Australasian Transport Research Forum 2017 Proceedings
27 – 29 November 2017, Auckland, New Zealand
Publication website: http://www.atrf.info

Balancing computational requirements with performance in model predictive traffic control

Simon Stebbins[1], Mark Hickman[1], Jiwon Kim[1], Hai L Vu[2]

[1]School of Civil Engineering, University of Queensland

[2] Monash Institute of Transport Studies, Faculty of Engineering, Monash University

Email for correspondence: s.stebbins@uq.edu.au

Abstract

In some form or another, model predictive traffic control has been proposed for some decades. Because it predicts future traffic conditions, rather than reacting to past conditions in hindsight, it has advantages over adaptive traffic control techniques. However, it has not been widely adopted for a couple of reasons. Firstly, there are doubts about its suitability when fine-grained traffic data is unavailable. This objection can be addressed with the promise of vehicle-to-infrastructure (V2I) communication. Secondly, previous implementations have found that the control algorithm's computational complexity is exponential or worse. This paper addresses this objection by introducing the A* algorithm to significantly decrease computation time. Other methods for reducing computation time include setting a suitable prediction horizon, clustering vehicles into platoons and implementing an incremental version of the control algorithm. With these methods combined, it is shown through simulation, that a real-time model predictive control algorithm is practical because computation time is manageable without having an adverse effect on total delay.

1. Introduction

Previous work has demonstrated that model predictive control (MPC) used in traffic control is capable of significantly reducing delays, fuel usage and stoppage time, compared to optimised fixed time signal control. However, if an exhaustive search of the state space is carried out, this may require exponential computational complexity (or worse) (Papageorgiou et al., 2003). Generally, for a large number of arriving vehicles, it is not possible to run a scheduling algorithm in real time if an exhaustive search is conducted. Xie et al. (2012) proposed the use of elimination criteria to reduce the state space to polynomial complexity. Apart from some caveats, Stebbins et al. (2017) showed that this technique is applicable even when a microscopic, car-following model is used to simulate future traffic movements.

In this paper, an alternative method is employed to reduce the state space. Rather than comparing similar states and eliminating poor choices, the A* search algorithm (Hart et al., 1968) traverses the state space tree in an efficient manner to find the optimal solution. Although originally developed to find the shortest path in a network, the A* algorithm has general applicability to any graph-based problem, as long as a measure from the origin to the current node can be obtained, and a quantifiable heuristic from the current node to the goal can be estimated. In this paper, the measure is not distance, but total delay. The A* algorithm has its advantages, because it is known to visit the fewest number of graph nodes for a given heuristic. As the results show, when this algorithm is used in traffic control, this translates to reduced computation time with no noticeable reduction in delay performance.

Some other techniques are also employed to reduce the computation time. Since computational complexity grows with queue size, it makes sense to reduce the size of each queue, by clustering vehicles together into platoons. The effect of clustering on delay is

examined in this paper up to a practical limit. This provides evidence that clustering does not significantly increase delays, while reducing computational requirements.

Even if clustering is not performed, MPC traffic control is not strictly optimal because a prediction horizon limits the look-ahead time or, alternatively, the look-behind distance. The effect of the prediction horizon on delay is examined for both undersaturated and oversaturated conditions.

2. Literature Review

Traditional signal control was performed offline. Traffic engineers utilised traffic data collected throughout the hours of each day and used this data to devise a signal timing strategy. This resulted in the operating parameters *cycle time*, *splits* and *offsets*. These parameters were generally derived from standard procedures, such as those described in the Highway Capacity Manual (Transportation research board, 2000) or signal-timing software such as TRANSYT (Robertson, 1969) and SYNCHRO (Husch and Albeck, 2006).

More modern traffic control systems are able to react to current traffic control conditions by sensing the presence of vehicles using inductive loops in the road or with alternative technologies such as traffic cameras. However, they still often assume that for the next several minutes or hours, traffic conditions can be accurately characterized by summarising averages. Control strategies that operate in this fashion include SCOOT (Robertson and Bretherton, 1991) and SCATS (Sims and Dobinson, 1980). Alternatively, traffic patterns that constantly vary may be better characterised as a discrete sequence of platoons of vehicles rather than approximated as a continuous flow.

Some researchers have developed an alternative approach to signal control that predicts, ahead of time, what the traffic pattern will be as it approaches the intersection and determines the control schedule based on this information. This is a form of model predictive control (MPC). In contrast to earlier traffic control systems that calculate average flows over time, this MPC approach incorporates a feedforward mechanism instead of a feedback mechanism, because it predicts traffic movements before they arrive. Such control systems include OPAC (Gartner et al., 2001), PRODYN (Henry et al., 1984), UTOPIA (Mauro and Di Taranto, 1990) and RHODES (Mirchandani and Head, 2001). More recently, Xie et al. (2012) devised an MPC traffic control algorithm that was formulated as a job shop scheduling problem. They clustered vehicles into platoons and used elimination criteria to reduce the search state space.

Since traffic signals change in a discrete manner, the problem of finding phase splits naturally becomes combinatorial, requiring exponential computation time to find an optimal solution (Papageorgiou et al., 2003). Therefore, some authors have employed a store-and-forward traffic model and applied linear-quadratic regulator theory (Diakaki et al., 2002) or quadratic programming (Aboudolas et al., 2010; Le et al., 2013) to devise a tractable control strategy. These methods assume average flow rates to efficiently obtain phase splits. However, they react to current traffic conditions instead of predicting future traffic movements. Also, the underlying traffic model of these methods is inherently reduced to simple flow rates and turning ratios, instead of a full-fledged microscopic model.

Other authors have formulated traffic control as a mixed integer linear programming (MILP) problem. Lo (1999) used a hydrodynamic (macroscopic) model, which captures kinematic waves. The timing plans produced were consistent with models that work for unsaturated conditions. In gridlock conditions, the produced timing plans were better than conventional queue management practices. Several other authors have formulated their own MILP traffic control schemes including Lin and Wang (2004) and Beard and Ziliaskopoulos (2006). Han et al. (2012) formulated a MILP problem using a link-based model instead of using the Cell Transmission Model (Daganzo, 1994). A variational type argument was applied so that the system dynamics can be determined without knowledge of the traffic state in the interior of each link. This resulted in a reduced number of binary variables and computational effort.

Guilliard et al. (2016) formulated their model using non-homogenous time intervals, resulting in lower delay solutions.

3. Overview of Model Predictive Traffic Control

3.1. Network Definitions

An intersection, I, is composed of a set of routes R. Each route $r \in R$ is a pair of entry and exit links, that is, route $r = (e, x)$ and $e \in E$, $x \in X$, where E is the set of entry links (approaches) and X is the set of exit links.

The set of all vehicles to be scheduled is V, and each vehicle belongs to a particular route. Vehicles on a route can be referred to by indexing the route, $r_i \in V$. Naturally, each vehicle has motion variables associated with it. For $c \in V$,

$x(c)$ refers to the vehicle's location along its route

$v(c)$ refers to the vehicle's speed

$a(c)$ refers to the vehicle's acceleration

$A(c)$ refers to the vehicle's maximum acceleration

$L(c)$ refers to the vehicle's length

3.2. Algorithm Description

Model Predictive Control is a method from process control that has the goal of finding an optimal control sequence. It works by simulating the effect of control decisions into the future. Time is treated as a discrete quantity and at each time step new control decisions are made that have a corresponding effect on the simulated output quantities. Once some or all possible control sequences into the future have been simulated, the sequence producing optimal (that is, maximum or minimum) output is chosen to be applied into the future. However, at the next time step the process is repeated and a new optimal control sequence is selected, implying that only the first step of each control sequence is actually applied before being superseded by a subsequent choice. This scheme is called a rolling horizon.

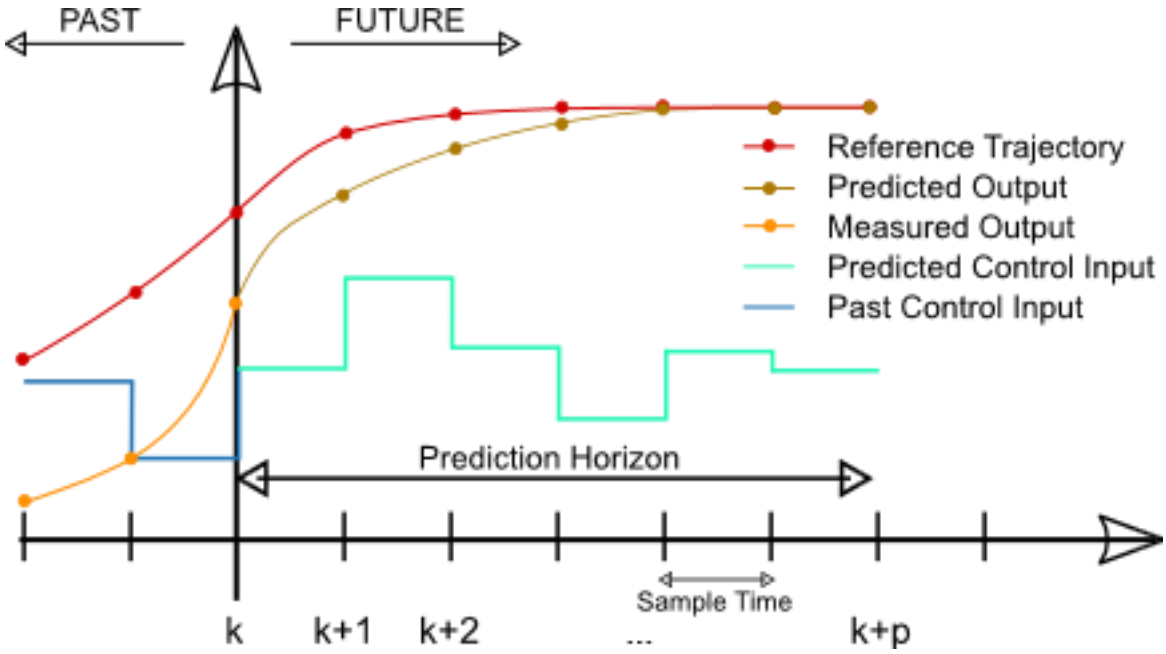
In the context of traffic control, each control sequence is a sequence of phases of determined lengths, allowing all vehicles within the prediction horizon to be given right of way and pass through the intersection. Conceptually, all possible control sequences can be generated by permuting all vehicles on all approaches, while preserving vehicle ordering on each approach (assuming no overtaking occurs). If vehicles are given right of way one at a time, the total number of sequences is given by

$$|S| = \frac{(\sum_{e \in E} |e|)!}{\prod_{e \in E} (|e|)!} \tag{1}$$

where S is the set of all control sequences and $|e|$ is the number of vehicles on approach e.

Since $|S|$ grows at least exponentially it is not considered computationally tractable to evaluate every sequence in order to find the optimal choice. Besides having a finite prediction horizon, a few techniques can be employed to reduce computational complexity.

Figure 1. Model predictive control. Courtesy of Martin Behrendt ©.



Instead of permuting individual vehicles, they can be clustered together to form platoons. The permutation unit then becomes a cluster of vehicles and the number of units per approach is reduced, which in turn results in fewer permutations. A vehicle is considered to have joined a platoon if it is required to reduce speed due to car-following constraints.

The second technique for reducing computational complexity involves eliminating partial schedules that are very unlikely to result in the optimal solution. This technique, proposed by Xie et al. (2012), involves comparing similar partial schedules where the same set of vehicles are yet to enter the intersection in both partial schedules. If one partial schedule has greater cumulative delay and accumulated time compared to the other, it, and all of its subsequent schedules, are eliminated from consideration. Xie et al. (2012) were able to show that eliminating schedules in this manner reduces the number of schedules to evaluate to a polynomial, which makes it a more tractable problem.

In this paper, we propose using the A* search algorithm instead of elimination criteria. Testing indicates that computation time is reduced as a result.

4. Overview of A* Algorithm

The A* search algorithm was developed by Hart et al. (1968), and its original application was finding shortest paths in a network. However, it has general applicability to graph or network problems that require optimal goal nodes to be found. The algorithm maintains an ordered list of graph/network nodes. At each step, neighbouring nodes of the first node in the list are added to the list, and the first node is removed from the list. Before being added, each node is evaluated by estimating its total cost from the origin node to a goal node. This estimate is calculated according to the formula

$$f(n) = g(n) + h(n) \tag{2}$$

where

n is the current node being evaluated

f is the total estimated cost

g is the known cost from the origin node to the current node

h is a heuristic that estimates the remaining cost from the current node to the goal node

The algorithm orders nodes into a list according to their f values. In order to guarantee optimality, the heuristic function, h, must be admissible, meaning that it doesn't overestimate remaining cost. The creators of the A* algorithm proved that it evaluates the fewest number of nodes in the search graph compared to any other admissible algorithm utilising the same heuristic. In this sense, its running time is the minimum required.

5. Traffic Control Implementation

In the traffic control case, the "cost" we wish to minimise is delay.

- $f(n)$ can be calculated by assuming:
  - each vehicle will accelerate at some maximum rate until it reaches its desired speed; or
  - it is restricted by the vehicle in front and its delay is dependent on that vehicle's final free flow motion
  - $f(n)$ can be progressively calculated for all vehicles, one after the other, in a queue
  - cross traffic is not given explicit right of way to calculate the admissible heuristic

There are essentially two ways to calculate a delay heuristic. For an intersection with simple routing, a function can estimate total delay, $f(n)$. An algorithm for calculating $f(n)$ is shown in Listing 1. This algorithm assumes that vehicle ordering in a queue doesn't change with time.

Each node in the search tree represents a partial schedule, that is, an ordered sequence of vehicles to be given right of way, taken from the set of vehicles within the prediction horizon. Each partial sequence can be simulated from the current time until the last vehicle in the sequence has entered the intersection, at which time a decision needs to be made to extend or change phases. At this time, $f(n)$ can be calculated for all vehicles in the current set (all vehicles within the prediction horizon) and assigned to the current node. Once a node has been evaluated it is added to the algorithm's ordered list, and the first node in the list is the next to be expanded, that is, its children are evaluated next. Its children are represented by the next choice of phase.

The simulator often needs to be reset to the current time and re-run when a new node in the ordered list is chosen to be the current node.

6. Incremental A* Algorithm

Reductions in computation time can be achieved with an incremental version of the A* algorithm. For the original algorithm, every time a vehicle crosses the prediction horizon, the scheduling algorithm is executed in its entirety. However, it is possible to, instead, re-use information from the previous execution to save processing time when a new vehicle crosses the prediction horizon. In this incremental version, the previous execution's state space tree is preserved and updated for the current execution. The delay estimate for each node in the state space tree is updated without performing an entire recalculation.

These heuristic updates can be calculated by considering how the system has changed in between executions of the scheduling algorithm. Firstly, some vehicles have exited the intersection. These vehicles no longer need to be scheduled by the control algorithm, so their estimated delay needs to be deducted from each node's f score. Since they have exited the intersection, their delay is not influenced by changes to the control schedule and should be constant for all subsequent states in the state space tree. In addition, at least one vehicle crossed the prediction horizon since the previous execution, and the estimated delays for these new vehicles need to be added to each node's total delay estimate.

Listing 1. An admissible algorithm for estimating total delay.

```
1: function CalculateDelay
2:     Constants:
3:         TimeOffset                          ▷ time into the future e.g. 600 s
4:         h                                   ▷ desired headway time between vehicles
5:         s                                   ▷ standstill distance between vehicles
6:
7:     Global Properties:
8:         t                                   ▷ simulator's current time
9:         v_L(r ∈ R)                          ▷ speed limit for the route r at location x
10:        t0(c ∈ V)                           ▷ time vehicle c enters system
11:        x0(c ∈ V)                           ▷ location vehicle c enters system
12:        length(c ∈ V)                       ▷ length of vehicle c
13:
14:    t_end ← t + TimeOffset
15:    delay_total ← 0
16:    for r ∈ R do
17:        r_prev ← ∞
18:        for i ∈ [1..|r|] do
19:            c ← r_i                         ▷ c is vehicle i on exit link x
20:            dt ← (v_L(r) − v(c))/A(c)        ▷ dt is time required to accelerate to the vehicle's desired speed
21:            t_0 ← t + dt
22:            x_0 ← x(c) + v(c)dt + ½A(c)dt²   ▷ x_0 is location of vehicle when it reaches desired speed
23:            x_acc ← x_0 + (t_end − t_0)v_L(r)   ▷ x_acc is location of vehicle at time, t_end, if it's unimpeded
24:            x_follow ← x_prev − hv_L(r) − s   ▷ x_follow is location of vehicle if it's impeded by previous vehicle
25:            x_real ← min(x_acc, x_follow)
26:
27:            Δt ← t_end − t0(c)
28:            Δx ← x_real − x0(c)
29:            delay = Δt − Δx/v_L(r)
30:            delay_total ← delay_total + delay
31:
32:            x_prev ← x_real − length(c)
33:        end for
34:    end for
35:    return delay_total
36: end function
```

CALCULATEDELAY in listing 1 calculates an estimate of the final total delay for all vehicles to be scheduled. Instead of computing a single value, this function can be modified to calculate two extra delay sums – one for vehicles that have exited the intersection (D_x), and another for new vehicles that have crossed the prediction horizon since the previous execution of the scheduling algorithm (D_n). Once these values are calculated, each node's new delay estimate, $f'(n)$, can be computed:

$$\Delta D = D_x - D_n \tag{3}$$

$$f'(n) = f(n) - \Delta D \tag{4}$$

The function, UPDATEDELAYESTIMATE is run for each node in the state space tree before running the A* algorithm. If the node has no child states, that is, if it is a terminal state it is updated directly. Otherwise, the function calculates the new total delay estimate for each child state and chooses the minimum as the new delay estimate for the current state.

After updating all delay estimates in the state space tree, the A* algorithm can be run as per usual. The only difference being that when a new delay estimate, $F(n)$, is calculated for node n, it is not used as the updated delay estimate unless it is greater than the existing delay estimate, that is, $\big(F(n) > f(n)\big) \rightarrow \big(f'(n) = F(n)\big)$. Because delay estimates are updated before the A* algorithm is run, this changes the ordering of nodes in the ordered list of nodes. This results in a faster running time for the algorithm because the optimal path in the search tree is found with less computation time.

Listing 2. Each node in the state space tree is updated recursively with UPDATEDELAYESTIMATE in the incremental version of the scheduling algorithm.

```
1: function UpdateDelayEstimate(state, ΔD)
2:     if children(state) = ∅ then
3:         f(state) ← f(state) − ΔD                                   ▷ Update f(state).
4:     else
5:         for child ∈ children(state) do
6:             UpdateDelayEstimate(child, ΔD)   ▷ First, update delay estimates for all child states.
7:         end for
8:         f(state) ← min{child ∈ children(state) | f(child)}   ▷ Set f(state) to minimum child f.
9:     end if
10: end function
```

7. Other Factors

7.1. Prediction Horizon

Model predictive control requires simulating alternative control schedules into the future. However, these predictions must terminate at some length of time in the future from the current time. This is called the prediction horizon and since it is finite, it prevents MPC from being truly optimal in all circumstances. However, it is possible for a finite prediction horizon to produce truly optimal results if the queue lengths on all approaches are less than the prediction horizon (for all examined potential future schedules). As the results demonstrate, not much reduction in delay is obtained by increasing the prediction horizon beyond a reasonable length, even in oversaturated conditions.

7.2. Clustering

Clustering vehicles into platoons is performed to reduce computation time. The exact method varies depending on the underlying car-following model, but the model can determine when a vehicle joins a platoon. If clustering is performed, the number of states in the state space is reduced because there are fewer platoons than individual vehicles in the system. Generally, there is a very small delay penalty for clustering, because it is inefficient to split vehicles in the same platoon by giving them right of way in separate phases.

8. Results

The traffic control algorithm requires that it be possible to revert the simulation to a previous time. Also, since there are many traffic states to examine for each time step, it is also a requirement that the simulator run much faster than real-time. Unfortunately, it is hard to achieve these requirements with current, commercial microscopic traffic simulation software. Therefore, a custom traffic simulator was created. Since lane changing and turning are not trivial to implement and not absolutely required to test the feasibility of the traffic control algorithm, they were not implemented.

The machine used for testing was a general purpose PC (CPU: Intel Core i7-4770, RAM: 16 GB). Although the A* algorithm typically examines hundreds of traffic states every time it runs, the Web browser running the algorithm used < 300 MB of RAM. For a modern PC this is a

relatively small fraction of the total amount available. For this reason, memory usage is not included in the results.

The following results were obtained in a custom, Web-based simulator. All code was written in JavaScript and run in the Google Chrome Web browser. The intersection was composed of two one-way roads, with no turning allowed. Arrivals occurred according to an exponential distribution. Testing occurred over a minimum of one hour of simulated time. Simulation continued until all vehicles that had arrived within the hour had exited the system. Each test was conducted 10 times to determine a 90% confidence interval by applying a Student's t-test, shown as error bars in each figure. For each figure, tests were performed for a range of average flow rates. MPC methods were tested with four state reducing techniques – A* algorithm, elimination criteria, A* algorithm with vehicle clustering into platoons, and elimination criteria with vehicle clustering into platoons. It was not computationally feasible to run the MPC methods without clustering at rates higher than 600 vehicles/hr, because it would not operate in real-time.

Figure 2. Average delay for various control algorithms.

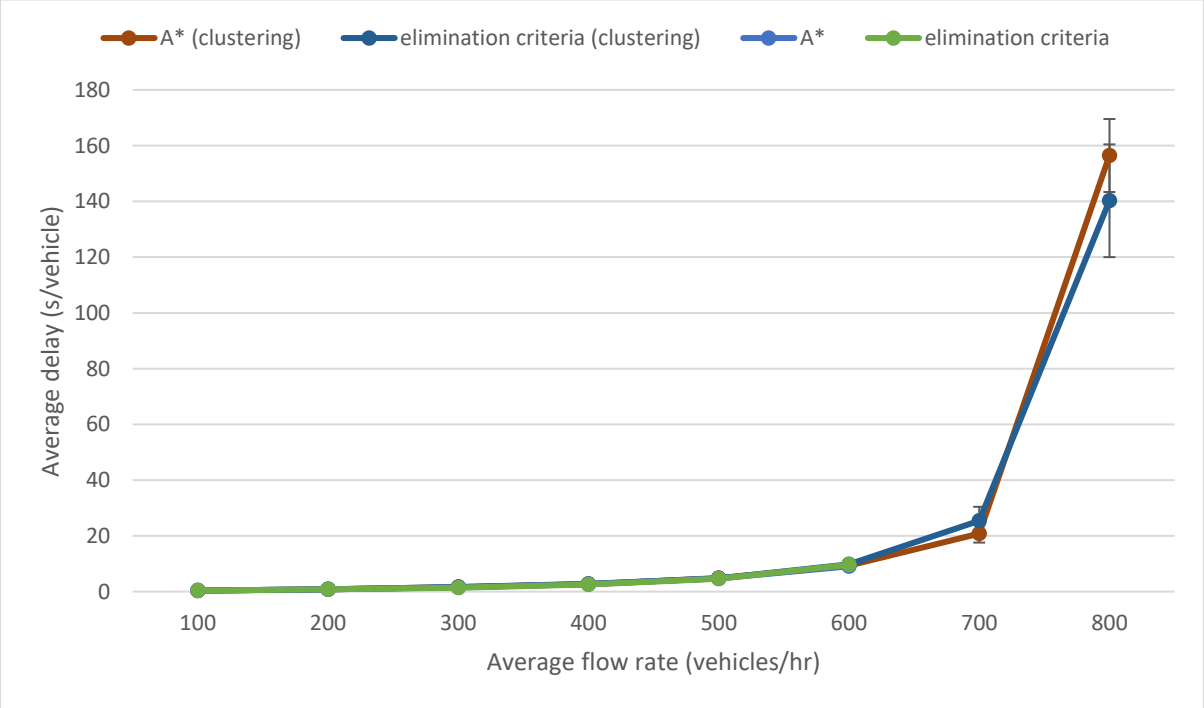


Figure 2 illustrates average vehicle delay over a range of flow rates. As can be seen, there is no significant difference in average delay between the A* algorithm and elimination criteria methods, and therefore there is no significant disadvantage to using the A* algorithm. Likewise, there is virtually no difference in average delay between the clustered and non-clustered versions. This indicates that treating individual vehicles separately, though computationally more expensive, does not result in noticeably reduced delay.

Without clustering vehicles into platoons, the performance of the MPC methods drops significantly when the arrival rate is 700 vehicles/hr. It would require significantly longer than 1 hour of processing time to compute the signal timings for one hour of simulated traffic at this arrival rate. Since this implies real-time operation is impractical at rates higher than 600 vehicles/hr without clustering vehicles into platoons, and due to the amount of time required to collect results, testing was not completed for these high flow rates.

Figure 3. Average CPU time per iteration for various control algorithms.

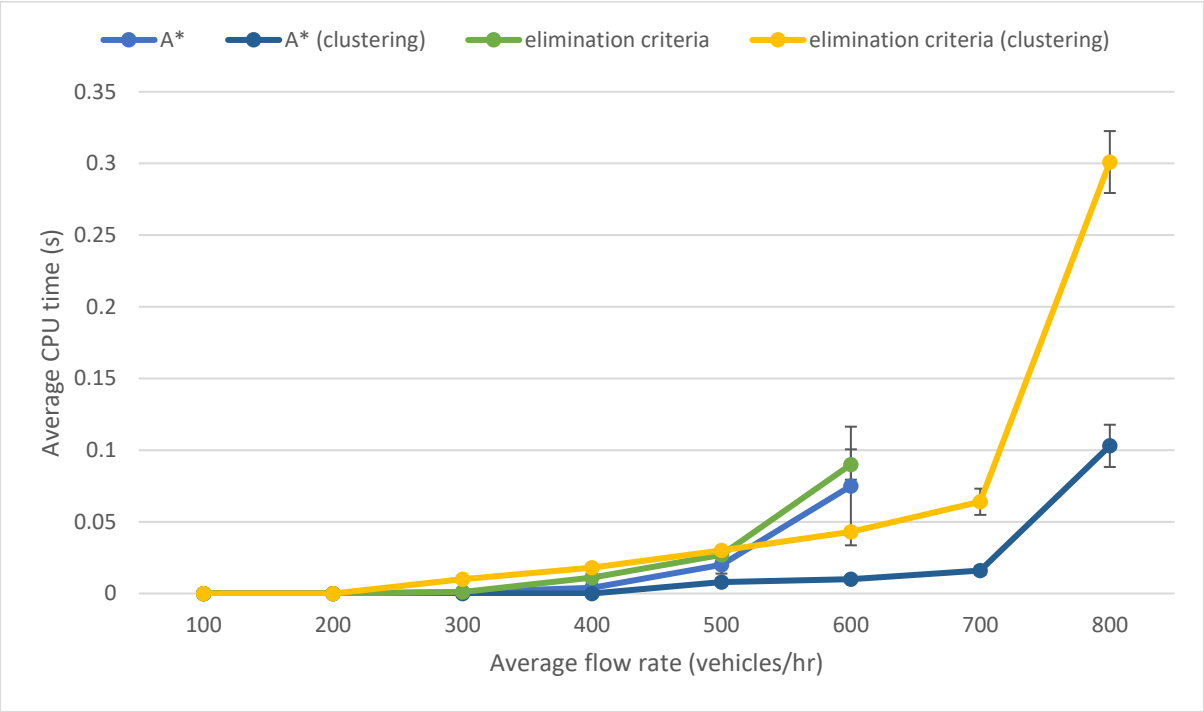


Figure 3 shows how the average CPU time per iteration of the scheduling algorithm increases with average flow rate. Without clustering, the A* search space reduction technique outperforms the use of elimination criteria, although the difference in CPU time is not large. At rates higher than 600 vehicles/hr, processing time made the non-clustered scheduling algorithms infeasible for real-time operation. With clustering enabled, CPU time is significantly reduced when using the A* algorithm instead of elimination criteria. The difference becomes even more pronounced at flow rates higher than the saturation flow rate (~ 650 vehicles/hr). However, even at these flow rates, average CPU time is acceptable. Since signal phase lengths are not usually adjusted by less than 1 second it is ideal if the scheduling algorithm execution time is < 0.5 seconds.

Figure 4 is similar to Figure 3, except that it illustrates how maximum CPU time per iteration increases with average flow rate. Maximum CPU time is actually greater for the non-clustered version of the A* algorithm compared to the algorithm using elimination criteria, but the difference is not significant for flow rates up to 600 vehicles/hr. However, the A* algorithm is significantly better than using elimination criteria when clustering is enabled. At flow rates greater than the saturation flow rate, maximum CPU time starts to make the use of elimination criteria impractical for real-time operation. However, the A* algorithm version performs much better as the maximum CPU time is almost always under 2 seconds, even for the highest tested flow rate.

Figures 5-7 compare the performance of the incremental and the original, non-incremental versions of the A* algorithm. In both cases, vehicles are clustered into platoons. There is no statistically significant difference in average delay between the two methods, indicating that there is no penalty in using the incremental version. However, average CPU time is significantly reduced by using the incremental version at higher flow rates. Maximum CPU time appears to be lower for the incremental version, but the difference between the two methods is not statistically significant.

Figure 4. Maximum CPU time of all iterations for various control algorithms.

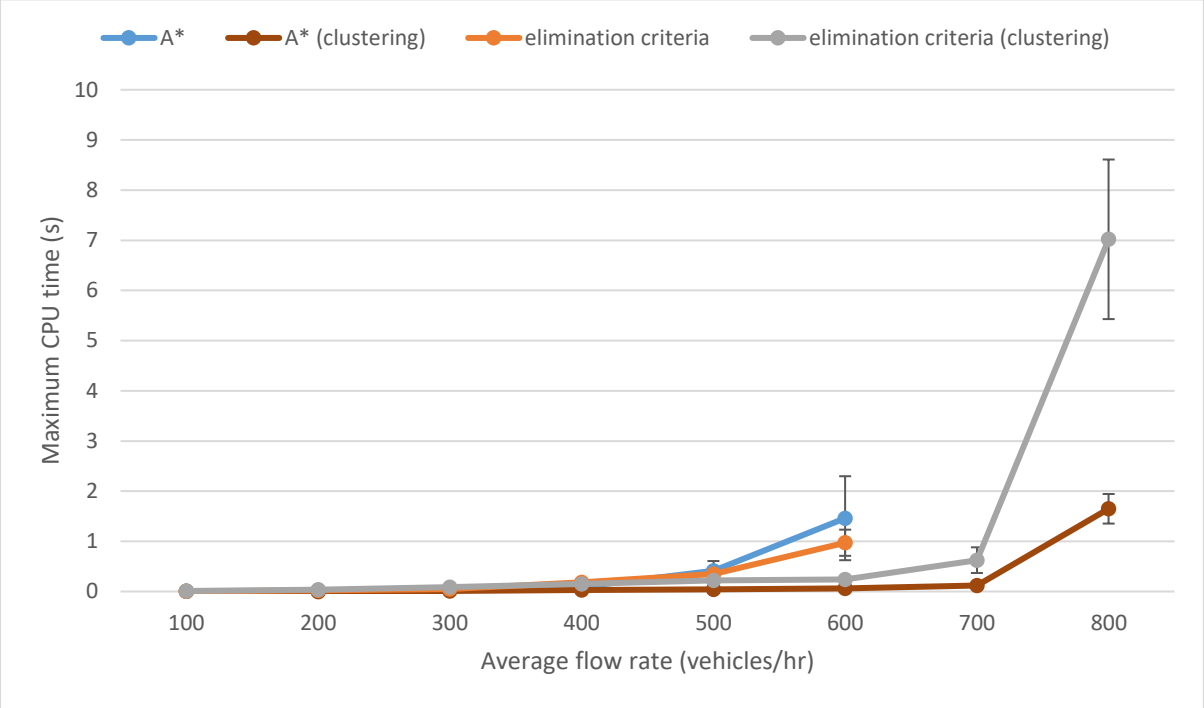


Figure 5. Average delay for the incremental and non-incremental versions of the A* algorithm.

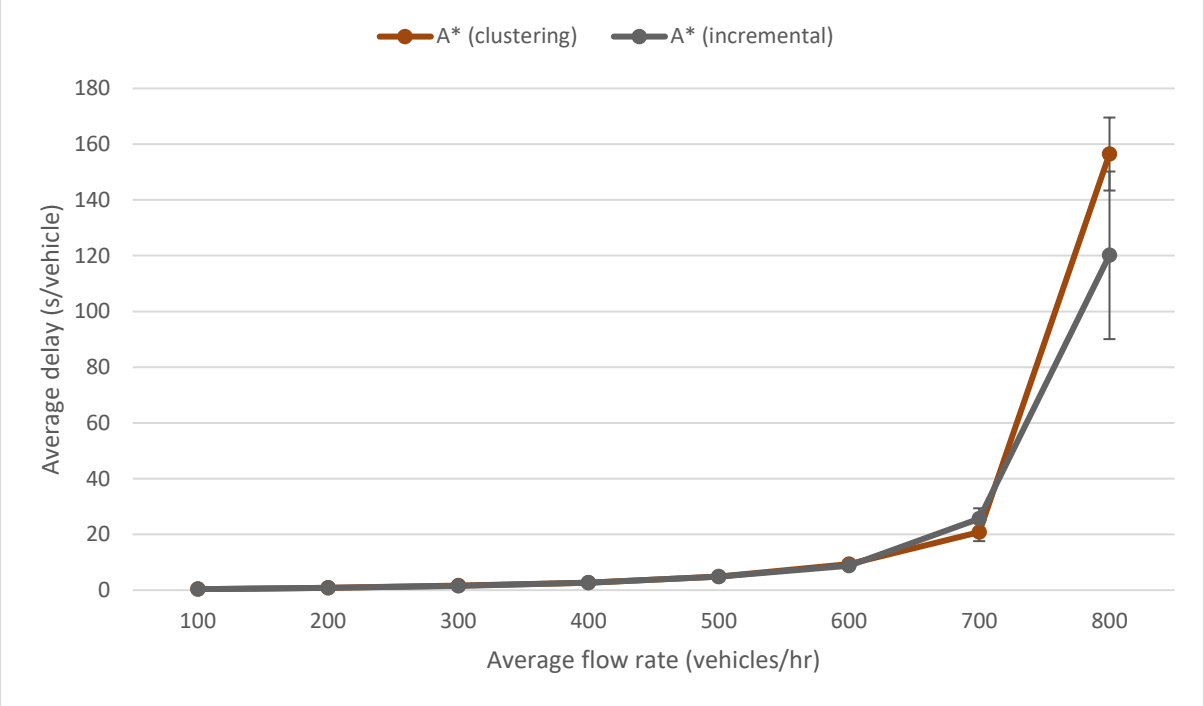


Figure 6. Average CPU time per iteration for the incremental and non-incremental versions of the A* algorithm.

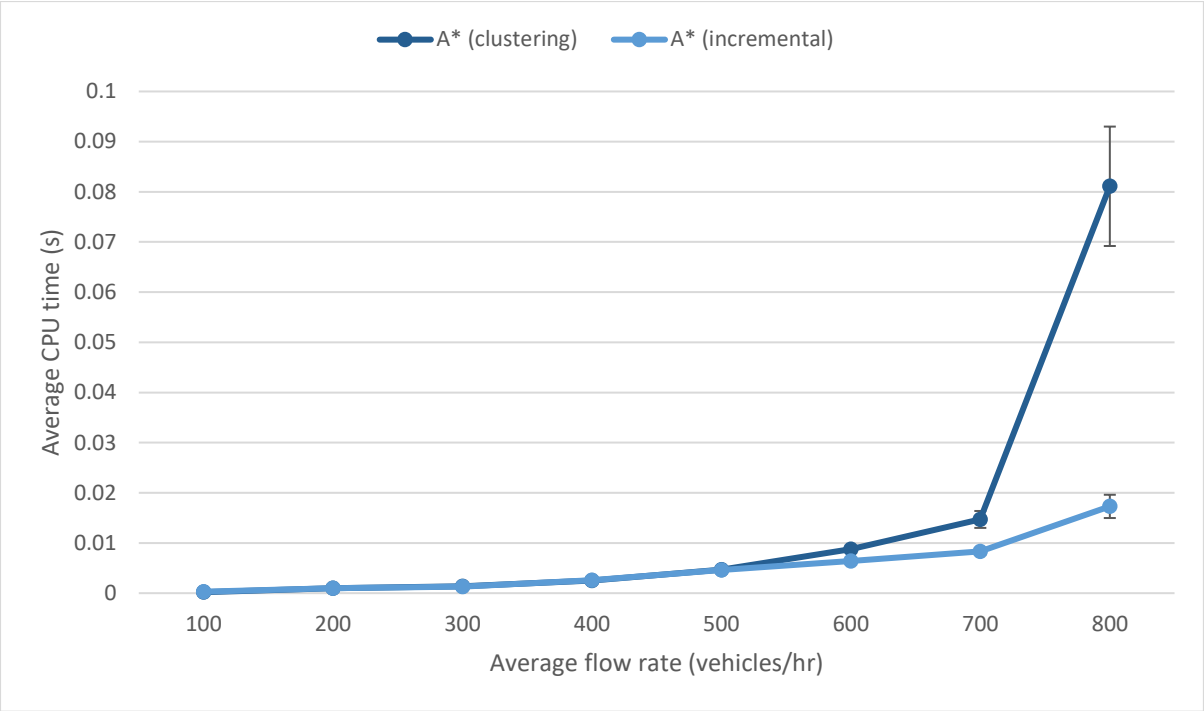
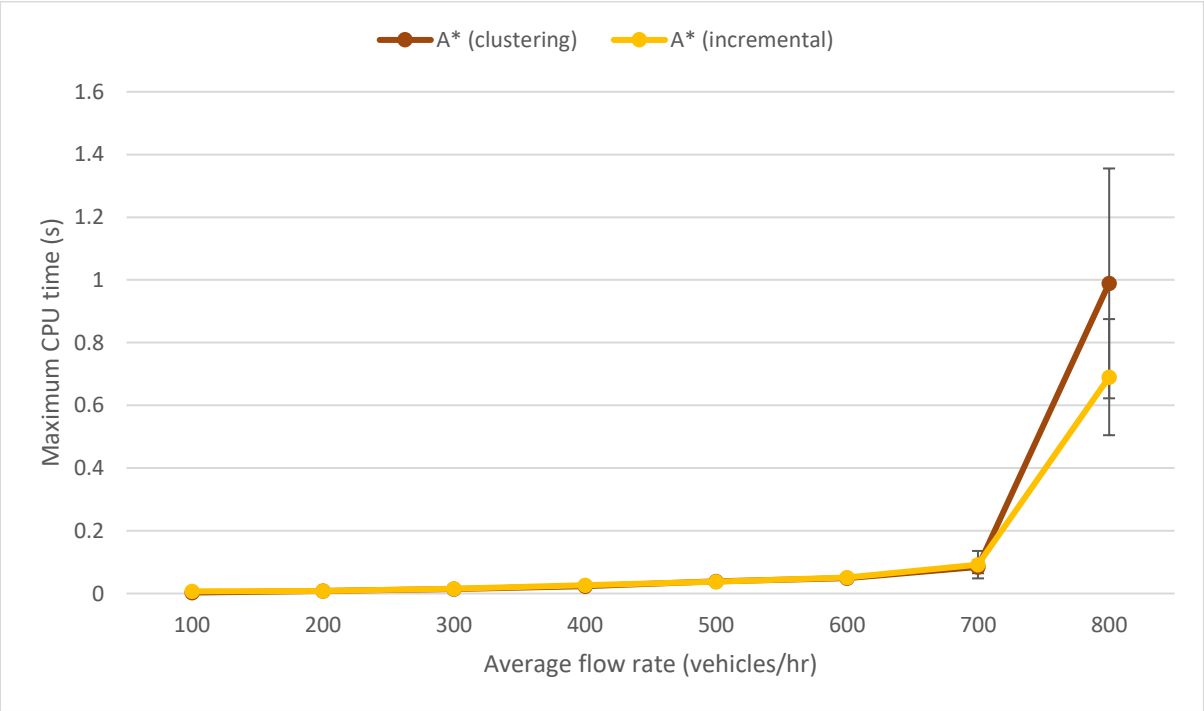


Figure 7. Maximum CPU time of all iterations for the incremental and non-incremental versions of the A* algorithm.



The following figures (8-13) show how queue size and length change over time for undersaturated and saturated conditions. The same randomisation seed was used in all cases so that comparisons can be made consistently across all cases.

Figures 8 and 9 illustrate how queue size and distance, respectively, changed over time when the flow rate was 600 vehicles/h and conditions undersaturated. There is no clear phase

switching pattern and the queue lengths appear to vary randomly, just as the arrival pattern is random. Rarely, over the course of an hour, do the queue lengths exceed the prediction horizon. This means that, most of the time, all vehicles are scheduled by the control algorithm, and any remaining vehicles, beyond the queued vehicles, are undelayed. This guarantees an optimal solution, despite having a finite prediction horizon. Even though occasionally a vehicle beyond the prediction horizon joins a queue, there is no obvious increasing trend in the length of the queues at this average flow rate.

Figure 8. The number of vehicles in each queue over time when the average flow rate on both queues was 600 veh/h.

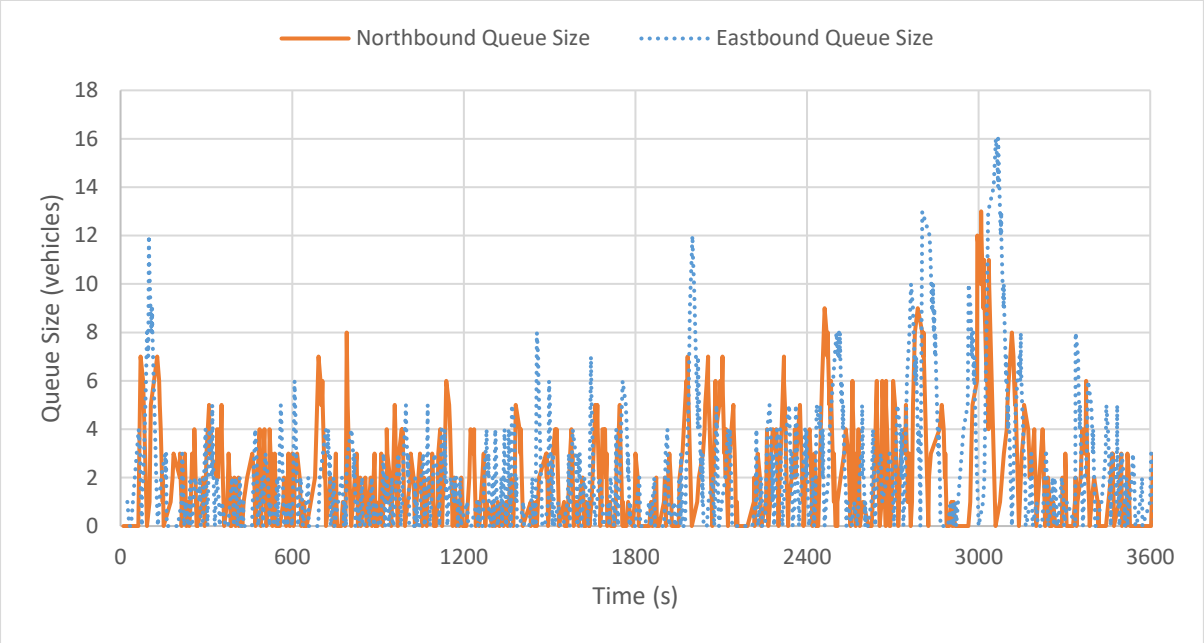


Figure 9. The length of each queue over time when the average flow rate was 600 veh/h.

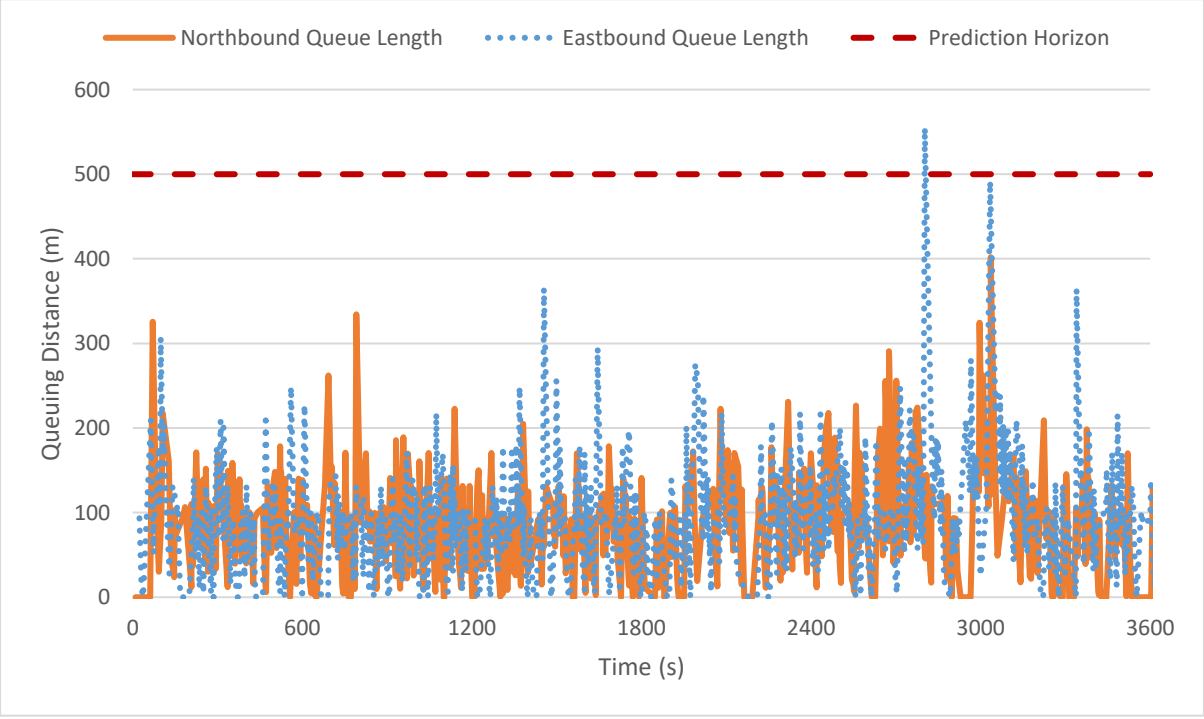


Figure 10. The number of vehicles in each queue over time when the average flow rate on both queues was 800 veh/h.

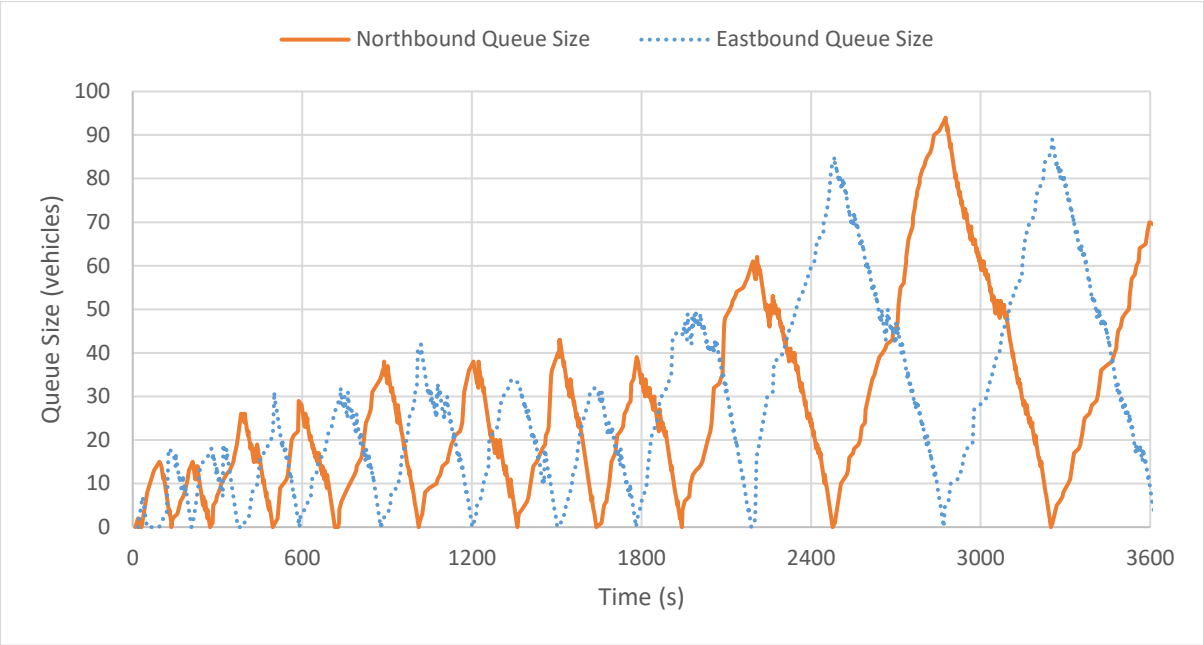


Figure 11. The length of each queue over time when the average flow rate was 800 veh/h.

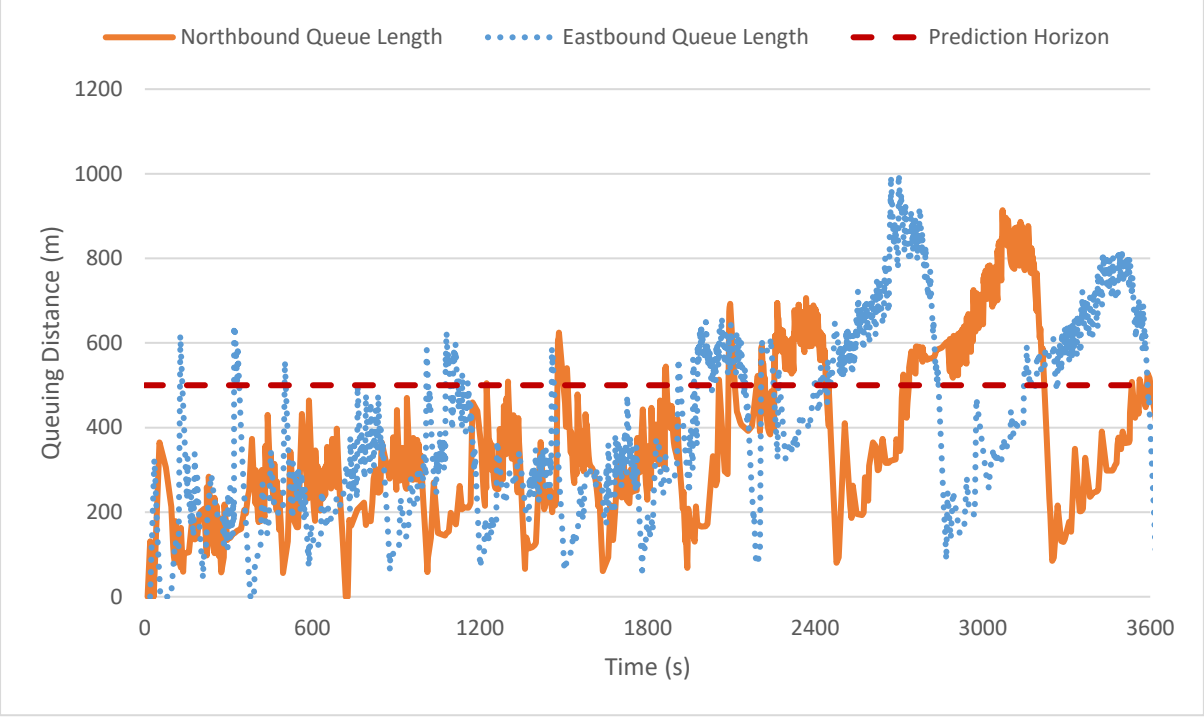
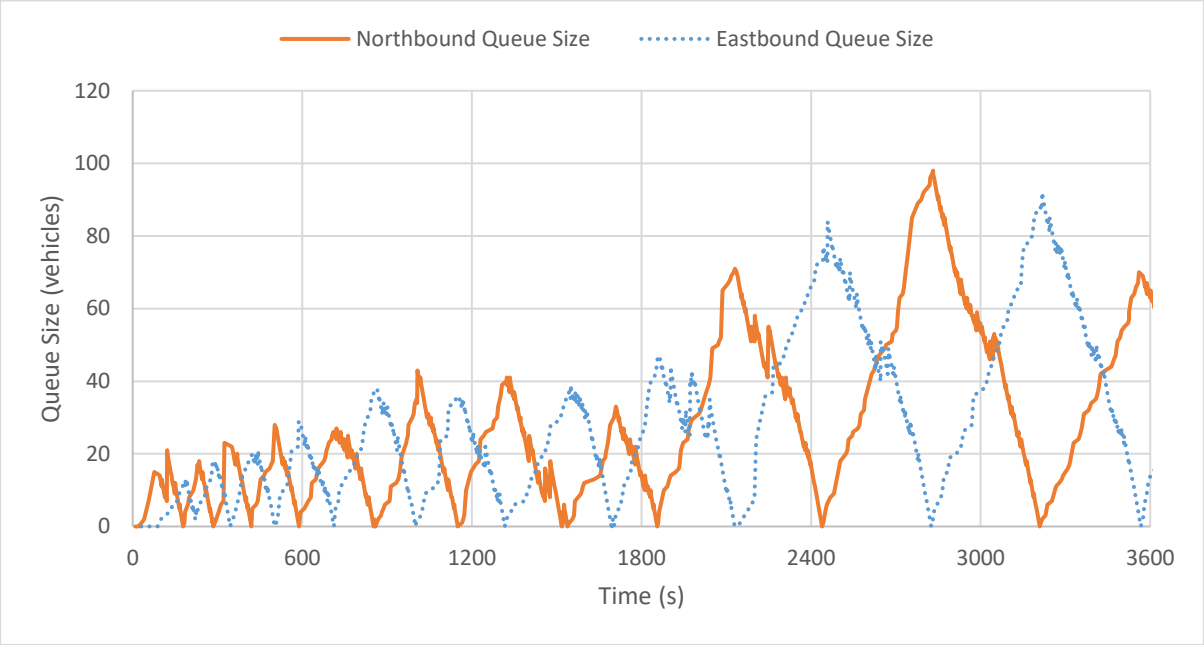


Figure 10 and Figure 11 illustrate how queue size and distance respectively changed over time when the average flow rate was 800 vehicles/h. Queuing increases over time, so the conditions are saturated. At this flow rate, the optimal solutions require phase switching to become less and less frequent. However, the cycle times becomes longer and longer, essentially allowing a queue to empty before switching right of way. In fact, these cycle times are so long they are probably impractical for realistic use, even though they result in system

optimal minimum delay. The queue distances do not match up with the queue sizes plots, because a queue continues to grow in size after the approach is given right of way as new vehicles are added to the end of the queue. However, eventually the queue is depleted and right of way is switched to the alternate approach. At this higher flow rate, the calculated solutions require the phase switching pattern is be much more regular and predictable. However, the queue lengths are frequently beyond the prediction horizon (500 m). Therefore, optimal solutions are not guaranteed.

Figure 12. The size of each queue when the average flow rate was 800 veh/h and the prediction horizon 1000 m before the intersection.



Figures 12 and 13 were obtained with the same average flow rate of 800 veh/h. However, the prediction horizon was increased to 1000 m. Compared to figures 3 and 4, the resultant graphs appear similar. The statistics on Table 1 show that as the prediction horizon was increased, average delay decreased slightly. However, it decreased by only 2%, even though the prediction horizon doubled. Computation time was much greater with a longer prediction horizon, indicating that the benefits of a longer prediction horizon do not outweigh the computation costs, even though, with a longer prediction horizon, optimal scheduling solutions are guaranteed until the queues grows to a longer length.

These results are concordant with early research on oversaturated intersection control. Gazis (1964), and Michalopoulos and Stephanopoulos (1977) proved that, assuming a constant flow rate, the optimal control solution tends toward the maximum and minimum phase lengths. This implies that control schedules may be much more predictable in oversaturated conditions, compared to undersaturated conditions, as these results demonstrate.

Figure 13. The length of each queue when the average flow rate was 800 veh/h and the prediction horizon was 1000 m before the intersection.

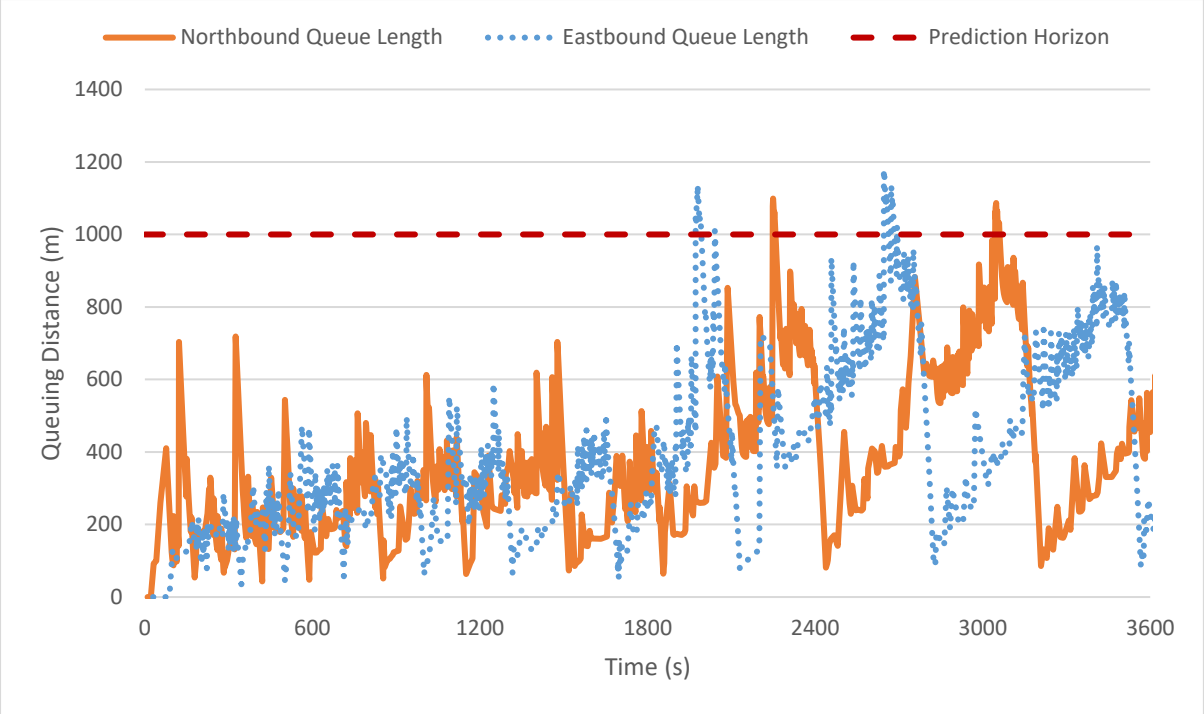


Table 1. Statistics when the average flow rate was 800 veh/h.

Statistics	Prediction Horizon: 500 m	Prediction Horizon: 1000 m
Average Vehicle Delay (s)	115.8	113.2
Average Computation Time (s/iteration)	0.075	0.263
Maximum Computation Time (s/iteration)	0.876	2.558

9. Conclusion

Effective, real-time traffic control requires a balance between algorithmic performance and efficient processing. For model predictive traffic control, various strategies can be employed to manage computational requirements without significantly increasing total delay and sacrificing performance. The A* algorithm was applied to find optimal solutions efficiently. Computation time required was reduced, compared to another recently proposed control algorithm that applies elimination criteria to reduce the size of the state space. Average CPU time was further reduced when an incremental A* algorithm was tested.

Up to 600 vehicles/hr, there was no noticeable difference to average delay when vehicles were clustered into platoons, even though computation time was greatly reduced, implying that it is a sound strategy to cluster vehicles. For undersaturated conditions, the prediction horizon did not need to be longer than 500 m, but for oversaturated conditions, a longer prediction horizon was needed to ensure optimal results. However, since queue lengths grow over time in oversaturated conditions, no fixed prediction horizon can guarantee optimal results if the flow rate is oversaturated for long periods of time. Moreover, there was little reduction in average delay by doubling the prediction horizon, probably because optimal control schedules are more predictable in oversaturated conditions.

Building upon the current work, the authors are investigating how to integrate the control algorithm in VISSIM, a commercial traffic simulator. However, this is a challenge since commercial traffic simulators are not typically designed to allow reversion to previous states and do not necessarily run much faster than the real-time rate.

In order to test the control algorithm in a more realistic setting, future work may also compare the performance of the control algorithm with field data obtained from an intersection that operates in a real traffic network. In this case, new vehicles in the simulation would not be generated with an exponential distribution, but with the real arrival data.

References

Aboudolas, K., Papageorgiou, M., Kouvelas, A., Kosmatopoulos, E., 2010. A rolling-horizon quadratic-programming approach to the signal control problem in large-scale congested urban road networks. Transportation Research Part C: Emerging Technologies 18, 680–694. doi:10.1016/j.trc.2009.06.003

Beard, C., Ziliaskopoulos, A., 2006. System Optimal Signal Optimization Formulation. Transportation Research Record: Journal of the Transportation Research Board 1978, 102–112. doi:10.3141/1978-15

Daganzo, C.F., 1994. The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory. Transportation Research Part B: Methodological 28, 269–287. doi:10.1016/0191-2615(94)90002-7

Diakaki, C., Papageorgiou, M., Aboudolas, K., 2002. A multivariable regulator approach to traffic-responsive network-wide signal control. Control Engineering Practice 10, 183–195. doi:10.1016/S0967-0661(01)00121-6

Gartner, N.H., Pooran, F.J., Andrews, C.M., 2001. Implementation of the OPAC adaptive control strategy in a traffic signal network, in: 2001 IEEE Intelligent Transportation Systems, 2001. Proceedings. Presented at the 2001 IEEE Intelligent Transportation Systems, 2001. Proceedings, pp. 195–200. doi:10.1109/ITSC.2001.948655

Gazis, D.C., 1964. Optimum Control of a System of Oversaturated Intersections. Operations Research 12, 815–831.

Guilliard, I., Sanner, S., Trevizan, F.W., Williams, B.C., 2016. A Non-homogeneous Time Mixed Integer LP Formulation for Traffic Signal Control, in: Transportation Research Board 95th Annual Meeting.

Han, K., Friesz, T.L., Yao, T., 2012. A Link-based Mixed Integer LP Approach for Adaptive Traffic Signal Control. arXiv:1211.4625 [math].

Hart, P.E., Nilsson, N.J., Raphael, B., 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics 4, 100–107. doi:10.1109/TSSC.1968.300136

Henry, J.-J., Farges, J.L., Tuffal, J., 1984. The PRODYN real time traffic algorithm, in: IFACIFIPIFORS Conference on Control in.

Husch, D., Albeck, J., 2006. Synchro studio 7 user guide. Trafficware Ltd., Sugar Land, TX.

Le, T., Vu, H.L., Nazarathy, Y., Vo, B., Hoogendoorn, S., 2013. Linear-Quadratic Model Predictive Control for Urban Traffic Networks. Procedia - Social and Behavioral Sciences, 20th International Symposium on Transportation and Traffic Theory (ISTTT 2013) 80, 512–530. doi:10.1016/j.sbspro.2013.05.028

Lin, W.-H., Wang, C., 2004. An enhanced 0-1 mixed-integer LP formulation for traffic signal control. IEEE Transactions on Intelligent Transportation Systems 5, 238–245. doi:10.1109/TITS.2004.838217

Lo, H.K., 1999. A novel traffic signal control formulation. Transportation Research Part A: Policy and Practice 33, 433–448. doi:10.1016/S0965-8564(98)00049-4

Mauro, V., Di Taranto, C., 1990. Utopia. Control, computers, communications in transportation.

Michalopoulos, P.G., Stephanopoulos, G., 1977. Oversaturated signal systems with queue length constraints—I: Single intersection. Transportation Research 11, 413–421. doi:10.1016/0041-1647(77)90006-5

Mirchandani, P., Head, L., 2001. A real-time traffic signal control system: architecture, algorithms, and analysis. Transportation Research Part C: Emerging Technologies 9, 415–432. doi:10.1016/S0968-090X(00)00047-4

Papageorgiou, M., Diakaki, C., Dinopoulou, V., Kotsialos, A., Wang, Y., 2003. Review of road traffic control strategies. Proceedings of the IEEE 91, 2043–2067. doi:10.1109/JPROC.2003.819610

Robertson, D.I., 1969. " TRANSYT" method for area traffic control. Traffic Engineering & Control 11.

Robertson, D.I., Bretherton, R.D., 1991. Optimizing networks of traffic signals in real time-the SCOOT method. IEEE Transactions on Vehicular Technology 40, 11–15. doi:10.1109/25.69966

Sims, A.G., Dobinson, K.W., 1980. The Sydney coordinated adaptive traffic (SCAT) system philosophy and benefits. IEEE Transactions on Vehicular Technology 29, 130–137. doi:10.1109/T-VT.1980.23833

Stebbins, S., Hickman, M., Kim, J., Vu, H.L., 2017. Combining Model-Predictive Intersection Control with Green Light Optimal Speed Advisory in a Connected-Vehicle Environment. Presented at the Transportation Research Board 96th Annual Meeting.

Transportation research board, 2000. Highway Capacity Manual. National Research Council, Washington, DC, USA.

Xie, X.-F., Smith, S.F., Lu, L., Barlow, G.J., 2012. Schedule-driven intersection control. Transportation Research Part C: Emerging Technologies 24, 168–189. doi:10.1016/j.trc.2012.03.004