On the Complexity of an Unregulated Traffic Crossing

Philip Dasler(✉) and David M. Mount

Department of Computer Science, University of Maryland,
College Park, MD 20742, USA
{daslerpc,mount}@cs.umd.edu

Abstract. One of the most challenging aspects of traffic coordination involves traffic intersections. In this paper we consider two formulations of a simple and fundamental geometric optimization problem involving coordinating the motion of vehicles through an intersection.

We are given a set of n vehicles in the plane, each modeled as a unit length line segment that moves monotonically, either horizontally or vertically, subject to a maximum speed limit. Each vehicle is described by a start and goal position and a start time and deadline. The question is whether, subject to the speed limit, there exists a collision-free motion plan so that each vehicle travels from its start position to its goal position prior to its deadline.

We present three results. We begin by showing that this problem is NP-complete with a reduction from 3-SAT. Second, we consider a constrained version in which cars traveling horizontally can alter their speeds while cars traveling vertically cannot. We present a simple algorithm that solves this problem in $O(n \log n)$ time. Finally, we provide a solution to the discrete version of the problem and prove its asymptotic optimality in terms of the maximum delay of a vehicle.

1 Introduction

As autonomous and semi-autonomous vehicles become more prevalent, there is an emerging interest in algorithms for controlling and coordinating their motions to improve traffic flow. The steady development of motor vehicle technology will enable cars of the near future to assume an ever increasing role in the decision making and control of the vehicle itself. In the foreseeable future, cars will have the ability to communicate with one another in order to better coordinate their motion. This motivates a number of interesting algorithmic problems. One of the most challenging aspects of traffic coordination involves traffic intersections. In this paper we consider two formulations of a simple and fundamental geometric optimization problem involving coordinating the motion of vehicles through an intersection.

Supported by the National Science Foundation under grant CCF-1117259 and the Office of Naval Research under grant N00014-08-1-1015.

F. Dehne et al. (Eds.): WADS 2015, LNCS 9214, pp. 224–235, 2015.
DOI: 10.1007/978-3-319-21840-3_19

Traffic congestion is a complex and pervasive problem with significant economic ramifications. Practical engineering solutions will require consideration of myriad issues, including the physical limitations of vehicle motion and road conditions, the complexities and dynamics of traffic and urban navigation, external issues such as accidents and break-downs, and human factors. We are motivated by the question of whether the field of algorithm design can contribute positively to such solutions. We aim to identify fundamental optimization problems that are simple enough to be analyzed formally, but realistic enough to contribute to the eventual design of actual traffic management systems. In this paper, we focus on a problem, the *traffic crossing problem*, that involves coordinating the motions of a set of vehicles moving through a system of intersections. In urban settings, road intersections are *regulated* by traffic lights or stop/yield signs. Much like an asynchronous semaphore, a traffic light locks the entire intersection preventing cross traffic from entering it, even when there is adequate space to do so. Some studies have proposed a less exclusive approach in which vehicles communicate either with one another or with a local controller that allows vehicles, possibly moving in different directions, to pass through the intersection simultaneously if it can be ascertained (perhaps with a small adjustment in velocities) that the motion is collision-free (see, e.g., [9]). Even though such systems may be beyond the present-day automotive technology, the approach can be applied to controlling the motion of parcels and vehicles in automated warehouses [17].

Prior work on autonomous vehicle control has generally taken a high-level view (e.g., traffic routing [5,6,15,18]) or a low-level view (e.g., control theory, kinematics, etc. [10,14]). We propose a mid-level view, focusing on the control of vehicles over the course of minutes rather than hours or microseconds, respectively. The work by Fiorini and Shiller on velocity obstacles [11] considers motion coordination in a decentralized context, in which a single agent is attempting to avoid other moving objects. Much closer to our approach is work on *autonomous intersection management* (AIM) [2,4,7–9,16]. This work, however, largely focuses on the application of multi-agent techniques and deals with many real-world issues. As a consequence, formal complexity bounds are not proved. Berger and Klein consider a dynamic motion-panning problem in a similar vein to ours, which is loosely based on the video game *Frogger* [3]. Their work is based, at least in part, on the work of Arkin, Mitchell, and Polishchuk [1] in which a group of circular agents must cross a field of polygonal obstacles. These obstacles are dynamic, but their motion is fixed and known *a priori*.

We consider a simple problem formulation of the traffic crossing problem, but one that we feel captures the essential computational challenges of coordinating crosswise motion through an intersection. Vehicles are modeled as line segments moving monotonically along axis-parallel lines (traffic lanes) in the plane. Vehicles can alter their speed, subject to a maximum speed limit, but they cannot reverse direction. The objective is to plan the collision-free motion of these segments as they move to their goal positions.

After a formal definition of our traffic crossing problem in Section 2, we present three results. First, we show in Section 3 that this problem is NP-complete.

(While this is a negative result, it shows that this problem is of a lower complexity class than similar PSPACE-complete motion-planning problems, like sliding-block problems [12].) Second, in Section 4 we consider a constrained version in which cars traveling vertically travel at a fixed speed. This variant is motivated by a scenario in which traffic moving in one direction (e.g., a major highway) has priority over crossing traffic (e.g., a small road). We present a simple algorithm that solves this problem in $O(n \log n)$ time.

Finally, we consider the problem in a discrete setting in Section 5, which simplifies the description of the algorithms while still capturing many of the interesting scheduling elements of the problem. As part of this consideration, we provide a solution to the problem that limits the maximum delay of any vehicle and prove that this solution is asymptotically optimal.

2 Problem Definition

The Traffic Crossing Problem is one in which several vehicles must cross an intersection simultaneously. For a successful crossing, all vehicles must reach the opposite side of the intersection without colliding, and they must do so in a reasonable amount of time. Formally, a traffic crossing is defined as a tuple $C = (V, \delta_{max})$. This tuple is comprised of a set of n vehicles V which exist in $\mathbb{R}^2$ and a global speed limit $\delta_{max} \in \mathbb{R}^+$, where $\mathbb{R}^+$ denotes the set of nonnegative reals. Each vehicle is modeled as a vertical or horizontal open line segment that moves parallel to its orientation. Like a car on a road, each vehicle moves monotonically, but its speed may vary between zero and the speed limit. A vehicle's position is specified by its leading point (relative to its direction).

Each vehicle $v_i \in V$ is defined as a set of properties, $v_i = \{l_i, p_i^{\vdash}, p_i^{\dashv}, t_i^{\vdash}, t_i^{\dashv}\}$[1], where l_i is the vehicle's length, $p_i^{\vdash}$ and $p_i^{\dashv}$ are its initial and goal positions, respectively, and $t_i^{\vdash}$ and $t_i^{\dashv}$ are its start time and deadline for reaching its goal position.

The set V and the global speed limit δ_{max} define the problem and remain invariant throughout. Our objective is to determine whether there exists a collision-free motion of the vehicles that respects the speed limit and satisfies the goal deadlines. Such a motion is described by a set of functions, called speed profiles, that define the instantaneous speed of the vehicles at time t.

This set of functions is defined as $D = \{\delta_i(t) \mid i \in [1, n], \forall t, 0 \leq \delta_i(t) \leq \delta_{max}\}$. A set D of speed profiles is *valid* if no vehicle (1) moves prior to its start time or after its deadline, (2) violates the speed limit or travels in reverse (3) collides with another vehicle or (4) fails to reach its goal prior to its deadline.

A traffic crossing C is solvable if there exists a valid set of speed profiles D.

3 Hardness of Traffic Crossing

Determining whether a given instance of the traffic crossing problem is solvable is NP-complete. We show its NP-hardness by proving the following theorem:

[1] The notational use of $\vdash$ and $\dashv$ set above a variable (e.g., $\alpha^{\vdash}$) represents the beginning and end of a closed interval, respectively (e.g., start and end times).

Theorem 1. *Given a Boolean formula F in 3-CNF, there exists a traffic crossing $C = (V, \delta)$, computable in polynomial time, such that F is satisfiable if and only if there exists a valid set of speed profiles D for C.*

The input to the reduction is a boolean formula F in 3-CNF (i.e., an instance of 3-SAT). Let $\{z_1, \ldots, z_n\}$ denote its variables and $\{c_1, \ldots, c_m\}$ denote its clauses. Each variable z_i in F is represented by a pair of vehicles whose motion is constrained to one of two possible states by intersecting their paths with a perpendicular pair of vehicles. This constraining mechanism (seen in Fig. 1) is the core concept around which all mechanisms in the reduction are built. It allows us to represent logical values, to transmit these values throughout the construction, and to check these values for clause satisfaction.

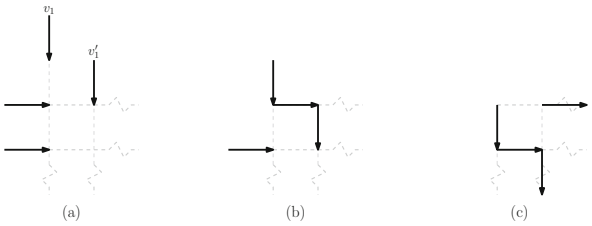

Fig. 1. (a) An example of transferring values at $t_i^{\vdash}$. v_1 and v_1' are **true** and **false**, respectively. (b) At time $t_i^{\vdash} + 1$, the upper horizontal vehicle will take on the value of v_1' while the lower takes the value of v_1. (c) $= t_i^{\vdash} + 2$.

All vehicles in the reduction are of unit length and (barring a few special cases) their deadlines are set so that they can reach their goal position with at most one unit time delay. More formally, $t_i^{\dashv} - t_i^{\vdash} - 1 = \frac{(\|p_i^{\dashv} - p_i^{\vdash}\|)}{\delta_{max}}$. In general, the delay may take multiple forms (e.g., the vehicle could take a delay of 1 at any point during its travel or spread the delay out by traveling slower than δ_{max}), but the mechanism described above constrains the delay to only one of two types: a delay of exactly 0 or 1 taken immediately at the vehicle's start time.

For each clause $c_i \in F$, a mechanism is created that forces a collision if, and only if, all three literals are **false**. This mechanism checks the positive and negative literals separately, then combines the results in order to determine whether the clause is satisfied.

These mechanisms each require only a constant number of vehicles, resulting in a reduction complexity on the order of $O(n + m)$, where n and m are the number of variables and clauses, respectively[2].

3.1 Membership in NP

Lemma 1. *The Traffic Crossing Problem is in* NP.

[2] Detailed descriptions of these mechanisms have been omitted due to space constraints, but can be found in the arXiv version of this paper.

First, observe that for each pair of orthogonal vehicles, v_i, v_j, their paths cross at a single intersection. The certificate provides a priority for each such pair, specifying which vehicle crosses through the intersection first. Next, it can be shown that if there exists a valid set of speed profiles for an instance of the problem, then there exists another valid set where vehicles move at the maximum speed and are subject to the constraints in the certificate. Finally, when proving the validity of a solution provided by the certificate, only a number of events polynomial in n must be processed and the number of bits of precision required is polynomial in the number of bits in the input plus $\log n$. For the sake of space, the formal proof has been omitted.

4 A Solution to the One-Sided Problem

While the generalized Traffic Crossing Problem is NP-complete, it is possible to solve a constrained version of the problem more efficiently. The complexity of the generalized Traffic Crossing Problem arises from the interplay between horizontal and vertical vehicles, which results in a complex cascade of constraints. To break this interdependency, the vertically traveling vehicles are given priority, allowing them to continue through the intersection at a fixed speed. In this variant, called the *one-sided problem*, the horizontal vehicles can plan their motion with complete information and without fear of complex constraint chains.

First, we assume that the vertically traveling vehicles are invariant and are all traveling at the same speed, s_n. With vertical vehicle motion now fixed, there is no way for horizontal vehicles to affect each other and movement profiles for each can be found in isolation from the others. Finally, we assume that all vehicles are of length l and in general position.

For the purpose of illustration we begin with a simplified version of the problem and then, over the course of three cases, relax the restrictions until we are left with a solution to the original problem under the fixed, one-sided policy described above. These three cases are:

Intersection Between One-Way Highways
- Vertical vehicles approach from the North only.
- Horizontal vehicles approach from the West only.
- Each vehicle is in its own lane (i.e., no two vehicles are collinear).

Intersection Between a One-Way Street and a Two-Way Highway
- Vertical vehicles approach from the North and the South.
- Horizontal vehicles approach from the West only.
- There is a single horizontal lane (i.e., all horizontal vehicles are collinear) and one or more vertical lanes.

Intersection Between Two-Way Highways
- Vertical vehicles approach from the North and the South.
- Horizontal vehicles approach from the West and the East.
- There are k horizontal lanes, one or more vertical lanes, and vehicles may share lanes.

4.1 Intersection Between One-Way Highways

Formally, vehicles from the North are in the subset $N \subset V$ and their direction of travel is $d_n = (0, -1)$, where as vehicles from the West are in the subset $W \subset V$ with a direction of travel of $d_w = (1, 0)$. Again, our only task is to find valid speed profiles for vehicles coming from the West.

To begin, the problem space is transformed so that the vehicles in W are represented as points rather than line segments. This makes movement planning simpler while maintaining the geometric properties of the original space. Every vehicle in W is contracted from left to right, until it is reduced to its leading point. In response, the vehicles in N are expanded, transforming each into a square obstacle with sides of length l and with their left edges coincident with the original line segments.

Given the global speed limit δ_{max}, there are regions in front of each obstacle in which a collision is inevitable (this concept is similar to the obstacle avoidance work done in [13]). These triangular zones (referred to as *collision zones*) are based on the speed constraints of the vehicles and are formed by a downward extension of the leading edge of each obstacle. The leftmost point of this edge is extended vertically and the rightmost point is extended at a slope derived from the ratio between δ_{max} and the obstacle speed. As one last concession to clarity, we scale the axes of our problem space so that this ratio becomes 1. Formally, a collision zone Z_O for the obstacle O is the set of all points p, such that there is no path originating at p with a piecewise slope in the interval $[1, \infty]$ that does not intersect O.

Expanding the vehicles in N into rectangular obstacles may cause some to overlap, producing larger obstacles and, consequently, larger collision zones. This merger and generation of collision zones is done through a standard sweep line algorithm and occurs in $O(n \log n)$ steps, where n is the number of obstacles, as described below.

Merging Obstacles and Growing Collision Zones. This process is done using a horizontal sweep line moving from top to bottom. While the following is a relatively standard application of a sweep line algorithm, it is included for the sake of completeness. First, the event list is populated with the horizontal edges of every obstacle, in top-to-bottom order, requiring $O(n \log n)$ time for $O(n)$ obstacles. The sweep line status stores a set of intervals representing the interiors of disallowed regions (e.g., the inside of an obstacle or collision zone). Each interval holds three pieces of information: the location of its left edge, a sorted list of the right edges of any obstacles within the interval, and the slopes of these right edges. These slopes will be either infinite (i.e., the edges are vertical) or will have a slope of 1.

In addition to horizontal edge positions, the event list must keep track of three other events which deal with the termination of the sloped edges of the collision zones. These edges begin at the bottom right edge of an obstacle and terminate in one of three ways: against the top of another obstacle, against the right edge of another obstacle, or by reaching the left edge of an interval. The

first case is already in the event list as the top edges were added at the start of this process. The remaining two cases are added as the sweep line progresses through the obstacles.

The initial population of the event list occurs in $O(n \log n)$. As the sweep line progresses through the obstacle space, it adds and removes the right edges of obstacles to the appropriate intervals. These lists of edges are built incrementally in sorted order, requiring only $O(\log n)$ time. Finally, as there is a constant number of possible events per obstacle (a single top edge, a single bottom edge, and a single termination of its sloped edge), there are at most $O(n)$ events to be processed. Thus, the sweep line processes the obstacle space in $O(n \log n)$ time.

Movement Planning. Currently, vehicles only move horizontally and obstacles only move vertically. Instead, we will treat the obstacles as static objects and add a corresponding vertical component to the vehicles' motion. To find a movement plan, a vehicle moves through the obstacle space at maximum speed (giving it a slope of 1 under our scaled axes) until either reaching its goal or encountering an obstacle. If the goal is reached, the plan is complete. If an obstacle is encountered, the vehicle travels vertically until it is no longer blocked (this vertical motion corresponds to stopping and waiting for the obstacle to pass). Once the path is clear, the vehicle continues at maximum speed.

The path created by the above behavior can be found with another line sweep. The sweep line in this case is perpendicular to the vehicles' trajectories (giving it a slope of -1), moves from the upper right to the lower left, and determines how obstacles occlude one another, as seen from the vehicles' perspective. These occlusions reveal which obstacles are encountered and how the vehicle must move in order to follow the strategy laid out above.

4.2 Intersection Between a One-Way Street and a Two-Way Highway

In this case, vertical vehicles approach from the North and the South while horizontal vehicles travel in a single lane.

To account for the bidirectional vertical vehicles we fold the space along the horizontal lane. This rotates the northbound traffic to an equivalent southbound set of vehicles (see Fig. 2). This only requires a $O(n)$ transformation. Using the plane sweep algorithm above yields a combined obstacle space.

Finally, we must prevent the vehicles from rear-ending each other. Once the lead vehicle has found a motion plan through the obstacles, it creates a new set of constraints for the vehicles behind it. The monotonic path of the lead vehicle is stored in a binary search tree, allowing for easy collision queries. As each vehicle finds its own path through the obstacles, this search tree is updated to appropriately constrain subsequent vehicles [3].

In the end, we can still account for shared lanes without a running time greater than $O(n \log n)$.

[3] Details of how this is done can be found in the arXiv version of this paper.

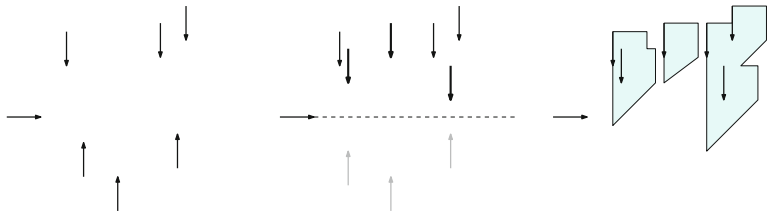

Fig. 2. (a) An example of bidirectional cross-traffic. (b) To account for how these vehicles interact when they reach a horizontal lane, we can fold the space along the lane, rotating one set of vehicles about it. (c) Then, we run the same space transformation and obstacle merger detailed above.

4.3 Intersection Between Two-Way Highways

Finally, this case combines the two above, allowing for bidirectional movement horizontally and vertically, with multiple lanes along each axis, and the possibility of collinear vehicles.

The vehicles approaching from the East are independent of those approaching from the West, presenting a symmetric problem that can be solved with the techniques discussed above. The addition of horizontal lanes, however, impacts the running time of the algorithm. Previously, the bidirectional vertical traffic was accounted for by folding the obstacle space along a single horizontal lane, but in this case, because the position of the vertical vehicles relative to each other is different at any given lane, the folding must occur individually for each lane. Thus, the algorithm runs in $O(kn \log n)$, for k horizontal lanes. In general, we assume that k is a relatively small constant.

5 Traffic Crossing in the Discrete Setting

In this section we consider the problem in a simple discrete setting, significantly simplifying the description of the algorithms and freeing us from a number of cumbersome continuous issues while still capturing the most salient elements of the original traffic-crossing problem. We assume that each vehicle occupies a point on the integer grid in the plane, $\mathbb{Z}^2$. Time advances discretely in unit increments, and at each time step a vehicle may either advance to the next grid point or remain where it is. A collision occurs if two vehicles occupy the same grid point.

The *discrete traffic crossing problem* is defined in much the same manner as in the continuous case. The problem is presented as a set V of n vehicles on the integer grid. Each vehicle v_i is represented by its initial and goal positions $p_i^{\vdash}$ and $p_i^{\dashv}$, respectively, both in $\mathbb{Z}^2$. Also given are a starting time $t_i^{\vdash}$ and deadline $t_i^{\dashv}$, both in $\mathbb{Z}^+$ (where $\mathbb{Z}^+$ denotes the set of nonnegative integers). A vehicle's direction d_i is a unit length vector directed from its initial position to its goal, which is either horizontal or vertical. Time proceeds in unit increments starting at zero. The motion of v_i is specified as a function of time, $\delta_i(t) \in \{0, 1\}$. Setting $\delta_i(t) = 0$ means that at time t vehicle i remains stationary, and $\delta_i(t) = 1$ means

that it moves one unit in direction d_i. Thus, v_i's position at time $t \geq 0$ is $p_i(t) = p_i^{\vdash} + d_i \sum_{x=0}^{t} \delta_i(x)$.

Generalizing the problem definition from Section 2, the objective is to compute a speed profile $D = \langle \delta_1, \ldots, \delta_n \rangle$ involving all the vehicles that specifies a collision-free motion of the vehicles in such a manner that each vehicle starts at its initial position and moves monotonically towards its goal, arriving there at or before its given deadline. Similar to road networks, we assume that along any horizontal or vertical grid line, the vehicle direction vectors are all the same.

5.1 Maximum Delay

Because we will be largely interested in establishing approximation bounds in this section, we will depart from the decision problem and consider a natural optimization problem instead, namely, minimizing the maximum delay experienced by any vehicle, defined formally as follows. For each vehicle we consider only its initial and goal positions, and let us assume that all vehicles share the same starting time at $t = 0$. A vehicle v_i experiences a *delay* at time t if it does not move at this time (that is, $\delta_i(t) = 0$). The *total delay* experienced by a vehicle is the total number of time instances where it experiences a delay until the end of the motion simulation. The *maximum delay* of the system is the maximum total delay experienced by any vehicle.

While we will omit a formal proof, it is not hard to demonstrate that the NP-hardness reduction of Section 3 can be transformed to one showing that it is NP-hard to minimize maximum delay in the discrete setting. (Intuitively, the reason is that the reduction involves purely discrete quantities: integer vehicle coordinates and starting times, vehicles of unit length, and unit speed limit. The system described in the reduction is feasible if and only if the maximum delay is at most five time units.) However, it is interesting to note that the question of whether there exists a solution involving at most single unit delay can be solved efficiently. This is stated in the following result.

Theorem 2. *There exists an $O(nm)$ time algorithm that, given an instance of the discrete traffic crossing problem with n vehicles where each vehicle encounters at most m intersections, determines whether there exists a solution with maximum delay of at most one time unit.*

Due to space limitations, we have omitted the proof, but the algorithm involves a straightforward reduction to 2-SAT. The key insight is that each vehicle can be in one of two states, *not-delayed* or *delayed*. Since all potential collisions involve pairs of vehicles, we can express the feasibility of a single unit delay solution through an instance of 2-SAT.

5.2 The Parity Heuristic

In the discrete setting it is possible to describe a simple common-sense heuristic. Intuitively, each intersection will alternate in allowing horizontal and vertical traffic to pass. Such a strategy might be far from optimal because each time a vehicle arrives at an intersection, it might suffer one more unit of delay. To

address this, whenever a delay is imminent, we will choose which vehicle to delay in a manner that will avoid cross traffic at all future intersections. Define the *parity* of a grid point $p = (p_x, p_y)$ to be $(p_x + p_y) \bmod 2$. Given a horizontally moving vehicle v_i and a time t, we say that v_i is *on-parity* at t if the parity of its position at time t equals $t \bmod 2$. Otherwise, it is *off-parity*. Vertically moving vehicles are just the opposite, being *on-parity* if the parity of their position is *not* equal to $t \bmod 2$. Observe that if two vehicles arrive at an intersection at the same time, one moving vertically and one horizontally, exactly one of them is on-parity. This vehicle is given the right of way, as summarized below.

Parity Heuristic: If two vehicles are about to arrive at the same intersection at the same time t, the vehicle that is on-parity proceeds, and the other vehicle waits one time unit (after which it will be on-parity, and will proceed).

The parity heuristic has a number of appealing properties. First, once all the vehicles in the system are on-parity, every vehicle may proceed at full speed without the possibility of further collisions. Second, the heuristic is not (locally) wasteful in the sense that it does not introduce a delay into the system unless a collision is imminent. Finally, the rule is scalable to large traffic systems, since a traffic controller at an intersection need only know the current time and the vehicles that are about to enter the intersection.

5.3 Steady-State Analysis of the Parity Heuristic

Delays may be much larger than a single time unit under the parity heuristic. (For example, a sequence of k consecutive vehicles traveling horizontally that encounters a similar sequence of k vertical vehicles will result in a cascade of delays, spreading each into an alternating sequence of length $2k$.) This is not surprising given the very simple nature of the heuristic. It is not difficult to construct counterexamples in which the maximum delay of the parity heuristic is arbitrarily large relative to an optimal solution. We will show, however, that the parity heuristic is asymptotically optimal in a uniform, steady-state scenario (to be made precise below).

Consider a traffic crossing pattern on the grid. Let m_x and m_y denote the numbers of vertical and horizontal lanes, respectively. Each lane is assigned a direction arbitrarily (up or down for vertical lanes and left or right for horizontal). Let R denote a $W \times W$ square region of the grid containing all the intersections (see Fig. 3(a)). In order to study the behavior of the system in steady-state, we will imagine that R is embedded on a torus, so that vehicles that leave R on one side reappear instantly in the same lane on the other side (see Fig. 3(b)). Equivalently, we can think of this as a system of infinite size by tiling the plane with identical copies (see Fig. 3(c)). We assume that W is even.

If the system is sufficiently dense, the maximum delay of the system will generally grow as a function of time. Given a scheduling algorithm and a discrete traffic crossing, define its *delay rate* to be the maximum delay after t time units divided by t. Define the *asymptotic delay rate* to be the limit supremum of the delay rate for $t \to \infty$. Our objective is to show that, given a suitably uniform traffic crossing instance on the torus, the asymptotic delay rate of the parity algorithm is optimal.

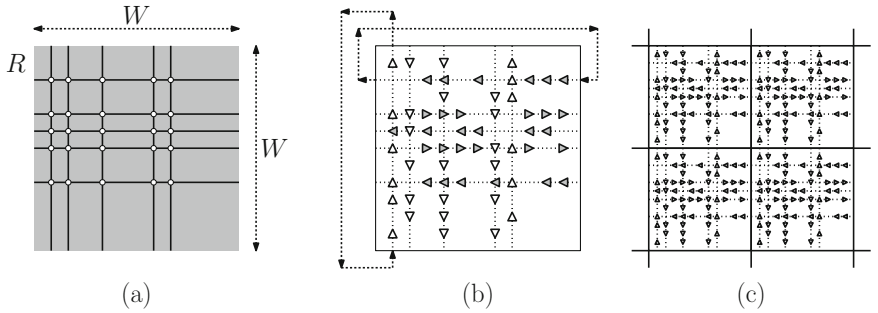

Fig. 3. Analysis of the Parity Heuristic

We say that a traffic crossing on the torus is *uniform* if every lane (within the square R) has an equal number of vehicles traveling on this lane. Letting n' denote this quantity, the total number of vehicles in the system is $n = n'(m_x + m_y)$. (The total number of positions possible is $W(m_x + m_y) - m_x m_y$, and so $n' \leq W - m_x m_y/(m_x + m_y)$.) The initial positions of the vehicles within each of the lanes is arbitrary. Let $p = n'/W$ denote the density of vehicles within each lane. Let $\rho_\infty^{\text{par}} = \rho_\infty^{\text{par}}(W, p, m_x, m_y)$ denote the worst-case asymptotic delay rate of the parity heuristic on any uniform discrete traffic crossing instance of the form described above, and let $\rho_\infty^{\text{opt}} = \rho_\infty^{\text{opt}}(W, p, m_x, m_y)$ denote the worst-case asymptotic delay for an optimum scheduler.

Our approach will be to relate the asymptotic performance of parity and the optimum to a parameter that describes the inherent denseness of the system. Define $\chi = \max(0, 2p - 1)$ to be the *congestion* of the system. Observe that $0 \leq \chi \leq 1$, where $\chi = 0$ means that the density is at most $1/2$ and $\chi = 1$ corresponds to placing vehicles at every available point on every lane (which is not really possible given that $n' < W$). To demonstrate that the parity heuristic is asymptotically optimal in this setting, it can be shown that $\rho_\infty^{\text{par}} \leq \chi/(1+\chi) \leq \rho_\infty^{\text{opt}}$. This is a consequence of the following two lemmas, whose proofs are omitted due to space constraints.

Lemma 2. *Given any uniform traffic crossing instance on the torus with congestion χ, $\rho_\infty^{\text{par}} \leq \chi/(1 + \chi)$.*

Lemma 3. *Given any uniform traffic crossing instance on the torus with congestion χ, $\rho_\infty^{\text{opt}} \geq \chi/(1 + \chi)$.*

While the proofs are somewhat technical, the intuition behind them is relatively straightforward. If $\chi = 0$, then while local delays may occur, there is sufficient capacity in the system for them to dissipate over time, and hence the asymptotic delay rate tends to zero as well. On the other hand, if $\chi > 0$, then due to uniformity and the cyclic nature of the system, delays will and must grow at a predictable rate. As an immediate consequence of the above lemmas, we have the following main result of this section.

Theorem 3. *Given a uniform traffic crossing instance on the torus, the asymptotic delay rate of the parity heuristic is optimal.*

References

1. Arkin, E.M., Mitchell, J.S.B., Polishchuk, V.: Maximum thick paths in static and dynamic environments. Comput. Geom. Theory Appl. **43**(3), 279–294 (2010)
2. Au, T.-C., Stone, P.: Motion planning algorithms for autonomous intersection management. In: Bridging the Gap Between Task and Motion Planning (2010)
3. Berger, F., Klein, R.: A traveller's problem. In: Proc. 26th Annu. Sympos. Comput. Geom., SoCG 2010, pp. 176–182. ACM, New York (2010)
4. Carlino, D., Boyles, S.D., Stone, P.: Auction-based autonomous intersection management. In: 2013 16th International IEEE Conference on Intelligent Transportation Systems-(ITSC), pp. 529–534. IEEE (2013)
5. Clarke, G., Wright, J.W.: Scheduling of vehicles from a central depot to a number of delivery points. Operations Res. **12**(4), 568–581 (1964)
6. Dantzig, G.B., Ramser, J.H.: The truck dispatching problem. Management Sci. **6**(1), 80–91 (1959)
7. Dresner, K., Stone, P.: Multiagent traffic management: a reservation-based intersection control mechanism. In: Proc. Third Internat. Joint Conf. on Auton. Agents and Multi. Agent Syst., pp. 530–537. IEEE Computer Society (2004)
8. Dresner, K., Stone, P.: Multiagent traffic management: an improved intersection control mechanism. In: Proc. Fourth Internat. Joint Conf. on Auton. Agents and Multi. Agent Syst., pp. 471–477. ACM (2005)
9. Dresner, K.M., Stone, P.: A multiagent approach to autonomous intersection management. J. Artif. Intell. Res. **31**, 591–656 (2008)
10. Fenton, R.E., Melocik, G.C., Olson, K.W.: On the steering of automated vehicles: Theory and experiment. IEEE Trans. Autom. Control **21**(3), 306–315 (1976)
11. Fiorini, P., Shiller, Z.: Motion planning in dynamic environments using velocity obstacles. Internat. J. Robot. Res. **17**(7), 760–772 (1998)
12. Hearn, R.A., Demaine, E.D.: Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. Theo. Comp. Sci. **343**(1–2), 72–96 (2005)
13. Petti, S., Fraichard, T.: Safe motion planning in dynamic environments. In: 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005. (IROS 2005), pp. 2210–2215, August 2005
14. Rajamani, R.: Vehicle Dynamics and Control. Springer Science & Business Media, December 2011
15. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Res. **35**(2), 254–265 (1987)
16. Van Middlesworth, M., Dresner, K., Stone, P.: Replacing the stop sign: unmanaged intersection control for autonomous vehicles. In: Proc. Seventh Internat. Joint Conf. on Auton. Agents and Multi. Agent Syst., pp. 1413–1416. International Foundation for Autonomous Agents and Multiagent Systems (2008)
17. Wurman, P.R., D'Andrea, R., Mountz, M.: Coordinating hundreds of cooperative, autonomous vehicles in warehouses. The AI magazine **29**(1), 9–19 (2008)
18. Yu, J., LaValle, S.M.: Multi-agent path planning and network flow. In: Frazzoli, E., Lozano-Perez, T., Roy, N., Rus, D. (eds.) Algorithmic Foundations of Robotics X. Springer Tracts in Advanced Robotics, vol. 86, pp. 157–173. Springer, Heidelberg (2013). http://dx.doi.org/10.1007/978-3-642-36279-8_10