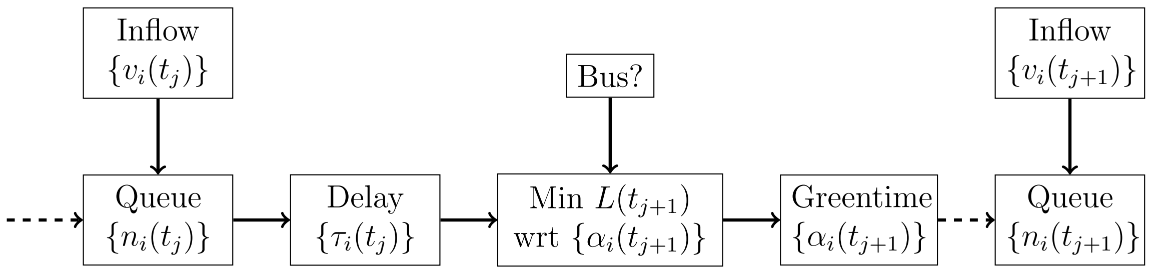

Inflow $\{v_i(t_j)\}$
Bus?
Inflow $\{v_i(t_{j+1})\}$
Queue $\{n_i(t_j)\}$
Delay $\{\tau_i(t_j)\}$
Min $L(t_{j+1})$ wrt $\{\alpha_i(t_{j+1})\}$
Greentime $\{\alpha_i(t_{j+1})\}$
Queue $\{n_i(t_{j+1})\}$

Figure 1: Schematic of Model operation: A single iteration of the alogorithm, showing the evolution of the state of the system $\{n_i(t_j)\}$ at time t_j to the next state of the system $\{n_i(t_{j+1})\}$ at time t_{j+1}.

3.2 The Model

(3.7) A test model was implemented in python. It takes as input the flow into each arm over each full cycle and the initial queue length. The inflow was determined from Halcrow's MOVA simulation of a comparable junction. This provides a realistic level of flow, as well as measures to test how well the model performs in comparison to current methods. The average delay times for cars and buses can be directly compared. For simplicity, and to fit with the data provided by Halcrow, it was assumed that the cycle length was fixed at one minute. The junction had five incoming arms, it is shown schematically in Figure 2.

(3.8) In the test implementation, each cycle of the junction considered consists of three stages. Each cycle has a total redtime of $R = 19s$ in accordance with the Halcrow plan. During the first stage, green light is shown for arms 3, and 5, in the second for arms 1, 3, and 4, and in the third for arms 1, and 2. During each cycle, the minimum greentime of each stage is $5s$.

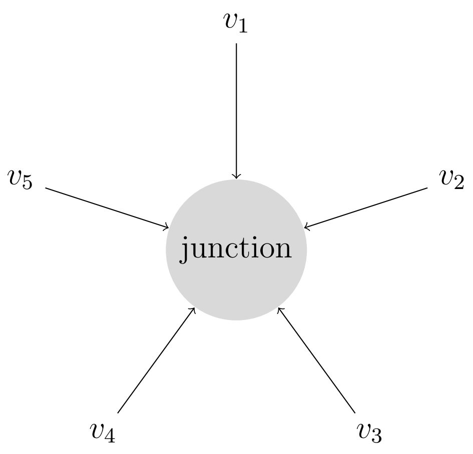


Figure 2: Diagram of the junction used in the Model.

(3.9) The vehicle data is provided as a set of inflows per minute over a single hour. It assumes that vehicles enter the queues at the edge of the simulation area. In reality, data on vehicles joining a queue is gained by induction loops at a maximum distance of around $100m$ from the junction. Halcrow have run a MOVA simulation optimizing the junction control with the existing bus priority methods. This allows the average delay on each arm, for buses and for other vehicles, to be determined.

(3.10) For each time step (of one minute) the model calculates the optimum set of fractions of green time α_i, as described above. The chosen green times are then used for one cycle and the new length of the queue is updated according to the model (1). The overall average delay times (as a fraction of total greentime) have been determined for each arm, and the results are shown in the figures that follow. The model has been run with different parameters p for the bus weighting.

(3.11) Since this approach is designed to be used in a saturated flow scenario, the raw data for inflow provided by Halcrow was scaled up. This allowed the results to be analyzed in the regime intended. The delay times that the model creates are thus not directly comparable to those generated from Halcrow's MOVA data. However, by comparing the percentage decrease in delay times for buses with a given weighting with the decrease in delay time using the existing bus priority methods, a meaningful comparison can be made.

(3.12) Figure 3 shows the average dely time τ_i on each arm of the junction as a function of the bus weighting parameter p. The delay is measured in number of cycles. The dashed lines show the average delay for buses on the given arms. As p increases, buses gain greater priority and have smaller delay times. The other vehicle types on the same arms also suffer smaller delay times. However, vehicles on arms with no buses, such as arm 4, lose out considerably. The oscillations and peaks, that buck the general decaying trend, are artifacts of the optimization procedure. These arise because the solution to the minimization problem is not unique; a choice has to be made as to which set of solutions $\{\alpha_i\}$ are used. Currently, the algorithm making the choice has not been calibrated. This could be rectified given more time.

(3.13) Other features to notice in Figure 3 are that for arm 5 when $p = 1$ the buses still appear to gain priority. Logically it is expected that their delay would be equal to that for other vehicles, however we believe that this apparent contradiction can easily be explained. There are few buses in the dataset

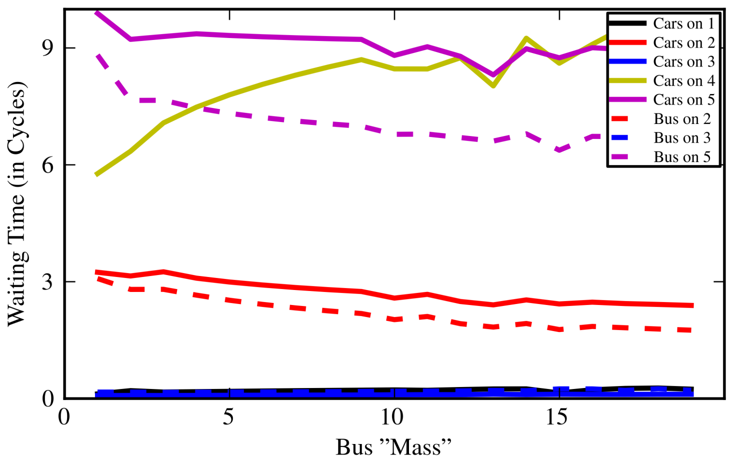


Figure 3: Model delay times Vs bus weight (or mass) the value p.

and they appear rarely. When they happen to appear on arm 5 there is lighter than average traffic flow - meaning the buses traverse the junction with less than average delay anyway. Such an anomoly highlights that testing must be performed on a greater variety of datasets to better assess the model.

(3.14) The table below shows the percentage increase in delay time (when buses are given a weighting of p=15) for the model by arm and the increase in delay time for the same arms for Halcrow's existing bus priority approach. Since buses can be tracked by GPS, it is assumed that one can determine when they approach a junction. Hence the weighting for a given arm is switched on four cycles before the bus actually joins the queue. It is switched off when it is estimated that the bus has left the queue. A bus is estimated to leave the queue by treating it as an average member of the queue. A negative percentage indicates less delay.

	Increase in delay	time for buses
Lane (i)	Model	Halcrow
2	−12%	−42%
3	−87%	+51%
5	−37%	−27%

(3.15) For arm 2, the proposed algorithm performs better than the existing bus priority method. Halcrow's method performs slightly better on arm 5, and much better on arm 3. Arm 3 is oversaturated, arm 5 is saturated and arm 2 is unsaturated. This indicates that the algorithm performs better in situations that are more saturated and worse as the junction becomes unsaturated. Importantly it shows that Halcrow's existing method works well in light traffic conditions. Although the large percentage increase in bus delay times on arms 3 from the model reflect that the delay time is very small in the first place, and any increase in delay, even if very small in absolute value, creates a large percentage difference. In absolute terms the increase in delay on arm 3 is actually small.

(3.16) The lumped-element discrete time model approach works well in heavy traffic, but worse than existing methods in light traffic. This suggests that to implement better junction control it should be combined with a strategy that works well in unsaturated condition - such at the Markov Decision Process approach.

3.3 Weaknesses of the Approach

(3.17) There are a number of subtle presumtions underlying this approach. The most important assumption is that vehicles enter or leave the queue in-

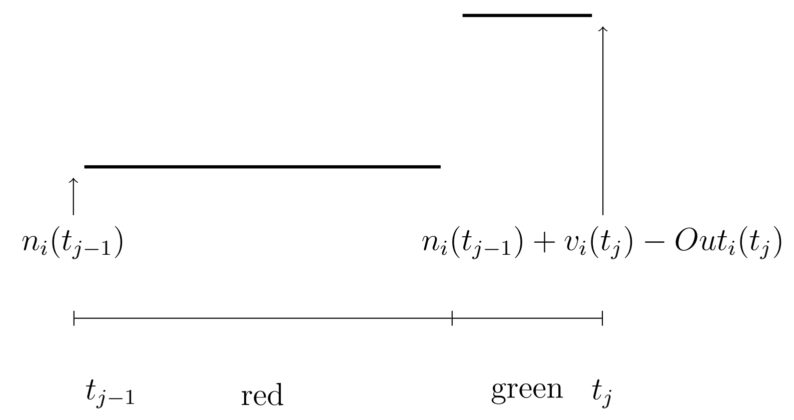


Figure 4: Vehicles enter and leave queue instantaneously.

stantaneously at the end of the given discrete time period. This is clearly not what is physically happening: there is a discrete flow of vehicles into the queue and a discrete flow out while the lights are green. Two natural questions arise from this assumption. When is it valid? And what are this assumptions' limitations?

(3.18) The number of vehicles that leave the queue Out_i are always approximated well - this value depends on the fraction of the total green time allocated (for the given arm), α_i. The assumption is good if the queue is saturated. The additional vehicles that join the queue in a given discrete time do not all leave the queue in the same discrete time. The joining and exiting of the queue can thus be visualized as occurring instantaneously at the end of the cycle. With the same image in mind, the assumption is clearly good for the arm that has greentime last in the cycle. This is pictured diagramatically in Figure 4.

(3.19) The assumption that the inflow and outflow of the queue occurs instantaneously at the end of the cycle is poor if the queue is short and the greentime for a specific arm is not at the end of a period. In this case some of the vehicles that arrive over the cycle, v_i, join the queue after the light has been green meaning they cannot all exit the queue. In the model formalism however it is assumed that they would be able to leave the queue. The problem is shown diagramatically in Figure 5.

(3.20) However, if the additional assumptions are made, that the inflow v_i and the fraction of greentime α_i vary slowly between adjacent discrete time steps, then the inflow in arm i since the previous greentime is approximately v_i as one cycle T_c has been passed through. This means that the instantaneous inflow and outflow assumption can fruitfully be applied.

(3.21) The results in section 3.2 show that, as expected from these assumptions,

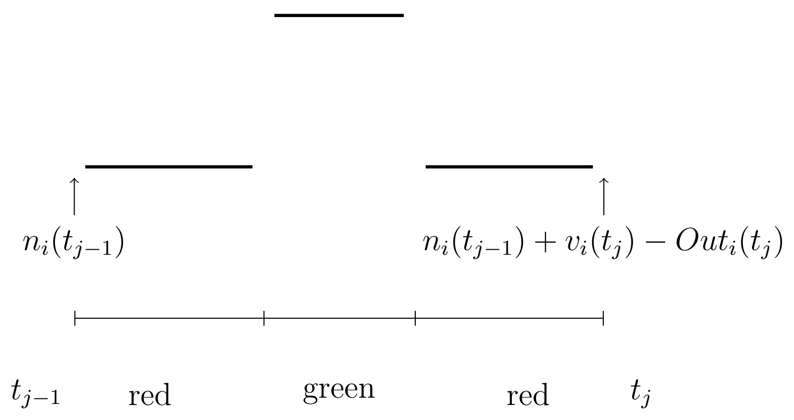


Figure 5: Poor assumption when the lights are red in the middle of the period

in saturated flows the lumped-element discrete time model works well. It ensures traffic flows smoothly on all arms, while significantly decreasing the time that buses on these arms wait. However, in unsaturated arms the model does not provide a reliable prediction of the state of the junction at the next cycle, which is necessary both for the correct function of the algorithm, and for the generation of the data in the simulations. In light traffic, we can assume that both the algorithm and the simulations are unreliable. Further tests need to be performed to determine how many arms need to be saturated before the lumped-element discrete time model works better than the current method. In addition, a greater variety of input data sets need to be run through the model to test whether the decreased bus delay times remain significant.

4 Ordinary Differential Equation Approach

4.1 Introduction

(4.1) In this section we consider an approach in which the traffic is viewed as a continuous stream. An analogy would be fluid flowing through a pipe. Some strengths of this approach are that it can easily be adapted to incorporate different flows into the junction and that it can describe the situation in which the junction is not busy. On the other hand, this approach may not be appropriate for modelling non-busy flows since such flows are not really continuous (in particular this presents a problem with regards to infrequent buses). The model also allows there to be a non-integer number of cars in the queue, but this could easily be fixed by taking the floor of the solution once the light turns red. Finally, we note that the model described assumes that each branch of traffic only gets a green light once per cycle. This will not be the case for many real-life junctions. Hopefully adapting the model to deal with such situations would not be too difficult a task.

(4.2) First we develop a model that only deals with cars, and then we will describe how the model can be adapted to incorporate a bus.

4.2 Cars-only model

(4.3) We suppose that there are N different branches to the junction, indexed by i. We let $c_i(t)$ be the number of cars queueing on branch i and suppose that this number is $c_{0,i}$ initially. If $k_i(t)$ is the rate at which cars join the queue and $r_i(t)$ is the rate at which cars leave the queue, then "conservation of cars" tells us that the number of cars in the queue is governed by the equation

$$\frac{dc_i}{dt} = k_i(t) - r_i(t), \quad c_i(0) = c_{0,i}, \quad \text{for } i = 1, \ldots, N. \tag{5}$$

(4.4) The in-flow $k_i(t)$ will be known from sensors at any given time. However, our algorithm requires knowledge of $k_i(t)$ throughout the upcoming cycle, which will not be known in advance. We therefore assume that the in-flow remains constant throughout the cycle and that its value k_i is equal to that measured at the start of the cycle.

(4.5) While the light is red, the out-flow $r_i(t)$ is, of course, zero. When the light turns green, there is an initial start-up period in which the out-flow increases from zero to its saturation rate S_i. For simplicity we have assumed that this is a linear increase, but the model can easily be adapted to use a more realistic function (one possibility that we have discussed is to use a hyperbolic tangent function). Cars will then flow out at the saturation flow rate until the queue is empty, after which time traffic will flow freely through the green light so that the out-flow $r_i(t)$ is equal to the in-flow k_i. Mathematically, we have

$$r_i(t) = \begin{cases} 0, & \text{before the light turns green,} \\ \dfrac{(t - t_{0,i})}{\tau_d} S_i, & \text{during the initial start-up period,} \\ S_i, & \text{while the queue empties at its saturation flow rate,} \\ k_i, & \text{after the queue is empty and traffic is flowing freely,} \\ 0, & \text{after the light turns red again,} \end{cases} \tag{6}$$

where $t_{0,i}$ is the time at which the light turns green, and τ_d is the duration of the start-up period.

(4.6) We note that some of these periods may get skipped under certain conditions. Firstly, if the initial queue length is sufficiently short, then the queue may empty during the initial start-up period and then the saturation flow period will be skipped. Secondly, if the light turns back to red before the queue is emptied, then the free-flow period will be skipped. (Since τ_d is typically small, at most one of these possibilities will occur for each queue).

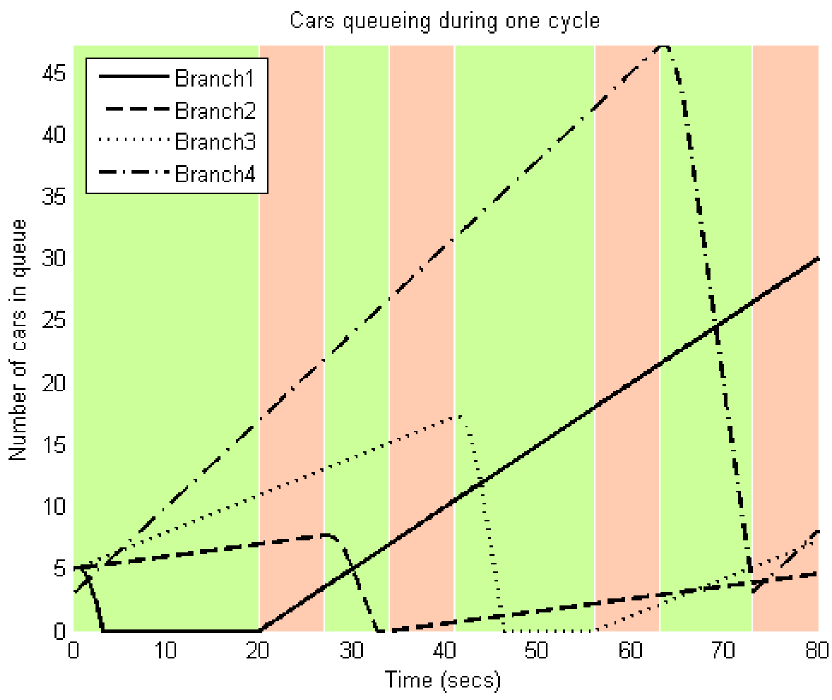


Figure 6: Example solution of cars-only model without optimisation

(4.7) Once we have determined the appropriate form for the out-flow $r_i(t)$, we can solve the equations (5) to determine the number of cars $c_i(t)$ in each queue in terms of the green times given to each branch. We can then determine the total cycle-averaged queue length

$$\mathcal{L} = \sum_{i=1}^{N} \frac{1}{T} \int_0^T c_i(t)\, dt, \tag{7}$$

as a function of the green times, where T is the total cycle time, which is the sum of all the green times plus the intergreen times. We then want to choose the green times so as to minimise this function subject to the constraints that (i) the green time for each branch must exceed some minimum value, and (ii) the total cycle time must not exceed some maximum value.

(4.8) We can then repeat the process for the next cycle by using the values of the queue length at the end of the current cycle, *i.e.* $c_i(T)$, as the new values of the initial queue length $c_{0,i}$.

4.3 Bus model

(4.9) We now extend the model to include a bus. The approach is similar to the cars-only model, except now we have to consider the flow of the bus as well. Analogous to (5), we now have

$$\frac{dc_i}{dt} = k_i(t) - r_i(t), \quad c_i(0) = c_{0,i}, \tag{8}$$

$$\frac{db_i}{dt} = p_i(t) - q_i(t), \quad b_i(0) = 0, \quad \text{for } i = 1, \dots, N. \tag{9}$$

Here, $p_i(t)$ is the rate at which buses enter the queue and $q_i(t)$ is the rate at which buses exit the queue. The quantities $k_i(t)$ and $p_i(t)$ are thus related, and the same can be said for $r_i(t)$ and $q_i(t)$. If $P_i(t)$ denotes the total in-flux of vehicles and $Q_i(t)$ denotes the total out-flux of vehicles, then we have the equations

$$k_i(t) + 2p_i(t) = P_i(t), \tag{10}$$

$$r_i(t) + 2q_i(t) = Q_i(t), \tag{11}$$

where the factor 2 comes from the fact that a bus takes twice as long as a car to enter and exit the queue. The functions $P_i(t)$ and $Q_i(t)$ will behave like $k_i(t)$ and $r_i(t)$, respectively, in the cars-only model. The form of $k_i(t)$ and $p_i(t)$ are

$$k_i(t) = \begin{cases} P_i(t), & \text{if } 0 \le t < t_b, \\ 0 & \text{if } t_b \le t < t_c, \\ P_i(t), & \text{if } t \ge t_c, \end{cases} \quad p_i(t) = \begin{cases} 0, & \text{if } 0 \le t < t_b, \\ \frac{P_i(t)}{2} & \text{if } t_b \le t < t_c, \\ 0, & \text{if } t \ge t_c, \end{cases} \tag{12}$$

where t_b is the time at which the bus starts to enter the queue, and t_c is the time at which the bus has entered the queue. Similarly, the out-flux of cars and buses are given by

$$r_i(t) = \begin{cases} Q_i(t) & \text{if } 0 \le t < t_d, \\ 0 & \text{if } t_d \le t < t_e, \\ Q_i(t) & \text{if } t \ge t_e, \end{cases} \quad q_i(t) = \begin{cases} 0 & \text{if } 0 \le t < t_d, \\ \frac{Q_i(t)}{2} & \text{if } t_d \le t < t_e, \\ 0 & \text{if } t \ge t_e, \end{cases} \tag{13}$$

where t_d is the time at which the bus starts to exit the queue, and t_e is the time at which the bus has exited the queue. These times t_d and t_e are to be determined, and they depend on the green time. Equations (12) and (13) simply mean that when the bus is entering or exiting the queue, the cars aren't, and vice-versa.

(4.10) Next, we combine the buses and cars together. We consider the quantity

$$\frac{\mathrm{d}}{\mathrm{d}t}(c_i + W_i b_i) = \left(1 - \frac{W_i}{2}\right)[k_i(t) - r_i(t)] + \frac{W_i}{2}[P_i(t) - Q_i(t)], \tag{14}$$

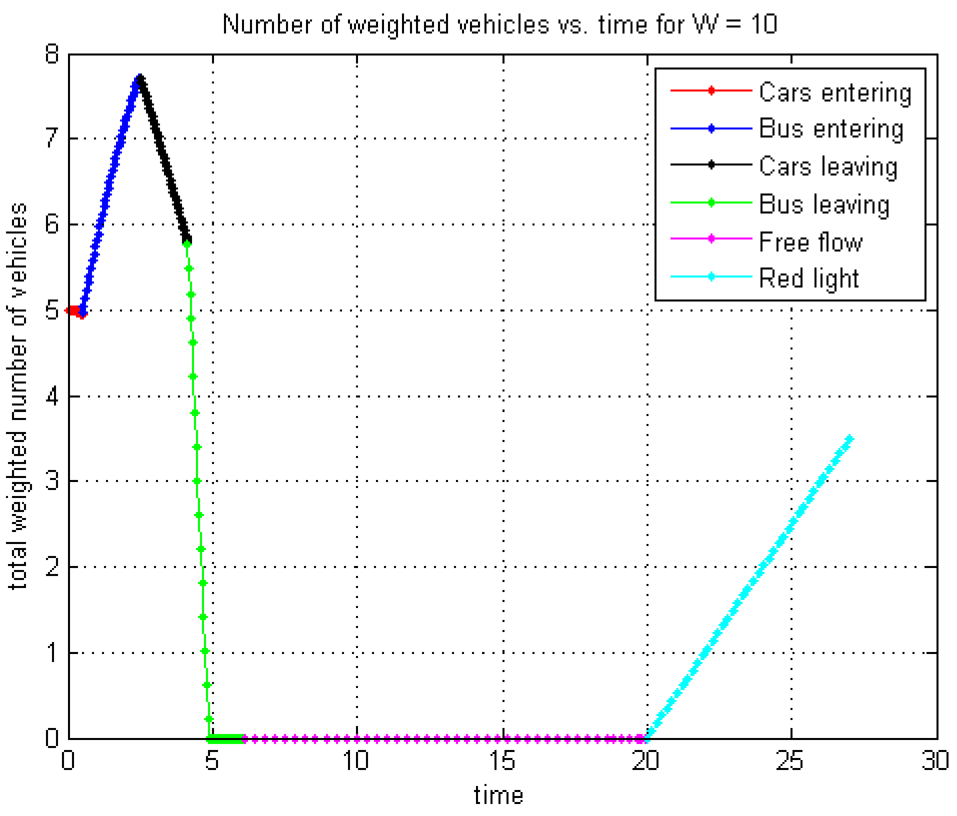
where W_i is the weight that we give to a bus. If $W_i = 2$, then it means that the cars and the buses are given the same priority, while if $W_i > 2$, then the buses are given priority over cars.

(4.11) The results of a simulation with $W_i = 10$ is shown in Figure 10. The blue curve represents the period where the bus is entering the queue, and since there is a higher priority for the bus, the total weighted number of vehicles increases drastically. The green curve represents the time the bus exits the queue, and we can see that there is also a similar steep decrease in the weighted number of vehicles as well. The light turns red at $t = 20$, and this is where the queue starts to build up again.

(4.12) Upon successfully simulating the flow of traffic on a road with no junctions, but one traffic light, the next step would be to consider the same bus model for an actual junction, and subsequently add more buses into the model. We would then want to choose the green times such that the total cycle-averaged queue length

$$\mathcal{L} = \sum_{i=1}^{N} \frac{1}{T} \int_0^T c_i(t)\,dt, \tag{15}$$

is minimised, subject to the constraints that (i) the green time for each branch must exceed some minimum value, and (ii) the total cycle time must not exceed some maximum value.


Number of weighted vehicles vs. time for W = 10
total weighted number of vehicles
time
0
1
2
3
4
5
6
7
8
0
5
10
15
20
25
30
Cars entering
Bus entering
Cars leaving
Bus leaving
Free flow
Red light

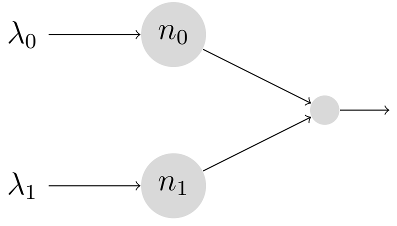


Figure 7: Diagram of the Y junction used in MDP formulation 1. Traffic flows from left to right. See the text for the definitions of λ_i and n_i.

5 Markov Decision Process Approach

(5.1) Markov Decision Processes (MDPs) are a formalism for making sequential decisions under uncertainty. This section formulates the junction control problem in the language of MDPs. Two formulations are presented, both for the simple 'Y' junction in Figure 7, but the MDP approach can in principle scale to include more complicated junctions, and indeed multiple junctions to be managed simultaneously. The first formulation includes only one traffic type (cars). The second formulation considers cars and buses separately, so that they can be weighted differently; it also allows for minimum and maximum green times to be enforced for each stage. Preliminary results are presented for each formulation.

(5.2) To phrase the problem in the language of MDPs, we have to describe our problem in the following terms of *states* (which are discrete), *actions*, *transition probabilities* (between states, depending on actions) and *rewards* (objective). Time is discrete. In each time step, we collect a reward for the current state, then we take an action that determines a probability distribution for our next state. A solution is a *policy* that determines which action to take in each state. An optimal policy is one that collects the maximal amount of reward over time.

5.1 MDP Formulation 1

(5.3) Here we aim to find the simplest non-trivial formulation of the problem as an MDP. The state of the system at a given time is described by the vector

$$s = (n_0, n_1, j)$$

where n_i is the number of cars queueing on arm i, and j the current stage — $i = j$ when i has the green. The possible actions are to either stay in the current stage or change to the next stage; these are denoted by $a = 0$ and $a = 1$, respectively.

(5.4) The transition probabilities define the dynamics of the system — that is, how vehicles arrive at and depart from the queue on each arm. The randomness in the system is due to the arriving vehicles. For each arm i, let

N_i be the number of new vehicles that arrive in the current time step. It is assumed that $N_i \sim \text{Poisson}(\lambda_i)$ for known arrival rates λ_i.

(5.5) The dynamics of the vehicles leaving the queue are goverened by a single parameter: let f be the maximum number of vehicles that can leave the queue in a single time step. The length of the queue in the next state is then given by

$$n'_i = n_i - [i = j] \times \min\{n_i, f\} + N_i$$

where $[i = j]$ denotes an indicator function that is 1 when $i = j$ and 0 otherwise. That is, if arm i has the green, then up to f vehicles can leave during this timestep.

(5.6) The next stage, j', is determined by the action, according to

$$j' = \begin{cases} j & a = 0 \\ 1 & a = 1, j = 0 \\ 0 & a = 1, j = 1 \end{cases}$$

(5.7) The reward for state s is defined to be

$$R(s) = -(n_0 + n_1)$$

which is the negative of the total number of vehicles in the system. The rationale is that the total reward accumulated over time is the negative of the total delay to vehicles in the system, so maximum long run reward corresponds to minimum long run total delay.

5.2 Exact Solution Methods

(5.8) Once the problem is specified in this form, standard techniques can be used to obtain solutions for small problems. Here, 'small' refers mainly to the number of possible states. For MDP formulation 1, the number of possible states is in principle unbounded, because (particularly if vehicles are arriving faster than they can be served) the queues could grow to any length. In order to solve the problem exactly, it is therefore necessary to truncate the queues at some finite length $\bar{n}$, so that the n_i are in the range $\{0, 1, \dots, \bar{n}\}$. When $n_i = \bar{n}$, this means that there are $\bar{n}$ or more cars in queue i. This, together with the fact that $j \in \{0, 1\}$, implies that size of the state space is $2\bar{n}^2$.

(5.9) Here a 'solution' is a 'policy' that specifies what action to take ($a = 0$ or 1) in each state. An optimal policy is one that maximises the amount of reward we collect over time. To formalise this, we have to deal with one more technicality: if we ran the system for an infinitely long time, some policies might accrue infinite reward, which would make them uncomparable. To make all policies comparable, we introduce a 'discount factor', γ

with $0 < \gamma < 1$, and multiply future rewards by γ — intuitively, this makes rewards collected now slightly more valuable than rewards collected in the future, and technically it guarantees that all policies collect finite long run discounted reward.

(5.10) Formally, we wish to find an optimal value function $V^*(s)$ that satisfies the Bellman optimality equations

$$V^*(s) = R(s) + \max_a \gamma \sum_{s'} \Pr(s'|s,a)V^*(s')$$

and a corresponding optimal policy $\pi^*(s)$ that satisfies

$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} \Pr(s'|s,a)V^*(s')$$

for all states, where $\Pr(s'|s,a)$ denotes the probability of transitioning to state s' from state s if that we take action a. That is, the optimal value of a state is its immediate reward plus the expected discounted optimal value that we receive by following an optimal policy in all subsequent states. The optimal policy is determined by choosing in each state the action that maximises the expected optimal value of its successors. An optimal policy can be obtained by an algorithm called policy iteration. A freely available implementation of this algorithm is available at `https://github.com/jdleesmiller/finite_mdp`. For details (and a much better explanation), please refer to [1].

5.3 Example Results from MDP Formulation 1

(5.11) In practice, the result of policy iteration is a table that maps each state to the optimal action. The optimal policy for the example network with parameters

$$\lambda_0 = 0.1, \lambda_1 = 0.5, f = 1, \bar{n} = 1, \gamma = 0.95$$

is given in Table 1. This MDP has 8 states and 48 non-zero transition probabilities.

(5.12) The optimal policy determined by the algorithm is intuitively reasonable. For example, because there is more demand on arm 1 than arm 0 (because $\lambda_0 < \lambda_1$), when the system is empty ($n_0 = n_1 = 0$), the system prefers to give the green light to arm 1 (if $j = 0$, the option action is to change ($a = 1$), and if $j = 1$, the optimal action is to stay ($a = 0$).

(5.13) One interesting feature is that when the queues are at their maximum permitted length ($n_0 = n_1 = \bar{n} = 1$), the optimal policy is to switch between stage 0 and stage 1 in every time step. We have not specified the time step in physical terms as yet, but if it is short, say 5s, then switching the lights every 5s would not be physical. This motivated the addition of minimum and maximum stage times in MDP Formulation 2.

n_0	n_1	j	action	value
0	0	0	change	-6.588
0	0	1	stay	-6.370
0	1	0	change	-8.400
0	1	1	stay	-7.779
1	0	0	change	-7.779
1	0	1	change	-8.546
1	1	0	change	-9.597
1	1	1	change	-10.069

Table 1: Optimal policy for an example of MDP formulation 1.

5.4 MDP Formulation 2

(5.14) Here we make the following significant changes to MDP formulation 1, namely:

1. Instead of representing the length of the queue on arm i with a single number, n_i, we break it down into the number of cars c_i and the number of buses b_i in the queue. Similarly, the numbers of new buses and new cars arriving in each time step are also specified separately as N_{ib} and N_{ic} for buses and cars, respectively. It is assumed that

$$N_{ib} \sim \text{Poisson}(\lambda_{ib})$$

and

$$N_{ic} \sim \text{Poisson}(\lambda_{ic})$$

for known arrival rates λ_{ib} for buses and λ_{ic} for cars. Note that the assumption that buses are Poisson should be regarded as suspicious, because they should in principle be running on schedules (in most cases); this is a limitation of this formulation.

2. To represent minimum and maximum stage times, we add an additional component τ to the state vector; τ counts the number of time steps since the last stage change. The actions available in each state are restricted to enforce the minimum and maximum stage times. In particular, let the minimum and maximum stage times for stage j be denoted $\tau_j^{\min}$ and $\tau_j^{\max}$. The action space for each state is restricted such that $a = 0$ (stay) if $\tau < \tau_j^{\min}$ and $a = 1$ (change) if $\tau = \tau_j^{\max}$. When $\tau_j^{\min} \leq \tau < \tau_j^{\max}$, both $a = 0$ and $a = 1$ actions are allowed, at the discretion of the controller.

3. To permit more than two stages, we allow j to range over $\{0, 1, \ldots, j_{\max}\}$, where $j_{\max}$ is one less than the number of stages, and we replace the single parameter f for junction throughput with a throughput matrix $\mathbf{F}$ with entries f_{ij} that give the throughput in one time step from arm i when in stage j. This allows us to represent 'intergreen' times by adding a dummy stage with zero flow on both arms. For example, the

two zero rows in

$$\mathbf{F} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{pmatrix}$$

are intergreens between the stages for the Y junction. The minimum and maximum stage times can be used to force the length of the intergreen time to a particular value by choosing $\tau_j^{\min} = \tau_j^{\max}$.

4. The reward function is now the negative of a weighted sum of bus and car queue lengths, namely

$$R(s) = -\sum_i (w_b b_i + w_c c_i)$$

where w_b and w_c are the weights for cars and buses.

(5.15) The main challenge in this formulation is the definition of the transition probabilities for the the 'mixed' queue of both buses and cars, because we do not know the order of the vehicles in the queue. That is, if a bus and two cars are waiting on one arm, and we know that one vehicle can leave the queue in the current time step, we do not know for sure whether it will be a bus or a car. It is worth noting that this may not be entirely unrealistic: if there are multiple lanes and error in the GPS positions reported by the vehicles, the actual order of the queue may be somewhat uncertain. However, this should be regarded as a limitation of this formulation, albeit not necessarily a severe one.

(5.16) The transition process that governs the departure of vehicles from the queue is one of sampling with replacement. For example, if we have two cars and one bus, and we can remove two vehicles in one timestep, and the first one is a bus, we know that the second one will be a car, because there are no more buses. This behaviour is captured by the Hypergeometric distribution. Let F_{ijb} and F_{ijc} denote the throughput of buses and cars, respectively, through the junction in one time step. It is assumed that

$$\begin{aligned} F_{ijb} &\sim \text{Hypergeometric}(b_i, c_i, \min\{b_i + c_i, f_{ij}\}) \\ F_{ijc} &\sim \text{Hypergeometric}(c_i, b_i, \min\{b_i + c_i, f_{ij}\}). \end{aligned}$$

For example, F_{ijb} is the number of buses that we draw from the queue, given that there are b_i buses and c_i cars in the queue to start with and we draw $\min\{b_i + c_i, f_{ij}\}$ vehicles in total.

5.5 Example Results from MDP Formulation 2

(5.17) The largest example solved so far has 800 states and 27k non-zero transition probabilities. The main limitation at this point is actually that the program that generates the MDP from the input parameters is very slow — it took about an hour to generate the 800-state problem, but once it has been generated, it took only a few seconds to solve. With improvements to its efficiency, somewhat larger examples could probably be solved.

(5.18) The smallest non-trivial example that exercises all of the features of MDP model 2, with parameters

$$\begin{gathered}\lambda_{0c} = 0.1, \lambda_{1c} = 0.5, \lambda_{0b} = 0.01, \lambda_{1b} = 0.05, \\ w_b = 2, w_c = 1, \bar{n} = 2, \gamma = 0.95, \\ \tau_{ij}^{\min} = 1 \; \forall \; j, \tau_{i1}^{\max} = \tau_{i3}^{\max} = 1, \tau_{i0}^{\max} = \tau_{i2}^{\max} = 2\end{gathered} \tag{16}$$

and

$$\mathbf{F} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \tag{17}$$

has 216 states, so it is not practical to present the full optimal policy here. A subset of the optimal policy is shown in Table 2.

5.6 Possible Extensions of the MDP Approach

(5.19) The general approach is to take the road network to be controlled and represent it as a directed graph with edges that represent the roads. At each node in this graph, there will a number of cars and a number of buses (analogous to b_i and c_i). Each of these nodes represents a physical region (a stretch of say 50m of road), and so has a natural maximum capacity of buses and cars (a sort of cellular automata approach). The transition probabilities govern the movement of vehicles forward in the graph, according to these capacity constraints and the same sorts of Hypergeometric transition probabilities that are used in MDP Formulation 2. The action space is now the joint action space over all junctions in the road network, because they can be controlled independently — each one can either stay in its current stage or move to its next stage, in each time step. Each junction has its own throughput matrix (analogous to **F**) that governs the movement of vehicles through the junction and either out of the system or into the approach for the next junction.

(5.20) The challenge for this general approach is that the size of the state space will grow very rapidly, and for any practical problem will likely be well beyond what it is likely to be possible to solve exactly using policy iteration. In general, the computational complexity of finding an optimal policy is polynomial in the number of states; the problem is that the number of states is exponential in essentially all of the parameters of interest. There is, however, a large literature on approximate dynamic programming, which can provide approximate solutions for extremely large MDPs. MDP-based controllers using ADP have been successful in controlling lifts (elevators), for example. ADP works mainly by representing the value function approximately. Devising appropriately heuristics for representing the value function will require knowledge of traffic engineering (and considerable experimentation) in order to isolate characteristics of the state vector are really important in determining the value function. This approach has the potential to deliver a junction control method that is both principled and practical.

Bibliography

[1] Sutton and Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge MA, 1998.

b_0	c_0	b_1	b_2	j	τ	action	value	b_0	c_0	b_1	b_2	j	τ	action	value
0	0	0	0	0	0	switch	-63.39829	0	2	0	0	0	0	switch	-127.20526
0	0	0	0	0	1	switch	-63.39829	0	2	0	0	0	1	switch	-127.20526
0	0	0	0	1	0	switch	-63.11068	0	2	0	0	1	0	switch	-126.71293
0	0	0	0	2	0	hold	-63.78838	0	2	0	0	2	0	hold	-127.49957
0	0	0	0	2	1	switch	-64.42649	0	2	0	0	2	1	switch	-128.29429
0	0	0	0	3	0	switch	-63.92799	0	2	0	0	3	0	switch	-127.84724
0	0	0	1	0	0	switch	-71.10782	0	2	0	1	0	0	switch	-135.92656
0	0	0	1	0	1	switch	-71.10782	0	2	0	1	0	1	switch	-135.92656
0	0	0	1	1	0	switch	-68.96821	0	2	0	1	1	0	switch	-133.54806
0	0	0	1	2	0	hold	-65.79818	0	2	0	1	2	0	hold	-129.50927
0	0	0	1	2	1	switch	-66.43629	0	2	0	1	2	1	switch	-130.30400
0	0	0	1	3	0	switch	-72.65085	0	2	0	1	3	0	switch	-137.38677
0	0	1	0	0	0	switch	-67.95676	0	2	1	0	0	0	switch	-133.14622
0	0	1	0	0	1	switch	-67.95676	0	2	1	0	0	1	switch	-133.14622
0	0	1	0	1	0	switch	-66.82507	0	2	1	0	1	0	switch	-131.65734
0	0	1	0	2	0	hold	-64.79325	0	2	1	0	2	0	hold	-128.50440
0	0	1	0	2	1	switch	-65.43136	0	2	1	0	2	1	switch	-129.29912
0	0	1	0	3	0	switch	-68.54112	0	2	1	0	3	0	switch	-133.50662
0	0	0	2	0	0	switch	-75.51333	0	2	0	2	0	0	switch	-138.94328
0	0	0	2	0	1	switch	-75.51333	0	2	0	2	0	1	switch	-138.94328
0	0	0	2	1	0	switch	-73.86579	0	2	0	2	1	0	switch	-137.83494
0	0	0	2	2	0	hold	-72.11909	0	2	0	2	2	0	hold	-136.66828
0	0	0	2	2	1	switch	-75.87923	0	2	0	2	2	1	switch	-140.37422
0	0	0	2	3	0	switch	-77.06780	0	2	0	2	3	0	switch	-139.99611
0	0	1	1	0	0	switch	-71.61982	0	2	1	1	0	0	switch	-134.65212
0	0	1	1	0	1	switch	-71.61982	0	2	1	1	0	1	switch	-134.65212
0	0	1	1	1	0	switch	-70.82984	0	2	1	1	1	0	switch	-134.37057
0	0	1	1	2	0	hold	-69.98658	0	2	1	1	2	0	hold	-134.07420
0	0	1	1	2	1	switch	-71.56150	0	2	1	1	2	1	switch	-135.34387
0	0	1	1	3	0	switch	-72.36021	0	2	1	1	3	0	switch	-134.91951
0	0	2	0	0	0	hold	-68.44518	0	2	2	0	0	0	hold	-131.60706
0	0	2	0	0	1	switch	-68.53912	0	2	2	0	0	1	switch	-132.21788
0	0	2	0	1	0	switch	-68.62580	0	2	2	0	1	0	switch	-132.86084
0	0	2	0	2	0	hold	-68.70358	0	2	2	0	2	0	hold	-133.53765
0	0	2	0	2	1	switch	-69.87090	0	2	2	0	2	1	switch	-134.51880
0	0	2	0	3	0	switch	-68.34536	0	2	2	0	3	0	switch	-131.02671
0	1	0	0	0	0	switch	-98.42783	1	1	0	0	0	0	switch	-103.20199
0	1	0	0	0	1	switch	-98.42783	1	1	0	0	0	1	switch	-103.20199
0	1	0	0	1	0	switch	-98.02734	1	1	0	0	1	0	switch	-102.78779
0	1	0	0	2	0	hold	-98.76719	1	1	0	0	2	0	hold	-103.53110
0	1	0	0	2	1	switch	-99.49302	1	1	0	0	2	1	switch	-104.26507
0	1	0	0	3	0	switch	-99.02110	1	1	0	0	3	0	switch	-103.79894
0	1	0	1	0	0	switch	-106.70613	1	1	0	1	0	0	switch	-111.54313
0	1	0	1	0	1	switch	-106.70613	1	1	0	1	0	1	switch	-111.54313
0	1	0	1	1	0	switch	-104.42971	1	1	0	1	1	0	switch	-109.25555
0	1	0	1	2	0	hold	-100.77880	1	1	0	1	2	0	hold	-105.54080
0	1	0	1	2	1	switch	-101.50462	1	1	0	1	2	1	switch	-106.27477
0	1	0	1	3	0	switch	-108.20849	1	1	0	1	3	0	switch	-113.03585
0	1	1	0	0	0	switch	-103.75379	1	1	1	0	0	0	switch	-108.61515
0	1	1	0	0	1	switch	-103.75379	1	1	1	0	0	1	switch	-108.61515
0	1	1	0	1	0	switch	-102.42229	1	1	1	0	1	0	switch	-107.26628
0	1	1	0	2	0	hold	-99.77297	1	1	1	0	2	0	hold	-104.53592
0	1	1	0	2	1	switch	-100.49880	1	1	1	0	2	1	switch	-105.26989
0	1	1	0	3	0	switch	-104.21701	1	1	1	0	3	0	switch	-109.05993
0	1	0	2	0	0	switch	-110.36088	1	1	0	2	0	0	switch	-115.08061
0	1	0	2	0	1	switch	-110.36088	1	1	0	2	0	1	switch	-115.08061
0	1	0	2	1	0	switch	-109.00395	1	1	0	2	1	0	switch	-113.76898
0	1	0	2	2	0	hold	-107.56915	1	1	0	2	2	0	hold	-112.38832
0	1	0	2	2	1	switch	-111.31013	1	1	0	2	2	1	switch	-116.11935
0	1	0	2	3	0	switch	-111.64467	1	1	0	2	3	0	switch	-116.32658
0	1	1	1	0	0	switch	-106.24037	1	1	1	1	0	0	switch	-110.94530
0	1	1	1	0	1	switch	-106.24037	1	1	1	1	0	1	switch	-110.94530
0	1	1	1	1	0	switch	-105.72744	1	1	1	1	1	0	switch	-110.46865
0	1	1	1	2	0	hold	-105.18132	1	1	1	1	2	0	hold	-109.96692
0	1	1	1	2	1	switch	-106.59264	1	1	1	1	2	1	switch	-111.35612
0	1	1	1	3	0	switch	-106.72258	1	1	1	1	3	0	switch	-111.39804
0	1	2	0	0	0	hold	-103.13207	1	1	2	0	0	0	hold	-107.84119
0	1	2	0	0	1	switch	-103.51042	1	1	2	0	0	1	switch	-108.25380
0	1	2	0	1	0	switch	-103.90172	1	1	2	0	1	0	switch	-108.68813
0	1	2	0	2	0	hold	-104.30542	1	1	2	0	2	0	hold	-109.14531
0	1	2	0	2	1	switch	-105.37352	1	1	2	0	2	1	switch	-110.19732
0	1	2	0	3	0	switch	-102.76692	1	1	2	0	3	0	switch	-107.44913
1	0	0	0	0	0	switch	-74.26411	2	0	0	0	0	0	switch	-79.19873
1	0	0	0	0	1	switch	-74.26411	2	0	0	0	0	1	switch	-79.19873

Table 2: A subset of the optimal policy for the MDP Formulation 2 example.