Enforcing Liveness in Autonomous Traffic Management

Tsz-Chiu Au
Dept. of Computer Science
The University of Texas at Austin
Austin, Texas 78712, U.S.A.
chiu@cs.utexas.edu

Neda Shahidi
Dept. of Electrical & Computer Engineering
The University of Texas at Austin
Austin, Texas 78712, U.S.A.
neda@mail.utexas.edu

Peter Stone
Dept. of Computer Science
The University of Texas at Austin
Austin, Texas 78712, U.S.A.
pstone@cs.utexas.edu

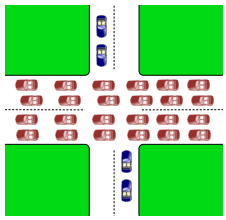
Abstract

Looking ahead to the time when autonomous cars will be common, Dresner and Stone proposed a multiagent systems-based intersection control protocol called *Autonomous Intersection Management* (AIM). They showed that by leveraging the capacities of autonomous vehicles it is possible to dramatically reduce the time wasted in traffic, and therefore also fuel consumption and air pollution. The proposed protocol, however, handles reservation requests one at a time and does not prioritize reservations according to their relative priorities and waiting times, causing potentially large inequalities in granting reservations. For example, at an intersection between a main street and an alley, vehicles from the alley can take an excessively long time to get reservations to enter the intersection, causing a waste of time and fuel. The same is true in a network of intersections, in which gridlock may occur and cause traffic congestion. In this paper, we introduce the batch processing of reservations in AIM to enforce liveness properties in intersections and analyze the conditions under which no vehicle will get stuck in traffic. Our experimental results show that our prioritizing schemes outperform previous intersection control protocols in unbalanced traffic.

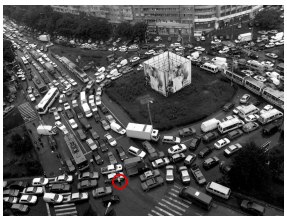
Introduction

Modern transportation is overly dependent on fossil fuel which is not only a finite resource but also a major source of greenhouse gas and air pollutants. Unfortunately, an ideal replacement for fossil fuel is not readily available yet. As demand for transport keeps increasing, an efficient transportation system is extremely important for the long-term sustainability of our society. Dresner and Stone (2008) proposed a novel intersection control mechanism called Autonomous Intersection Management (AIM), and in particular described a *First Come, First Served* (FCFS) policy to direct autonomous vehicles through an intersection. They showed that by leveraging the capacity of computerized driving systems FCFS significantly outperforms traditional traffic signals and stop signs, resulting in fuel savings since vehicles are less likely to stop and wait to enter intersections.

FCFS, however, fails to properly handle *unbalanced* traffic—the traffic on a main road is much heavier than the traffic on a crossing road—since vehicles from the crossing road can be blocked by the traffic on the main road. As shown in Figure 1(a), vehicles from the side road (the vertical direction) have difficulty in getting reservations to enter the intersection due to the heavy traffic on the main street (the horizontal direction). In the worst case, the vehicles from the side road will be denied from entering the intersection indefinitely, causing *starvation* (Dijkstra 1971).



(a) Unbalanced traffic



(b) Gridlock

Figure 1: Starvation due to unbalanced traffic and gridlock.

Unbalanced traffic is common as many intersections in cities are junctions connecting alleys or side roads to main streets. Traffic signals can smoothly and fairly handle this type of traffic but is an order of magnitude less efficient than FCFS (Dresner and Stone 2008). In this paper, we introduce a new intersection control policy called the *batch policy*, which is not only as efficient as FCFS but also able to prevent inequalities in granting reservations in unbalanced traffic. We further show that a modified version of the batch policy can enforce the *liveness* property of an intersection—every vehicle waiting at the intersection is guaranteed to enter the intersection eventually.

Starvation at one intersection may potentially develop into a network blockage called *gridlock* (Cervero 1986), like one in Figure 1(b). When gridlock occurs, the impact is no longer limited to one intersection, as many vehicles at different parts of the traffic network are involved. Gridlock occurs in all parts of the world. For instance, the "Great Chinese gridlock of 2010" in Hebei province, China, is considered the worst traffic jam in the history—the 60-mile jam lasted for 10 days. Thus it is very important that transportation systems guarantee no vehicle gets stuck in traffic and every vehicle eventually reaches its destination (the liveness property of transportation systems). Although the liveness of individual intersection controllers, as guaranteed by policies such

Copyright © 2011, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

as our batch policy, is necessary for preventing gridlock in a network, it is not by itself sufficient. We therefore analyze liveness properties in a road network and present the sufficient conditions for liveness of a simplified version of a road network. This analysis can shed light on what is needed to prevent starvation at the full network level.

Autonomous Intersection Management

The AIM protocol is based on a *reservation* paradigm, in which vehicles "call ahead" to reserve space-time in the intersection (Dresner and Stone 2008). The system assumes that computer programs called *driver agents* control the vehicles, while an arbiter agent called an *intersection manager* (IM) is placed at each intersection. The driver agents attempt to reserve a block of space-time in the intersection. The intersection manager decides whether to grant or reject requested reservations according to an *intersection control policy*. In brief, the paradigm proceeds as follows.

- An approaching vehicle announces its impending arrival to the IM. The vehicle indicates its predicted arrival time, velocity, acceleration, and arrival and departure lanes.
- The IM simulates the vehicle's path through the intersection, checking for conflicts with the paths of any previously processed vehicles.
- If there are no conflicts, the IM issues a reservation. It then becomes the vehicle's responsibility to arrive at, and travel through, the intersection as specified.
- The car may only enter the intersection once it has successfully obtained a reservation.

The prototype intersection control policy called *First Come, First Served* (FCFS) operates by dividing the intersection into a grid of *reservation tiles*. When a vehicle approaches the intersection, the intersection manager uses the data in the reservation request regarding the time and velocity of arrival, vehicle size, etc. to simulate the intended journey across the intersection. At each simulated time step, the policy determines which reservation tiles will be occupied by the vehicle. If the vehicle's space-time request has no conflict, the reservation is successful; otherwise, the reservation request will be rejected.

Empirical results in simulation demonstrated that the proposed reservation system with FCFS can dramatically improve the intersection efficiency when compared to traditional intersection control mechanisms (Dresner and Stone 2008). Overall, by allowing for much finer-grained coordination, the simulation-based reservation system can dramatically reduce per-car delay by two orders of magnitude in comparison to traffic signals and stop signs. This reduction of delays can translate into less traffic congestion (Au and Stone 2010; Quinlan et al. 2010), which in turn leads to better fuel efficiency and lower emissions.

One potential weakness of FCFS is that it handles reservation requests separately and does not take the history of vehicles' requests into account. Thus a car that is waiting on a side road may always find that a car from the main road has already requested a tile in the intersection that it needs. Though in balanced traffic, such a situation rarely occurs, in unbalanced traffic it occurs frequently. In the worst case, the cars on the side road are prevented from entering the main road indefinitely. A main contribution of this paper is a method for addressing this problem.

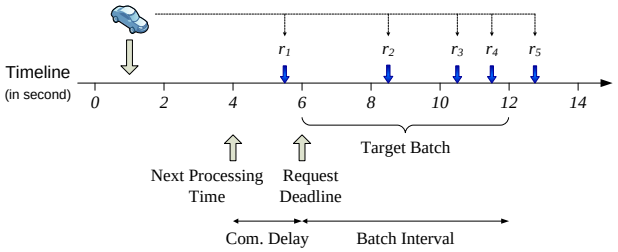


Figure 2: The batch policy.

Batch Processing of Requests

FCFS processes a request message immediately upon receiving it and sends out a confirmation message or a reject message in response to the request within a few milliseconds. While this approach can handle messages quickly with a small footprint in terms of system memory and computing power, it leaves little room for optimizing the ordering of requests and the allocation of resources. Thus, we propose a new intersection management policy called *batch policy* that puts the request messages on hold upon receiving them and then process several requests at once with a better ordering.

The central component of the batch policy is a sorted queue of request messages that acts as a buffer for temporarily storing the incoming request messages. As an example, suppose a vehicle sends a request message r at time 1 second, as shown in Figure 2. The request message contains 5 proposals, each of which is a tuple $r_i = (t_{\mathsf{arrival}}, v_{arrival}, l_{arrival}, l_{exit})$, where t_{arrival} is the arrival time, $v_{arrival}$ is the arrival velocity, $l_{arrival}$ is the arrival lane from which the vehicle arrives at the intersection, and l_{exit} is the exit lane from which the vehicle leaves the intersection. The intersection manager can choose at most one of the proposals to grant a reservation. These proposals, except r_1, will be put in the queue, which is sorted by the proposed arrival times, and they will be processed by the intersection manager at a future time called the *next processing time*. r_1 is not put in the queue because its proposed arrival time is before the *request deadline*, which is equal to the sum of the next processing time and the *computation and communication delay* (the com. delay in Figure 2). The com. delay is a small time delay due to the IM's processing of the request messages at the next processing time and the communication delay between IM and the vehicle. r_1 is considered late because the arrival time of r_1 may has passed after the request deadline. The intersection manager processes late proposals like r_1 immediately, to see if it is possible to grant the reservation between the reservations that were granted at the last processing time. If not, a reject message is sent.

The request handling procedure processes request messages in the queue at the next processing time. The procedure first identifies the *target batch* of request messages on the queue, which is the set of all request messages whose proposed arrival times are before $t_{\mathsf{deadline}} + t_{\mathsf{batch}}$, where t_{deadline} is the request deadline and t_{batch} is the *batch interval*

which is 6 seconds in Figure 2. The request messages in the target batch will be removed from the queue and reordered by a *cost function*, which is $f(wait) = a \times (wait)^b$, where a and b are constants and *wait* is the estimated amount of time the vehicle has been waiting to enter the intersection. The procedure grants reservations according to the new order and then rejects the requests from the vehicles that have no reservation and no remaining request messages on the queue. Finally, both the next processing time and the request deadline are increased by time t_{proc}, which is called the *processing interval* and is the time between the batch processing of requests.

Enforcing Liveness in AIM

The reordering of request messages is based on the "cost" of denying vehicles' reservations. Basically, the longer a vehicle is waiting to enter the intersection, the higher the cost is. If a vehicle has been waiting to enter an intersection for a very long time, the batch policy will regard its requests as top priority and consider granting reservations to this vehicle before other vehicles. This reordering strategy, however, does not guarantee that requests with the highest cost in a batch always get a reservation. It is possible that some of the existing reservations from the previous batches have taken reservation tiles that are needed by the request. If this occurs every time a vehicle makes a request, the vehicle will be stalled at the intersection indefinitely.

We say an intersection is *live* if every vehicle waiting to enter the intersection can eventually enter and then leave the intersection. We modify the batch policy to *guarantee* the liveness—vehicles can eventually get a reservation. The idea is that when the policy processes the request messages of a batch, if it finds that there is a request message r whose cost is larger than a threshold, it should not grant any reservation to any other vehicles whose paths intersects the path of the vehicle of r, until the vehicle of r gets a reservation. We say r is *locked* and call this modified policy *the batch policy with locking*.

Theorem 1 *When an intersection manager in AIM uses the batch policy with locking, a vehicle will eventually get a reservation if every vehicle resends a request message after receiving a reject message.* [1]

Enforcing Liveness in Road Networks

Achieving liveness at all *individual* intersections, however, is not sufficient to guarantee the liveness of an entire road *network*. For instance, the gridlock in Figure 1(b) cannot be avoided solely by preventing vehicles from waiting to enter the intersections indefinitely. In this section, we examine the sufficient condition for liveness of road networks.

Discretized Road Network

Let us analyze the liveness property in a discrete version of a road network as shown in Figure 3. A *discretized road network* is a finite, connected, directed graph $G = (P, E)$, where

[1]The proofs of all theorems in this paper are available in an online appendix at `http://www.cs.utexas.edu/~aim/papers/AAAI2011-Au-proofs.pdf`.

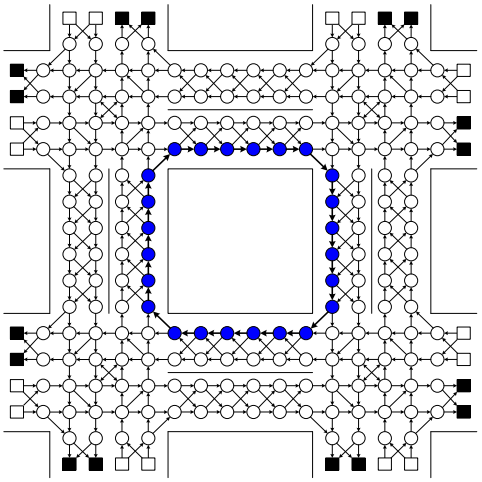


Figure 3: A discretized road network. The white squares are the sources and the black squares are the destinations. The vehicles at the solid blue circles form a soft gridlock for deterministic controllers since they are blocked in their direction of movement (bold arrows).

P is a finite set of nodes called *positions* and $E \subseteq P \times P$ is the set of directed edges among positions. Let $\mathsf{next}(p)$ be the set of *next* positions of the position p, such that $\mathsf{next}(p) = \{p' : (p, p') \in E\}$.

Each discretized road network has a non-empty set $P_{\mathsf{src}} \subset P$ of positions called *sources* at which vehicles enter the network. At each time step, a source $p \in P_{\mathsf{src}}$, if unoccupied, "spawns" a new vehicle with a probability ρ_p. The source of a vehicle v is denoted by $\mathsf{src}(v)$. Each discretized road network also has a non-empty set $P_{\mathsf{dst}} \subset P$ of positions called *destinations* at which vehicles leave the network. If a vehicle moves into a destination position, the vehicle will be removed from the network at the end of the time step. The *body* of G is a subgraph (P', E') where $P' = P \setminus (P_{\mathsf{src}} \cup P_{\mathsf{dst}})$ and $E' = \{(p_1, p_2) \in E : p_1, p_2 \in P'\}$. We assume 1) $(P_{\mathsf{src}} \cap P_{\mathsf{dst}}) = \emptyset$; 2) the in-degrees of sources and the out-degrees of destinations are zeros; 3) the out-degrees of sources and the in-degrees of destinations are non-zeros; 4) $\mathsf{next}(p) \subseteq P'$ for all $p \in P_{\mathsf{src}}$; and 5) $\{p' : p \in \mathsf{next}(p')\} \subseteq P'$ for all $p \in P_{\mathsf{dst}}$.

Each vehicle v occupies exactly one position $\mathsf{pos}(v)$, and each position p can be *occupied* by at most one vehicle $\mathsf{veh}(p)$. If p is *unoccupied*, $\mathsf{veh}(p) = \emptyset$. Let $\mathsf{next}(v) = \mathsf{next}(\mathsf{pos}(v))$ be the set of next positions of v. Vehicles at P_{dst} leave the road network at the end of a time step, thus all destinations are unoccupied at the beginning of a time step. When a vehicle v is spawned, a destination $\mathsf{dst}(v) \in P_{\mathsf{dst}}$ is assigned to v such that the goal of v is to reach $\mathsf{dst}(v)$.

Some positions such as those in an intersection do not allow vehicles to freely enter them, because the traffic on these positions are controlled by *traffic control mechanisms* such as traffic signals, stop signs, and the AIM protocol. We say these positions are *managed*, while those that are not controlled by any traffic control mechanisms are *unmanaged*. All vehicles have the *right of way* to enter un-

managed positions at all times, but that is not necessarily true for managed positions—the traffic control mechanism must assign the right of way of the managed position to a vehicle at time t to allow the vehicle to move into the position. Every managed position p is associated with a set $\mathsf{permitted}(p,t) = \{v_1, v_2, \ldots, v_n\}$ of *permitted* vehicles at every time step t, which means that the vehicle v_i has the right of way to enter p at time t. A traffic control mechanism is simply a protocol that assigns vehicles to $\mathsf{permitted}(p,t)$. For example, if a green signal lets all vehicles on the eastbound road to enter the intersection, $\mathsf{permitted}(p,t)$ will contain all vehicles on the eastbound road for all p in the intersection and all t at which the signal is green.

At the beginning of a time step t, a vehicle v must choose one of its next position $p \in \mathsf{next}(v)$ as its *chosen* next position, denoted by $\mathsf{chosen}_v(t)$. Two or more vehicles can have the right of way to enter the same position p at the same time t. If these vehicles choose to move into p at t, only one of them will move into p and the others will not move. Formally, suppose there exists a set $V_p(t) = \{v_1, v_2, \ldots, v_n\}$ of vehicles ($n \geq 2$) such that 1) v_i has the right of way to enter a position p at time t and 2) $\mathsf{chosen}_{v_1}(t) = \mathsf{chosen}_{v_2}(t) = \ldots = \mathsf{chosen}_{v_n}(t) = p$. We say the vehicles in $V_p(t)$ are *competing* with each other at p at time t. All the competitions at p over time are governed by a *coordination mechanism* Λ_p, which decides which competing vehicle in $V_p(t)$ can move into p at the end of the time step and the remaining vehicles in $V_p(t)$ cannot move. We denote the winner of the competition by $\Lambda_p(V_p(t))$.

In summary, a vehicle v at $\mathsf{pos}(v)$ chooses a next position $p \in \mathsf{next}(v)$ at the beginning of a time step t, and v can move into p by the end of t if and only if the following three conditions are satisfied: 1) p is unoccupied; 2) v has the right of way of p at time t (either p is unmanaged or $v \in \mathsf{permitted}(p,t)$; and 3) either v has no competing vehicle or the coordination mechanism Λ_p at p chooses v to move into p (i.e., $\Lambda_p(V_p(t)) = v$). If v cannot move into p, v will remain at $\mathsf{pos}(v)$ at the end of the time step.

Gridlock

Let us precisely define what gridlock is. We say a vehicle v is *completely blocked* if all positions of v in $\mathsf{next}(v)$ are occupied. A *hard gridlock* in a road network $G = (P,E)$ is a connected subgraph (P',E') of G, where $P' \neq \emptyset$ and $E' = \{(p_1,p_2) : p_1, p_2 \in P' \text{ and } p_2 \in \mathsf{next}(p_1)\}$, that satisfies the following two conditions: 1) all positions in P' are occupied; and 2) $\mathsf{next}(p) \subseteq P'$ for all $p \in P'$; In essence, a hard gridlock is a set of vehicles that are completely blocked by each other.

Theorem 2 *If the body of a discretized road network G is strongly connected (every position is reachable from every other position), there can be no hard gridlock in G.*

Theorem 2 implies that hard gridlocks do not frequently occur in the real world since most real road networks are strongly connected. But hard gridlock is by no means the only type of gridlock. In fact, a vehicle cannot move as long as the next position the vehicle *intends* to move into is occupied. A *soft gridlock for deterministic controllers (SGDC)* at time t is a cycle $\langle p_1, p_2, \ldots, p_n \rangle$ of n positions ($n \geq 2$) in G such that 1) p_i is occupied by v at time t for $1 \leq i \leq n$; and 2) $\mathsf{chosen}_{v_i}(t) = p_{i+1}$ for $1 \leq i < n$ and $\mathsf{chosen}_{v_n}(t) = p_1$. Basically a SGDC is a ring of vehicles that choose to follow one another at time t. Figure 3 shows an instance of SGDC.

Soft gridlock is much "weaker" than hard gridlock, because vehicles are *not* necessarily stuck forever, as some vehicles can choose a different next position in the future. We say a vehicle's controller is *stochastic* if it may choose different next positions in $\mathsf{next}(p)$ at different times for a given position p. A stochastic controller may potentially break a SGDC, unless all next positions that are *possibly* chosen by the vehicles are already occupied. Hence we define another version of soft gridlock based on stochastic controllers. We say a next position $p \in \mathsf{next}(v)$ is *relevant* if v will possibly choose p in the next time step. *Irrelevant* next positions are those that are not chosen by the vehicle. One possible reason for a position to be irrelevant is that it is not on any route to its destination. Let $\Pi_v(p) \subseteq \mathsf{next}(v)$ be the set of all relevant next positions when v is located at $p = \mathsf{pos}(v)$. A vehicle v is *essentially blocked* if all positions in $\Pi_v(p)$ are occupied. A *soft gridlock for stochastic controllers (SGSC)* is a connected subgraph (P',E') of G, where $P' \neq \emptyset$ and $E' = \{(p_1,p_2) : p_1, p_2 \in P' \text{ and } p_2 \in \mathsf{next}(p_1)\}$, such that 1) all positions in P' are occupied; and 2) $\Pi_{\mathsf{veh}(p)}(p) \subseteq P'$ for all $p \in P'$. As in hard gridlocks, vehicles in a SGSC are stuck forever since they are always blocked by some other vehicles no matter what the next position their stochastic controllers choose.

Liveness

Obviously vehicles in a SGDC cannot move if all vehicle controllers are deterministic (i.e., there is exactly one relevant next position $p' \in \Pi_v(p)$ and $\mathsf{chosen}_v(t) = p'$ whenever v is located at p). But the lack of SGDCs at all times is not sufficient to guarantee all vehicles with deterministic controllers can move towards their destinations; a few more conditions are needed in order to provide such a guarantee.

A traffic control mechanism Ψ_p for a position p is *open* if whenever a vehicle v repeatedly chooses to move into p when p is unoccupied, Ψ_p will eventually give v the right of way to enter p. More precisely, for any infinite sequence $\langle t_1, t_2, \ldots \rangle$ of times there exists an integer $n \geq 1$ such that if p is unoccupied at t_i and $\mathsf{chosen}_v(t_i) = p$ for $1 \leq i \leq n$, then $v \in \mathsf{permitted}(p,t_n)$. Traffic signals, stop signs, and the AIM protocol (with the batch processing of requests) are open traffic control mechanisms.

A coordination mechanism Λ_p at a position p is *fair* if whenever a vehicle v repeatedly chooses p as its chosen next position, v will eventually move into p. More precisely, for any infinite sequence $\langle t_1, t_2, \ldots \rangle$ of times there exists an integer $n \geq 1$ such that if $\mathsf{chosen}_v(t_i) = p$ for $1 \leq i \leq n$ then $\mathsf{pos}(v) = p$ at time t_n. An example of a fair coordination mechanism is one that chooses the vehicle that has spent the longest time waiting to enter the position.

When a vehicle v with a deterministic controller is spawned it has already chosen the path towards its destination. Let $\langle p_1, p_2, \ldots, p_n \rangle$ be the *chosen path*, where $p_1 = \mathsf{src}(v)$ and $p_n = \mathsf{dst}(v)$, such that the controller of v only chooses the positions on this path to move into (i.e.,

$\Pi_v(p_i) = \{p_{i+1}\}$ for $1 \leq i < n$). We assume chosen paths are finite.

Theorem 3 *Every spawned vehicle will eventually reach its destination if 1) all vehicle controllers are deterministic; 2) all traffic control mechanisms are open; 3) all coordination mechanisms are fair; and 4) there is no SGDC at any time.*

We say a road network G is *live* if every spawned vehicle will eventually reach its destination. Even if a SGDC does occur at a certain time, a road network may still be live if vehicle controllers are stochastic. Hence, we relax the conditions in Theorem 3 by allowing stochastic controllers and SGDCs, as long as these SGDCs do not constitute a SGSC.

A controller of a vehicle v is *progressive* if the visited positions of the vehicle always constitute a prefix of a non-cyclic path from the source to the destination. A controller of v is *opportunistic* if whenever a relevant next position $p \in \Pi_v(\mathsf{pos}(v))$ of v is unoccupied repeatedly, v will eventually choose p.

Theorem 4 *Every spawned vehicle will eventually reach its destination if 1) all traffic control mechanisms are open; 2) all coordination mechanisms are fair; 3) there is no SGSC at any time; and 4) all vehicle controllers are progressive and opportunistic.*

Assume all controllers are progressive (otherwise, a vehicle may wander around the network and never reach its destination). Theorem 4 suggests that one way to prevent starvation in a road network is to 1) use open traffic control mechanisms such as the batch policy with locking; 2) devise a mechanism to prevent SGSC from occurring; and 3) establish laws to enforce that autonomous vehicles coordinate with other vehicles in a fair manner and will choose different routes if they get stuck for too long. Even though our analysis is based on discretized road network which is different from road network in the real world, our analysis should teach us what is needed in order to avoid starvation in realistic scenarios.

Experiments

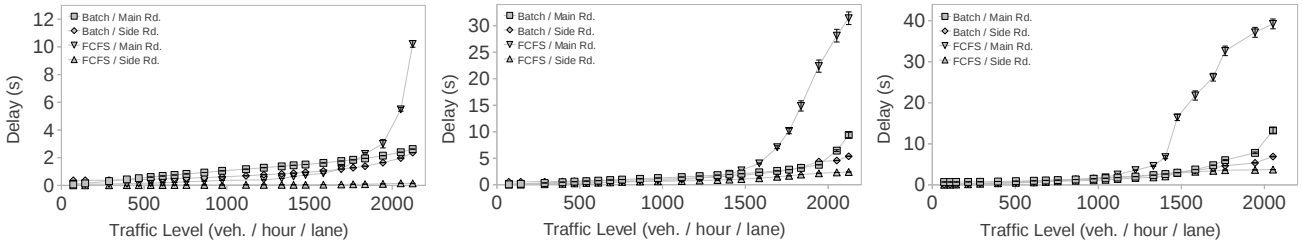
The theoretical results in the previous section present sufficient conditions for liveness of a *network* of intersections, using an abstract road model. One of the important conditions is that the individual intersections in the network be live themselves. The batch policy presented in the first part of this paper is the first intersection control policy in AIM that is guaranteed to satisfy this condition. In this section, we report on experiments designed to test the empirical performance of the batch policy. If its liveness came at a significant practical cost, it would not be a promising replacement for FCFS. Fortunately, we find that it both significantly improves performance in unbalanced traffic, as it was designed to do, and also performs roughly as well as, and in some cases significantly better than, FCFS in balanced traffic.

Unbalanced Traffic. We conducted an experiment on an intersection between a main road and a side road, where the main road has much higher traffic than the side road. Each of the roads has three lanes, and the vehicles on the main road go straight through the intersection without turning while the vehicles on the side road can either turn left, turn right or pass through the intersection. The vehicles are spawned at the beginning of each lane according to a poisson distribution such that the traffic level λ_{main} of the main road is varied from 72 vehicles per hour per lane to 2200 vehicles per hour per lane while the traffic level λ_{side} of the side road is held constant. We ran three sets of experiments with different values of λ_{side}: 360, 540, and 720 vehicles/hour/lane. For each set of experiments, we ran the simulation 100 times, and in each run the total simulation time was 1 hour. FCFS has no parameters. In the batch policy with locking, the coefficients of the cost function are set to $a = 1.0$ and $b = 2.0$, the batch interval is $3s$, the processing interval is $0.5s$, the com. delay is $0.02s$, and the locking threshold is 700 (which means that the requests of a vehicle are locked when the waiting time is larger than $26.5s$). The *delay* of a vehicle is defined as the amount of travel time incurred by the vehicle as the result of passing through the intersection. We measured the average delay of the vehicles by averaging the time difference of the vehicles with and without other vehicles on the roads.

Figure 4 shows the results of the three sets of experiments. Note that there are tiny error bars in the figure showing the 95% confidence intervals of the average delays in the 100 runs of the experiments. Contrary to the situation illustrated in Figure 1(a), in our experiments, it is the vehicles on the main road that are most affected by increased traffic rates. The reason is that during the experiments, only the first car in each lane was allowed to request a reservation. When cars further back can also request reservations, we expect to see higher delays on the side road. In our case, when λ_{main} is high and FCFS is used, the vehicles on the main road have difficulty getting reservations due to a single vehicle on the side road being able to block several on the main road. But according to Figure 4, the delay of the vehicles on the main road is reduced tremendously due to the use of the batch policy, at the cost of a very small increase of the delays on the side street. The batch policy is more effective when λ_{main} is large. When λ_{side} is 720, the batch policy cannot reduce the delays to less than 10 seconds at very high λ_{main}. But still it offers a significant reduction in delay. Note that in all cases, the increase of the delays of the vehicles on the side street is relatively small.

To see the overall effect of the batch policy on the traffic in all directions, we combined the data in Figure 4(b) and computed the average delays of *all* vehicles on both the main road and the side road. Figure 5 shows that the average delay is about 3.6 times smaller when the batch policy is used when λ_{main} is high. This is mainly the result of the large decrease in the delays on the main road. Thus, the batch policy is better than FCFS at intersections with unbalanced traffic in terms of the overall traffic flow of the intersection.

Balanced Traffic. So far we showed that batch policy outperform FCFS at intersections with unbalanced traffic—the traffic level of one road is much larger than the traffic level of the other. If the batch policy is to be used, it is also important that it works well in balanced traffic compared to FCFS. Here we present the result of an experiment in which the traffic levels of the roads of an intersection are the same. In this experiment we varied the traffic level on all roads from



(a) $\lambda_{\text{side}} = 360$ (b) $\lambda_{\text{side}} = 540$ (c) $\lambda_{\text{side}} = 720$

Figure 4: Average delays of the vehicles versus traffic levels of the main road. The delays of the vehicles on the main road and the side road are shown separately. The error bars on the data points indicate the 95% confidence intervals of the delays.

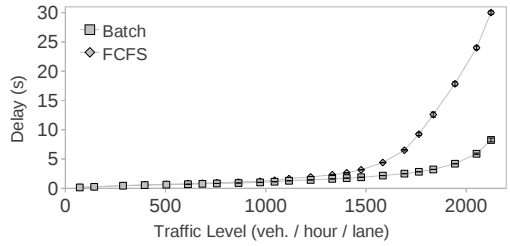


Figure 5: Average delays of all vehicles (on the main and side roads) vs. the traffic level of the main road. $\lambda_{\text{side}} = 540$.

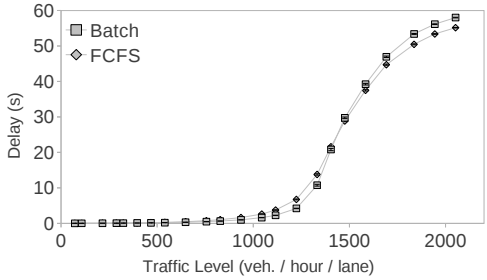


Figure 6: Average delays vs. traffic levels in balanced traffic.

72 to 2200 vehicles/hour/lane and ran the simulation for 1 hour. The parameters of the batch policy are the same as the parameters in the previous sections. We repeated the simulation 100 times and took the average of the delays of all vehicles passing through the intersection. The average delays were plotted against the traffic levels, as shown in Figure 6.

The performance of the batch policy is almost the same as FCFS's performance. This is expected since starvation as shown in Figure 1(a) does not occur in balanced traffic, and vehicles do not have to spend too much time to wait to enter the intersection. Nonetheless, at intermediate traffic levels (800 to 1400 vehicles/hour/lane), the delays from the batch policy are a small but significant amount below the delays from FCFS (the 95% confident intervals are not overlapped). Hence, the batch policy are still able to reduce delays by reordering the requests, even when few vehicles wait at the intersection for a long time. At high traffic levels, the delays from the batch policy are slightly but significantly higher than FCFS. Perhaps this is because the batch policy process requests only at certain times (defined by the processing interval) thus there is an inherent delay due to batching.

Conclusions and Future Work

Recent developments in autonomous vehicles lead us to believe that autonomous vehicles will be widely adopted in the future. Therefore, it is essential to develop multiagent techniques to properly manage the traffic of autonomous vehicles (Bazzan 2005). In this paper, we introduced a new intersection control policy to prevent inequalities in granting reservations due to the large discrepancy of traffic volume among the incoming roads. Our experimental results show that the new policy outperforms FCFS, the best autonomous intersection control protocol in the literature, in unbalanced traffic. We also introduced a modified scheme to guarantee that all vehicles can get a reservation eventually as long as they keep sending the request message after the rejection of previous requests. We analyzed conditions for system-wide liveness in a simplified road network and presented a set of conditions on traffic control mechanisms and vehicle controllers that is sufficient to prevent starvation. Our ongoing research agenda includes merging these contributions in traffic control system in a citywide network of intersections.

Acknowledgments. This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (IIS-0917122), ONR (N00014-09-1-0658), and the FHWA (DTFH61-07-H-00030).

References

Au, T.-C., and Stone, P. 2010. Motion planning algorithms for autonomous intersection management. In *AAAI 2010 Workshop on Bridging The Gap Between Task And Motion Planning (BTAMP)*.

Bazzan, A. L. C. 2005. A distributed approach for coordination of traffic signal agents. *Autonomous Agents and Multi-Agent Systems* 10(2):131–164.

Cervero, R. 1986. *Suburban Gridlock*. Center for Urban Policy Research.

Dijkstra, E. W. 1971. Hierarchical ordering of sequential processes. *Acta informatica* 1(2):115–138.

Dresner, K., and Stone, P. 2008. A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research (JAIR)*.

Quinlan, M.; Au, T.-C.; Zhu, J.; Stiurca, N.; and Stone, P. 2010. Bringing simulation to life: A mixed reality autonomous intersection. In *IEEE/RSJ International conference on Intelligent Robots and Systems*.