On the Complexity of an Unregulated Traffic Crossing*

Philip Dasler and David M. Mount
Department of Computer Science, University of Maryland

For submission to FWCG 2014 (Extended Abstract)

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.

We focus here on a problem, called the *Traffic Crossing Problem*, that involves coordinating the motions of a set of vehicles moving through an intersection. 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 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., [2]). 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 [5].

Prior work on autonomous vehicle control has generally taken a higher-level view (e.g., traffic routing [6]) or a lower-level view (e.g., control theory, kinematics, etc. [4]). We propose a mid-level view, focusing on the control of a vehicle over an interval spanning perhaps seconds to minutes. The work by Fiorini and Shiller on velocity obstacles [3] considers motion coordination in a decentralized context, in which a single moving agent is attempting to avoid other moving objects. Much closer to our approach is work on *autonomous intersection management* (AIM) [2]. The approach taken there focuses largely on the application of multi-agent techniques, and 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* [1]. A traveler must cross a horizontal strip (the river) but may jump onto rectangular moving carriers (floating logs) to reach a goal position on the opposite side of the strip.

We consider a very 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. We model the traffic network as a collection of axis-parallel lines, which represent roads. Vehicles are modeled as line segments moving monotonically along axis-parallel straight lines (traffic lanes) in the plane. Vehicles can alter their speed, subject to a maximum speed limit, but they cannot reverse direction, make turns, or change lanes. The objective is to plan the collision-free motion of these vehicles as each moves from a given start position to a desired goal position.

2 Problem Definition

A *traffic crossing* is defined as a set $C = (V, \delta_{max})$, which is comprised of a set $V = \{v_1, \ldots, v_n\}$ of n vehicles in the plane and a global speed limit $\delta_{max} \in \mathbb{R}^+$. 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 is associated with a start and goal position, denoted $p_i^{\vdash}$ and $p_i^{\dashv}$, respectively. It is also associated with a start time and deadline, denoted $t_i^{\vdash}$ and $t_i^{\dashv}$, respectively. The question is whether, subject to the speed limit, there exists a collision-free motion plan so that each vehicle travels monotonically from its start position and reaches its goal position prior to its deadline.

The motion of each vehicle is described by a function, called a *speed profile*, that defines the instantaneous speed of the vehicles at time t. Formally, the speed profile for the ith vehicle v_i is given as a function $\delta_i(t)$ of time. Given its speed profile, the position of a vehicle at time t, denoted $p_i(t)$, is $p_i^{\vdash} + d_i \left[\int_0^t \delta_i(x)dx\right]$. The vehicle inhabits

*Email: {mount,daslerpc}@cs.umd.edu. This work has been supported by the National Science Foundation under grant CCF-1117259 and the Office of Naval Research under grant N00014-08-1-1015.

the open line segment of unit length whose leading point is at $p_i(t)$, which we denote by $\sigma_i(t)$. Given a traffic crossing C, a set D of speed profiles is *valid* for C if:

$$\forall t \notin \left[t_i^{\vdash}, t_i^{\dashv}\right],\ \delta_i(t) = 0 \tag{1a}$$

$$\forall t \in \left[t_i^{\vdash}, t_i^{\dashv}\right],\ \delta_i(t) \in [0, \delta_{max}] \tag{1b}$$

$$\forall t \text{ and for } 1 \leq i < j \leq n,\ \sigma_i(t) \cap \sigma_j(t) = \emptyset \tag{1c}$$

$$p_i(t_i^{\dashv}) = p_i^{\dashv} \tag{1d}$$

These conditions state (respectively) that (1a) a vehicle may not move either prior to its start time nor after its deadline, (1b) does not reverse direction or exceed the speed limit, (1c) does not collide with other vehicles, and (1d) arrives at its goal position. The *Traffic Crossing* (*decision*) *Problem* is, given a traffic crossing C, does there exist a valid set of speed profiles for C.

3 Hardness of Traffic Crossing

Our first result states that the Traffic Crossing Problem is NP-Hard.

Theorem 3.1. *Given a Boolean formula F in 3-SAT, 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.*

Due to space limitations, we present a brief outline of the reduction. (See the full version of the paper for details.) A special set of *variable vehicles* are used to represent variables of the boolean formula. Through a careful assignment of the start and goal positions and start and deadline times together with the use of auxiliary *helper vehicles*, we constrain the legal motions of each vehicle to one of two types, either going at full speed to the goal, or delaying for a fixed time period and then going at full speed to the goal. These two choices are used to specify the truth value for each variable. Next, we define a *clause mechanism* to enforce clause satisfaction. Each makes use of a special *verifier vehicle*. The mechanism is designed so that the verifier vehicle must delay for five time units if all of the literals are false, but may delay for less if at least one is true. By an appropriate setting of the start and deadline times for each verifier vehicle, we enforce the condition that each clause must be satisfied. Finally, in order to guarantee that vehicles arrive at the appropriate times to the various mechanisms, we define a *delaying mechanism*.

4 One-Sided Traffic Crossing

In this section we show that it is possible to solve a constrained version of the Traffic Crossing Problem much more efficiently. The complexity of the generalized Traffic Crossing Problem comes from the interplay between the horizontal (east-west) and vertical (north-south) vehicles, which can result in a complex cascade of constraints. To break this interdependency, we consider a variant, called the *One-Sided Traffic Crossing Problem*, in which one direction of vehicles have their speeds fixed, and the other direction adjusts theirs to avoid collisions.

A one-sided traffic crossing instance for this variant is the same as described earlier, but the vertically-moving vehicles all move at the same, fixed speed from north to south. The horizontally-moving vehicles move from west to east, but their speeds are determined by the algorithm and may vary up to the maximum speed limit. The objective is to compute a valid speed profile for the horizontally-moving vehicles. We assume that the lines carrying the vehicles are disjoint, that is, there are no two vehicles in the same lane. Given such a instance, the problem is to determine whether such a speed profile exists.

Theorem 4.1. *The One-Sided Traffic Crossing Problem can be solved in $O(n \log n)$ time.*

The algorithm involves two applications of plane sweep. We begin by transforming the problem so that the each horizontally-moving vehicle is shrunken to a point, and each vertically-moving vehicle is represented as a rectangle. The first application of plane sweep constructs a set of *collision zones*. These are regions of the plane that have the property that any horizontally-moving vehicle whose leading point enters such a region cannot avoid intersecting at least one vertically-moving vehicle. The second plane sweep plans an optimal collision-free motion for each of the horizontally-moving vehicles by computing a time-minimum routing around these collision zones. We show that both sweeps run in $O(n \log n)$ time.

References

[1] F. Berger and R. Klein. A traveller's problem. In *Proceedings of the Twenty-sixth Annual Symposium on Computational Geometry*, SoCG '10, page 176182, New York, NY, USA, 2010. ACM.

[2] K. M. Dresner and P. Stone. A multiagent approach to autonomous intersection management. *J. Artif. Intell. Res.(JAIR)*, 31:591656, 2008.

[3] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, July 1998.

[4] R. Rajamani. *Vehicle Dynamics and Control*. Springer Science & Business Media, December 2011.

[5] P. R. Wurman, R. D'andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *The AI magazine*, 29(1):9–19, 2008.

[6] J. Yu and S. M. LaValle. Multi-agent path planning and network flow. *arXiv:1204.5717 [cs]*, April 2012.

On the Complexity of an Unregulated Traffic Crossing*

Philip Dasler and David M. Mount
Department of Computer Science
University of Maryland
College Park, Maryland 20742
{mount,daslerpc}@cs.umd.edu

For submission to FWCG 2014 (Full Version)

Abstract

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.

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 monotonically from its start position and reaches its goal position prior to its deadline.

We present two results. First, we show that this problem is NP-Hard, by 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.

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.

*This work has been supported by the National Science Foundation under grant CCF-1117259 and the Office of Naval Research under grant N00014-08-1-1015.

Traffic congestion is a complex and pervasive problem with significant economic ramifications. Our intent is to identify simple and clean formulations of algorithmic problems that may serve as the low-level tools for more inclusive solutions. We focus here on a problem, called the *traffic crossing problem*, that involves coordinating the motions of a set of vehicles moving through an intersection. 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 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., [8]). Even though such systems may be beyond the present-day automotive technology, the approach can be applied to the controlling the motion of parcels and vehicles in automated warehouses [16].

Prior work on autonomous vehicle control has generally taken a higher-level view (e.g., traffic routing [4,5,14,17]) or a lower-level view (e.g., control theory, kinematics, etc. [10,13]). 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 moving agent is attempting avoid other moving objects. Much closer to our approach is work on *autonomous intersection management* (AIM) [1,3,6–9,15]. 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* [2]. A traveller must cross a horizontal strip (the river) but may jump onto rectangular moving carriers (floating logs) to reach a goal position on the opposite side of the strip.

We consider a very 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. We model vehicles as line segments moving monotonically along axis-parallel straight 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 two results. First, we show in Section 3 that this problem is NP-hard, by a reduction from 3-SAT. Second, in Section 4 we consider a constrained version in which cars traveling horizontally can alter their speeds while cars traveling vertically cannot. 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.

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 any collisions between them and they must do so in a reasonable amount of time. This time-based restriction exists to encourage an improvement in efficiency over the traffic light regulated crossing. Here, a "reasonable amount of time" is short enough that the traffic cannot simply take turns

crossing the intersection (i.e., using the manner in which a traffic light regulates intersections) but instead forces some amount of simultaneity.

The Traffic Crossing Problem can be posed either as one of optimization (e.g., how quickly can all the cars get across without colliding) or as a decision problem (e.g., can all vehicles cross, collision-free, within a particular time limit). Here, it is treated as a decision problem so that its parallels to problems like that of Satisfiability can be more easily illustrated.

2.1 Formal Definition

A traffic crossing is defined as a set $C = (V, \delta_{max})$. This set is comprised of a set of n vehicles $V \in \mathbb{R}^2$ and a global speed limit $\delta_{max} \in \mathbb{R}^+$. 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], defined as follows:

l_i**:** The length of the vehicle's line segment.

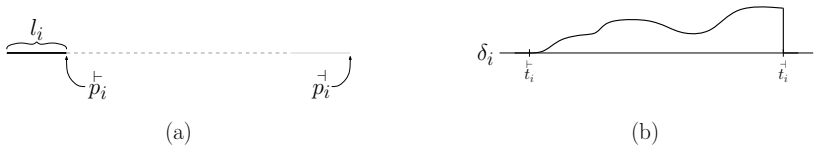
$p_i^{\vdash}$**:** The starting position of the vehicle, i.e., the vehicle's position prior to its start time (see below). The position is defined as a point and represents the leading edge of the vehicle.

$p_i^{\dashv}$**:** The goal position of the vehicle. The vehicle is considered to have successfully crossed the intersection if its leading point reaches this position either on or before its deadline (see below).

$t_i^{\vdash}$**:** The starting time of the vehicle. The vehicle may not move prior to this time.

$t_i^{\dashv}$**:** The deadline for the vehicle. This is an absolute point in time by which the vehicle must reach 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.



(a) (b)

Figure 1: (a) The physical specification of a vehicle v_i. (b) A possible speed profile, δ_i, for a vehicle v_i.

Formally, this set of functions is defined as $D = \{\delta_i(t) \mid i \in [1, n], \forall t, 0 \leq \delta_i(t) \leq \delta_{max}\}$. Given its speed profile, the position of a vehicle at time t is $p_i(t) = p_i^{\vdash} + d_i \left[\int_0^t \delta_i(x)dx\right]$ and the vehicle v_i inhabits the open line segment between $p_i(t)$ and $p_i(t) - d_i l_i$, which we denote by $\sigma_i(t)$.

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

A set D of speed profiles is valid if:

$$\forall t \notin \left[t_i^{\vdash}, t_i^{\dashv}\right] \quad \delta_i(t) = 0 \tag{1a}$$

$$\forall t \in \left[t_i^{\vdash}, t_i^{\dashv}\right] \quad \delta_i(t) \in [0, \delta_{max}] \tag{1b}$$

$$\forall t \text{ and } \forall v_j, v_i \in V : v_j \neq v_i, \;\; \sigma_i(t) \cap \sigma_j(t) = \emptyset \tag{1c}$$

$$p_i(t_i^{\dashv}) = p_i^{\dashv} \tag{1d}$$

Equation (1a) states that the vehicle may not move either prior to its start time nor after its deadline has passed. Equation (1b) enforces the speed limit and prevents vehicles from traveling in reverse. Equation (1c) prohibits collisions. Equation (1d) enforces the goal condition.

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

3 Hardness of Traffic Crossing

In this section, we will show that determining whether a traffic crossing is solvable is NP-Hard. In particular, we prove the following theorem:

Theorem 3.1. *Given a Boolean formula F in 3-SAT, there exists a traffic crossing $C = (V, \delta)$, computable in polynomial time, such that F is satisfiable iff there exists a valid set of speed profiles D for C.*

The input to the reduction is a boolean formula F in 3-CNF. Let $\{z_1, \ldots, z_n\}$ denote its variables and $\{c_1, \ldots, c_m\}$ denote its clauses. What follows is a high level overview of the reduction, with a more detailed description given further below.

Each variable z_i in F is represented by a pair of vehicles whose motion are constrained to one of two possible states. Then, for each clause $c_i \in F$, we create a mechanism that forces a collision if and only if all three literals of c_i are **False**. Finally, extra vehicles are added to carry the truth values of each variable to the appropriate clause mechanisms. For these clause mechanisms to function properly, the requisite vehicles must arrive at prescribed times. For this, one final mechanism is introduced that adjusts the relative timing of the vehicles. In the end, we will show that the original formula F is satisfiable if and only if the established traffic crossing is solvable, thus illustrating that the Traffic Crossing Problem is NP-Hard.

All vehicles in the reduction (except where noted) are of unit length and 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 a mechanism will be introduced to constrain the delay to only one of two types.

3.1 Variable Representation

Each variable z_i is represented by a pair of vehicles that encode the truth values for both the variable and its negation. The vehicles in this pair, referred to as *value vehicles*, travel downward in a coordinated manner along two vertical lines that are separated by the unit distance. As mentioned above, a vehicle can endure a delay in the interval $[0, 1]$. Through the use of additional

vehicles, when and how this delay occurs will be constrained. The vehicles "carry" a truth value based on their movement through the system, and doing so requires limiting the vehicles' movements to one of two states. In particular, each value vehicle can either delay for 1 time unit and then proceed at full speed to the goal or proceed at full speed directly to its final destination, arriving 1 time unit before its deadline. These two movement types will be referred to as delay-first (**True**) and delay-last (**False**) policies, respectively.

In order to constrain the delay policies of the value vehicles, two pairs of helper vehicles are added. The first pair forces one (and only one) of the value vehicles to incur an immediate unit delay. The second helper pair forces the remaining value vehicle to delay only at the end of its path.

The helper vehicles in a pair travel together horizontally, are separated vertically by the unit distance, and are placed so that they intersect the value vehicles' paths. Their goal positions, start times, and end times are all set so that an interaction occurs between them and the value vehicles.

Thus, let (x, y) and $(x + 1, y)$ denote the positions of the leading points of a value vehicle pair (v_1, v_1') at time t (see Fig. 2 and Remark A explaining it). Place a helper vehicle pair (u_1', u_1) at (x, y) and $(x, y + 1)$, respectively, set their goal positions to $(x + 2, y)$ and $(x + 2, y + 1)$, their start times to t, and their deadlines to $t + 3$. Similarly, place another helper vehicle pair (w_1', w_1) at $(x, y+\Delta)$ and $(x, y+1+\Delta)$, respectively, set their goal positions to $(x+2, y+\Delta)$ and $(x+2, y+1+\Delta)$, their start times to $t + \Delta$, and their deadlines to $t + 3 + \Delta$. The value of Δ is, essentially, arbitrary and is used here to illustrate that the distance traveled by the value vehicles does not affect their selection of and adherence to one of the two prescribed movement policies.

Lemma 3.1. *Given the pairs (v_1, v_1'), (u_1', u_1), and (w_1', w_1) as defined above, the value vehicles v_1 and v_1' must each adopt one of the following two movement policies: (a) delay for exactly 1 unit of time and then move beyond the paths of (u_1', u_1) at speed δ_{max} (i.e., delay-first); or (b) move beyond the paths of (w_1', w_1) at δ_{max} without delay (i.e., delay-last). Additionally, v_1 and v_1' may not select the same policy.*

Proof: First, notice that because v_1 and u_1' are in the same position at time t and are traveling toward each other, they will collide if neither one delays. Instead, they must choose different movement profiles so that one delays first, allowing the other to pass. This delay must be exactly 1 time unit long. Any longer and the delaying vehicle would miss its deadline; any shorter and there would not be sufficient time for the other vehicle to pass (traveling at δ_{max}, a vehicle of length 1 requires this much time).

Second, notice that a delay of v_1' necessitates a similar delay of u_1'. This is because it takes 1 time unit for u_1' to reach the point at which their paths intersect. If v_1' was to delay 1 time unit yet u_1' was to leave immediately, they would reach this point simultaneously and collide.

Given that u_1' must delay if v_1' does and v_1 cannot enact the same movement policy that u_1' does, it must be the case that both value vehicles cannot choose to delay for 1 time unit at this point. A similar dependency exists between the value vehicles and u_1, though this dependency prevents v_1 and v_1' from both leaving immediately.

The logic above also holds for the second helper pair, (w_1', w_1), constraining the value vehicles to opposing movement policies until they have moved beyond the paths of the helper vehicles. This also prevents the value vehicles from swapping movement policies. To do so would require the lagging vehicle (i.e., the vehicle that adopted the delay-first policy) to speed up while the

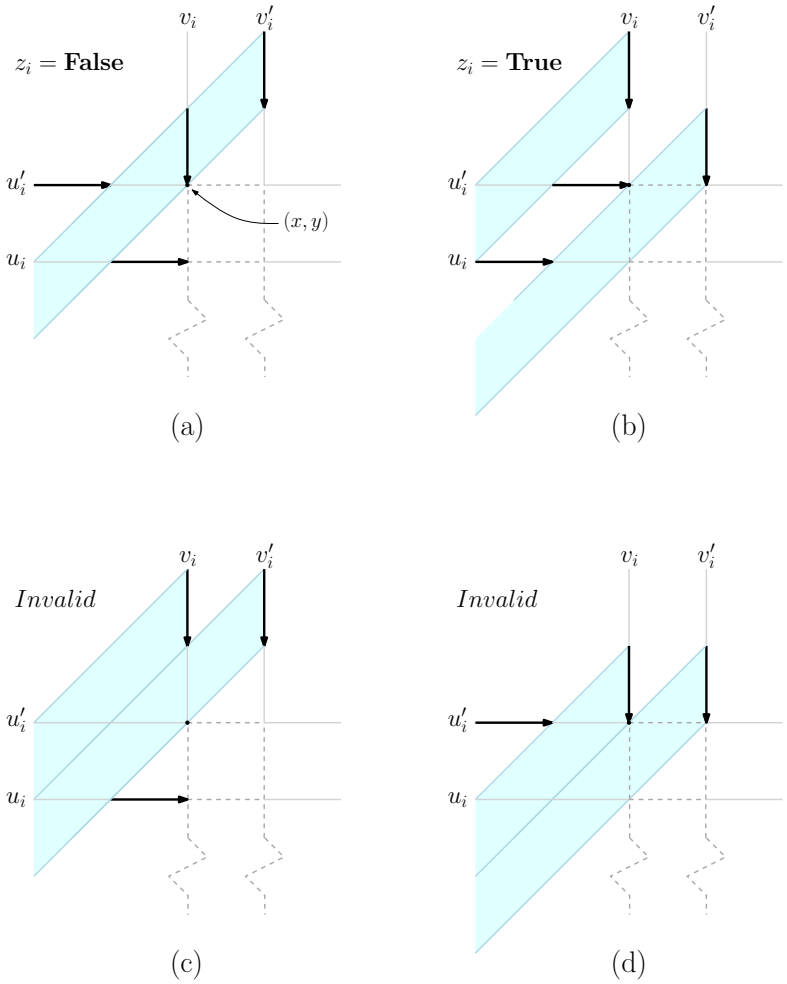


Figure 2: (a/b) Value vehicles taking on opposing values, allowing for valid paths for the helper vehicles. (c/d) If both value vehicles select the same delay policy, then there is no valid speed profile for one of the helper vehicles. (See Remark A on figure layouts.)

lead vehicle slows down. However, given the constraints placed on the vehicles, they are already traveling at the speed limit δ_{max}, so the lagging vehicle may not go any faster. □

To represent all of the variables in $\{z_1, \ldots, z_n\}$ we create multiple instances of the mechanism described above, one for each variable. These instances are lined up, one in front of the other, to form a common variable stream (see Fig. 3). The value vehicles' positions are initialized so that each member in a pair is colinear with the respective members of all other pairs of value vehicles. Additionally, the starting positions are spaced $s \geq 7$ units apart. This padding is to allow for the later insertion of a mechanism that regulates the timing of truth values flowing through the system.

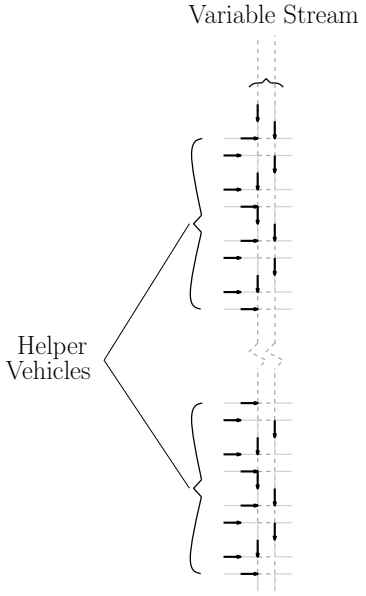


Figure 3: An example of value vehicles arranged into a variable stream representing four variables.

The variable stream is conceptually divided into blocks of length $s|V|$, long enough to accommodate all of the value vehicles and their requisite spacing. Every clause in F is associated with two of these blocks (one for the positive literals and one for the negative literals), requiring $2|C|$ such blocks (see Fig. 4). Two extra blocks are added, one at either end of the variable stream, to accommodate the initialization of the value vehicles with the helper vehicles. Truth values for the appropriate literals will be copied and transferred out of each block to a mechanism which adjusts their relative timing. This adjustment prepares the vehicles for a final mechanism that validates the satisfaction of the associated clause.

So, given a formula F with $|C|$ clauses and $|V|$ variables, each variable z_i is represented by a vehicle v_i with the following parameters:

$$p_i^{\vdash} = (0, si) \tag{2a}$$

$$p_i^{\dashv} = (0, 2s|V|(|C|+1) + si) \tag{2b}$$

$$t_i^{\vdash} = 0 \tag{2c}$$

$$t_i^{\dashv} = 2s|V|(|C|+1) + 1 \tag{2d}$$

In addition, the vehicle v_i' is created with similar parameters, but shifted 1 unit to the right.

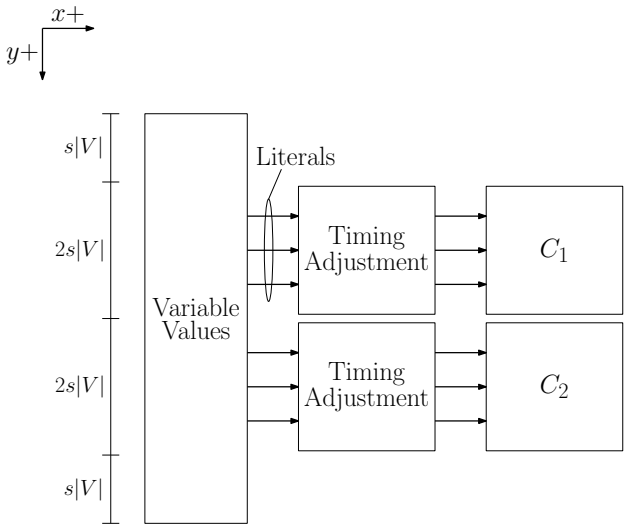


Figure 4: An overview of a reduction from 3-SAT to an instance of the Traffic Crossing Problem.

3.2 Value Transmission and Timing

For each clause, the three literal values will need to be carried to the appropriate clause mechanisms so that they arrive in the right place at the right time. This requires the introduction of two new mechanisms: one that copies truth values, and one that can adjust the timing of when a value reaches a particular location.

3.2.1 Value Duplication

In order to perform clause verification we will need the ability to transmit the variable values freely around our space. To do so, a new pair of parallel vehicles is created, separated by a distance of 1, whose purpose is to copy these values from the variable stream and carry them elsewhere. This pair is placed so that its starting position lies on the leftmost side of the variable stream, traveling to the right, and its start time $t_i^{\vdash}$ is the time at which the leading edge of the appropriate value pair reaches the vertical position of the uppermost vehicle (see Fig. 5). Just like the helper vehicles, each of these copy vehicles has their deadlines set so that they may delay for 1 time unit at most and because of this, the vehicles become a negative copy of the original value vehicles, with the negation on top and the original variable value on the bottom. We can continue to copy these values in order to carry them through the traffic space, taking orthogonal turns each time we do so. Any copies along this path that travel vertically will carry the variable's value on the left and the negation on the right. Any horizontal copy carries the negation on top and the original value below.

Each of a clause's positive literal values will be copied off of the variable stream simultaneously. The negative literals are copied similarly. By chaining vehicle copies across the space we can route the literal values to any location as necessary.

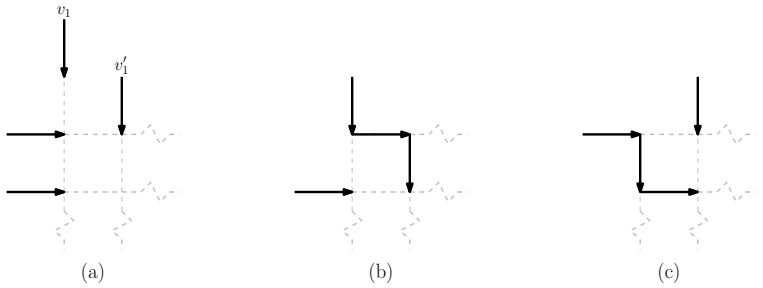


Figure 5: (a) An example of transferring a truth value at start time $t_i^{\vdash}$ for the copying vehicles. In this example, the variable z_1 is **True**, making v_1 and v_1' **True** and **False**, respectively. (b) At time $t_i^{\vdash} + 1$, notice that in the orthogonal copy the upper vehicle will take on the value of the negation while the lower vehicle takes the original value. (c) Another example of a value transfer at time $t_i^{\vdash} + 1$, but with $z_1 =$ **False**.

3.2.2 Timing and Delays

The routing of values may require that they travel different distances to reach certain points. By the structure of our reduction, except when stopped, all vehicles travel at the same speed. Because of this, any difference in path length will cause a difference in timing that may need to be corrected. This is done through the introduction of a delay mechanism. This mechanism is inserted into the path of every copy coming off of the variable stream and can be configured to delay a vehicle pair's leading edge by an arbitrary amount. This delay does not affect the values carried by the vehicles. Essentially, the value is routed through an S shape in the mechanism, doubling back on itself (see Fig. 6). The size of this S determines the extra distance that must be traveled and thus the total amount of delay. A parameter d represents the extra distance added to the S in order to tune the mechanism, leading to a delay of $2d$ (as described below). Vehicle pairs are arranged in the mechanism as follows, with the first and last referred to as the *incoming pair* and *outgoing pair*, respectively:

- (x, y) and $(x, y + 1)$ at time t,
- $(x + 2 + d, y)$ and $(x + 3 + d, y)$, with a start time of $t + 2 + d$,
- $(x + 3 + d, y + 2)$ and $(x + 3 + d, y + 3)$, with a start time of $t + 4 + d$,
- $(x, y + 2)$ and $(x + 1, y + 2)$, with a start time of $t + 6 + 2d$,
- $(x, y + 4)$ and $(x, y + 5)$, with a start time of $t + 8 + 2d$,
- $(x + 5 + d, y + 5)$ and $(x + 6 + d, y + 5)$, with a start time of $t + 13 + 3d$,
- and $(x + 5 + d, y)$ and $(x + 5 + d, y + 1)$, with a start time of $t + 17 + 3d$.

The distance between the incoming vehicle pair and the outgoing vehicle pair is $5 + d$, so, if the incoming pair were to continue on, both pairs would be in the same position at $t + 5 + d$. Since the outgoing pair starts at time $t + 17 + 3d$, the mechanism induces a delay in the transmission of the incoming pair of $12 + 2d$. By adding the delay mechanism to all copies made from the variable stream, we can adjust the relative timing of each vehicle pair by adjusting the value of d in each delay mechanism.

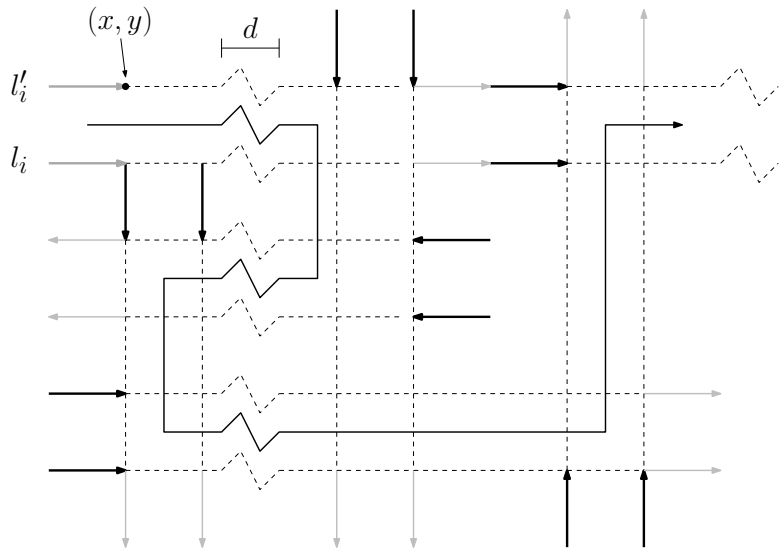


Figure 6: A diagram tracing the path a single pair of truth values (z'_i, z_i) take through the delay mechanism. The dotted vertical line represents where the mechanism can be expanded, separating the vehicles on either side by a distance of d and thus increasing the induced delay.

3.3 Clause Satisfaction

Clause satisfaction is verified with a mechanism that forces two particular vehicles to collide if all of the literals are **False**. The mechanism consists of two parts: one for the positive literals and one for the negative literals. Each part contains vehicle pairs representing the literals and their negations, blocking vehicles to appropriately constrain movement, and a verifying vehicle. If the set of literals do not satisfy the clause, each verifying vehicle is constrained to a single speed profile and they will collide. First, we look at the half that verifies the negative literals.

Define a point $r = (x, y)$ to be a reference point from which all other positions will be defined at a reference time t (see Fig. 7). Next, assume three pairs of incoming vehicles $(l'_1, l_1), (l'_2, l_2)$, and (l'_3, l_3), each a copy of the appropriate variables. These pairs travel horizontally, 1 unit apart vertically, with their leading edges 4 units behind the previous pair. Thus, the leading edges of the pairs are (x, y) and $(x, y+1)$, $(x-4, y+2)$ and $(x-4, y+3)$, and $(x-8, y+4)$ and $(x-8, y+5)$.

Next, place two blocking vehicles at $(x-2.5, y+1.5)$ and $(x-6.5, y+3.5)$. These vehicles have a start time of t, travel horizontally to the right, and have their deadlines set so that they must travel at δ_{max} with no delays. Finally, place a verifying vehicle v at (x, y) with a start time of t and traveling downward. The deadline for the verifying vehicle is set so that it can delay up to 5 time units.

Lemma 3.2. *Given the vehicle pairs, blocking vehicles, and verifying vehicle defined above, the verifier must delay for* 5 *time units if all of the literals are* **False** *but may delay for less if at least one is* **True**.

Proof: First, notice that every horizontal vehicle in the mechanism is on a possible collision course with the verifying vehicle. Thus, if the slope of the line between one of these vehicles and the verifying vehicle has a magnitude of 1 (or if their positions are equal), the vehicle will collide with

the verifying vehicle v if both continue without delay.

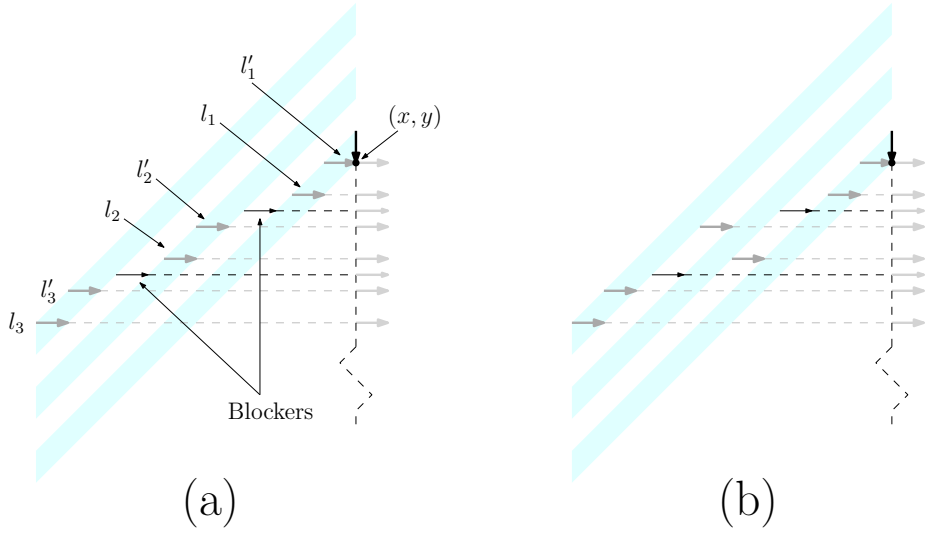


Figure 7: (a) The initialization of the negative half of a clause verifier for the clause $(\neg z_1 \vee \neg z_2 \vee \neg z_3)$ and with each variable $z_i =$ **True**. (b) The verifier with $z_2 =$ **False** and $z_1 = z_3 =$ **True**.

If each variable z_i is **True** then its negative copy l'_i is **False**, taking a delay-last movement policy. This places l'_1 at (x, y) and l_1 at $(x-1, y-1)$ at time t, which would lead to a collision with v. While l'_1 could still delay for 1 unit, l_1 no longer has this freedom as it has adopted the delay-first policy. Thus, to avoid a collision, v must delay for at least 1.

At time $t+1$, the first blocking vehicle has moved to $(x-1.5, y+1.5)$. The blocking vehicles' deadlines allow for no delay, so again v must delay.

At time $t+2$, l'_2 has moved to $(x-2, y+2)$ and l_2 has moved to $(x-3, y+3)$. Just as with l_1, v is forced to delay to avoid a collision.

At time $t+3$, the second blocking vehicle is at $(x-3.5, y+3.5)$, forcing another delay of v.

Finally, at time $t+4$, l'_3 has moved to $(x-4, y+4)$ and l_3 has moved to $(x-5, y+5)$, forcing one last delay of v.

Thus, if all of the variables z_i are **True**, making the negative literals l'_i all **False**, the verifying vehicle v must delay for 5 units of time in order to avoid a collision.

If any of the variables are **False**, their resultant copies l'_i and l_i will have shifted horizontal positions, no longer lying on the line of collision with v (i.e., their slopes are no longer magnitude 1), allowing v to delay for less than 5 units and slip between them. □

The positive half of the mechanism works in the same manner, with slight changes to the incoming literal vehicles and some added vehicles to account for these changes. First, the incoming literal pairs are not staggered with respect to each other but instead arrive with colinear leading edges and 1 unit apart (see Fig. 8(a)). Next, a copy of each literal pair is made, traveling downward. The first copy pair is placed at $(x+5, y)$ and $(x+6, y)$ and has a start time of $t+5$. The next pair is placed at $(x+3, y+2)$ and $(x+4, y+2)$ with a start time of $t+3$. The third pair is placed at $(x+1, y+4)$ and $(x+2, y+4)$ and has a start time of $t+1$.

Next, two blocking vehicles, traveling downward, are added at $(x+2.5, y+9.5)$ and $(x+4.5, y+5.5)$, both with a start time of $t+9$.

Finally, a verifying vehicle traveling to the right is added at $(x+1, y+12)$, with a start time of $t+9$ and deadline allowing for a delay of up to 5 time units. As before, the vehicle will be forced to delay for 5 time units if the clause is not satisfied by any of the positive literals.

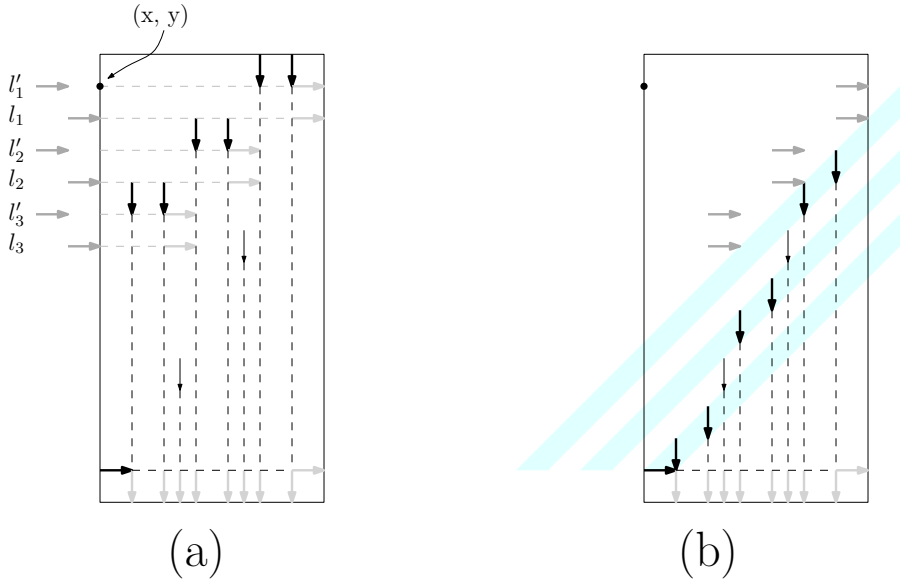


Figure 8: (a) The initialization of the positive half of a clause verifier for the clause $(z_1 \vee z_2 \vee z_3)$ and with each variable $z_i =$ **False**. (b) The verifier with $z_2 =$ **True** and $z_1 = z_3 =$ **False** at time $t+9$.

A clause will never have more than three literals, so it will never be the case that both the positive and negative halves of the clause verifier will have three literals. Blocking vehicles are added to take the place of missing literals in each half and their deadlines are set so that no delay is allowed. In this way, the verifying vehicles are still forced to delay for 5 units when their associated set of literals do not satisfy the clause.

The positive and negative halves of the mechanism are placed so that the paths of the verifying vehicles intersect. However, the time at which each half processes its literals may differ, dependent on which variables are being evaluated and the distance their values must travel to reach the mechanism. This can be compensated for in the delay mechanisms so that the verifying vehicles will collide with one another if both delay for 5 time units. In this way, if a clause is not satisfiable, a collision is inevitable, rendering the traffic crossing unsolvable. If the clause is satisfiable, one or both of the verifying vehicles will have at least two movement options, allowing them to avoid a collision.

3.4 Complete System Example

In the complete system, all of the variables are stacked on top of each other to form a variable stream. The appropriate literals are extracted, passed through a delay mechanism, and routed to their clause verifier halves. These mechanisms output a vehicle that will have delayed for 5 time

units if the variable assignments do not satisfy their respective clauses. The verifier vehicles from each clause will collide if neither set of literals satisfies them. An example of a 3-SAT reduction for the formula $(\neg z_1 \vee z_2 \vee \neg z_3)$ can be seen in Fig. 9.

3.5 Analysis of Translation Complexity

Every variable in the formula F requires $6n$ vehicles: one for the variable, one for its negation, and the two helper pairs. Next, when considering each of the m clauses, the greatest number of vehicles is necessary when all of the literals are positive. 15 are needed for the positive verifier, 9 for the negative verifier, 14 for each of the two delay mechanisms, and 12 for routing, for a total of at most 64 vehicles per clause. The complexity of translation is then $6n + 64m$ and is therefore on the order of $O(n + m)$.

4 A Solution to the One-Sided Problem

While the generalized Traffic Crossing Problem may be NP-Hard, it is possible to solve a constrained version of the problem much more efficiently. The complexity of the generalized Traffic Crossing Problem comes from the interplay between the two sets of vehicles. This interplay results in a complex cascade of constraints. To break this interdependency, one set of vehicles will have their speeds fixed as they travel through the intersection. In this version of the problem, referred to as the *one-sided problem*, this interplay is eliminated, and hence the horizontal vehicles can plan their motion based on full information of the constraints imposed by the vertical vehicles. For the sake of clarity, some other assumptions are made that simplify the formulation in a natural way while still displaying the salient issues.

First, we assume that the vertically traveling vehicles are invariant and are all traveling at the same speed, s_n. Given this invariance, the cascade of constraints leading to the original problem's NP-Hardness has been eliminated as the movements of the horizontal vehicles have become decoupled from each other (formerly being transmitted through the vertically traveling vehicles). This loss of dependence allows us to simplify the formulation of the problem, without loss of generality, by assuming all vehicles will approach the intersection either from the north or from the west. Finally, we assume that all vehicles are of length l and in general position.

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 $d_w = (1, 0)$. Thus, 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 controllable vehicles (i.e., 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 must be expanded, transforming each of them into a square obstacle with sides of length l (see Fig. 10) and with their left edges coincident with their original line segments.

Given the global speed limit δ_{max}, there are regions in front of each obstacle that, if a vehicle in W enters them, there is no possibility of escape before a collision occurs (this concept is similar to the obstacle avoidance work done in [12]). 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 downward and

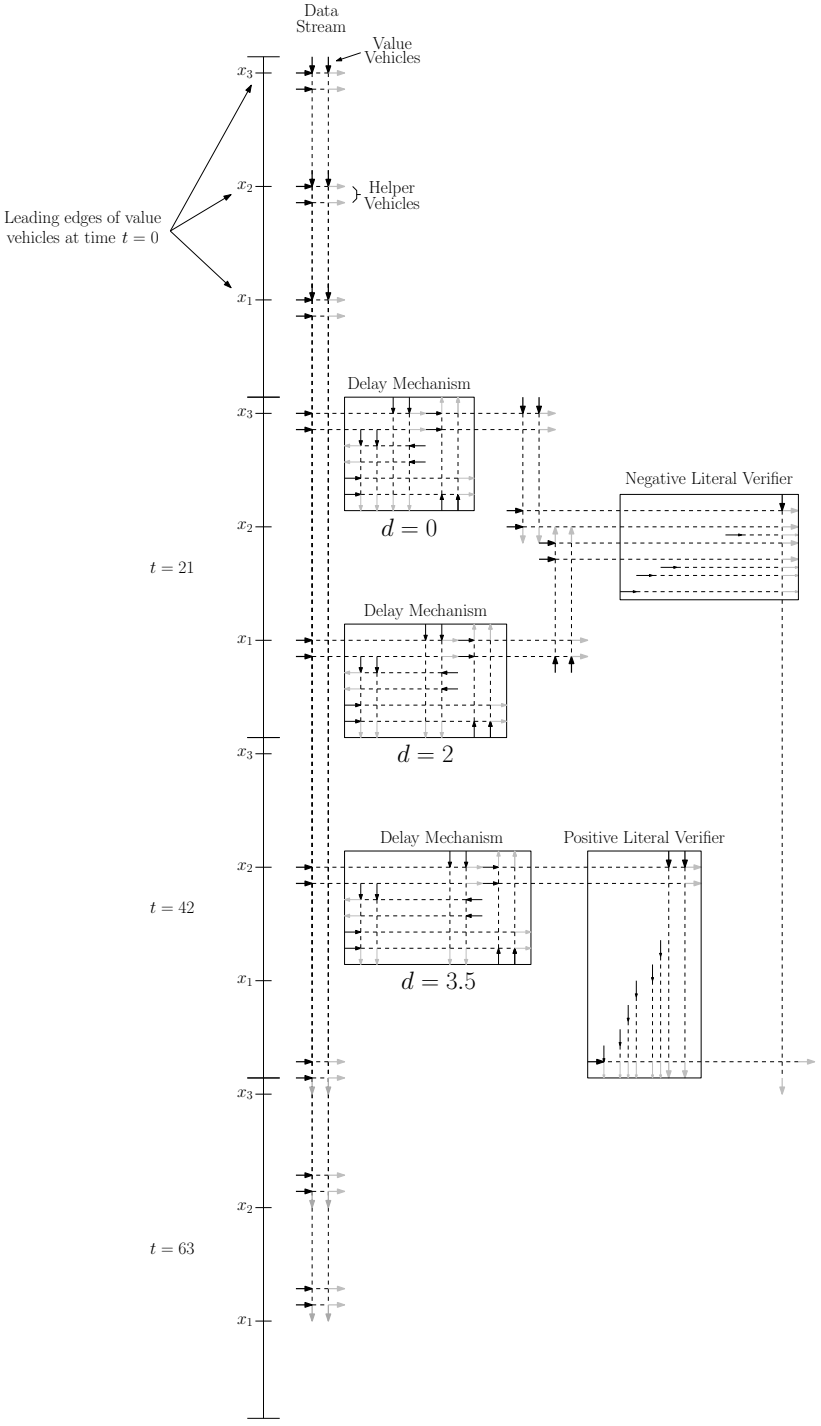


Figure 9: An example of a 3-SAT problem with $F = (\neg z_1 \vee z_2 \vee \neg z_3)$, expressed as a traffic crossing.

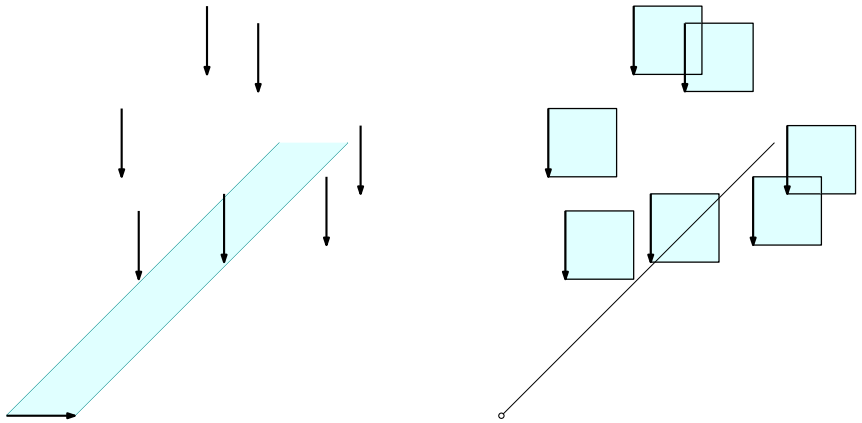


Figure 10: (a) A random traffic crossing problem as viewed from a single active vehicle. (b) The resulting space after the point transformation of the active vehicle problem in (a).

the rightmost point is extended downward and to the left 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.

When expanding the vehicles in N it is possible that the new rectangular obstacles may overlap one another, producing larger obstacles and, consequently, larger collision zones (see Fig. 11). These collision zones follow similar rules as those above, though the meaning of terms such as "leading edge," "leftmost," and "rightmost" are altered and, instead of applying locally, are now viewed across the entire set of contiguous obstacles. 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.

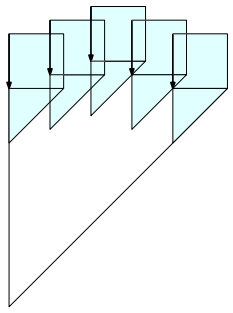


Figure 11: An example of merging obstacles. Here, the collision zones for each individual obstacle (represented as shaded triangles) are insufficient as the merger creates a larger area that vehicles must avoid (seen here as the unfilled triangle).

4.1 Merging Obstacles and Growing Collision Zones

The processing of obstacle mergers and the generation of collision zones will be 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 horizontal edges of every obstacle are stored, in order from

top to bottom, into an event list for the sweep line. Given $O(n)$ obstacles, this occurs in $O(n \log n)$ time as they are sorted. 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. Most of these slopes will be infinite (i.e., the edges are vertical) except for the rightmost edge of the collision zone that, at times, will have a slope of 1.

The event list must keep track of four different events, two of which are known *a priori* and two of which is generated during the sweep. The positions of the top and bottom edges of the obstacles are all know before hand and are added to the list at the start. The remaining event deals with the sloped edges of the collision zones. These edges begin at the bottom 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 will be added as the sweep line progresses through the obstacles.

Thus, the sweep line must handle the following events:

Top Edge Encounter - When the sweep line encounters the top edge of an obstacle it must either create a new interval or add this obstacle to an existing interval. The creation of a new interval is straightforward as the endpoints of the edge are all that need to be added (see Fig. 12(a)).

If the top edge intersects an existing interval, however, there is a little more work to be done. First, if the leftmost point of the edge does not lie within the interval then it becomes the new leftmost edge of the interval (see Fig. 12(b)). If the sloped edge of a collision zone has already formed for this contiguous block of obstacles (see **Bottom Edge Encounter** for a description of how these form), then the termination point of the sloped edge may need to be updated to account for a shift in the leftmost edge.

Second, the rightmost point of the encountered edge is inserted into the list of right edges in left-right order. The new edge may become the new rightmost edge and if the previous rightmost edge was sloped then it is removed from the edge list. For example, in Fig. 12(d) this has just occurred within the set of obstacles on the left. If the newly added right edge does not replace the sloped edge and the sloped edge intersects the newly added edge, the point at which they intersect is added to the list of events to be processed (this occurs in Fig. 12(c) on the right side). If there is an existing event in the event list for the sloped edges intersection with another obstacle, it must be deleted as the addition of the newest obstacle will truncate the edge before it reaches that event.

Bottom Edge Encounter - When the bottom edge of an obstacle is encountered, the obstacle's right edge is found in the interval's edge list. If it is not the rightmost, it is removed from the edge list (this occurs in Fig. 12(e) on the left, denoted by the grey slope arrow). If the edge to be removed is the rightmost edge in the list, rather than removing it, its slope is changed to that of the ratio between the vehicles' speed limit and the speed of the vehicles, $\frac{\delta_{max}}{s_n}$. Next, the termination point for this sloped edge is added to the event list. This is the point at which the leftmost edge of the interval and the sloped edge meet. This point is illustrated in Fig. 12(e), though it was added when the previous bottom edge was processed.

As noted above, this event may need to be updated if a top edge is encountered that moves the leftmost edge of this interval.

Sloped Edge Termination - When the sloped edge terminates against a right edge in the edge list, it is deleted from the edge list. This makes the edge with which it collided the new rightmost edge.

Interval Termination - In this case, the sloped edge of the collision zone has met the leftmost edge of the interval. When this is the case, the interval has finally closed and can be removed from the sweep line status (see Fig. 12(f)).

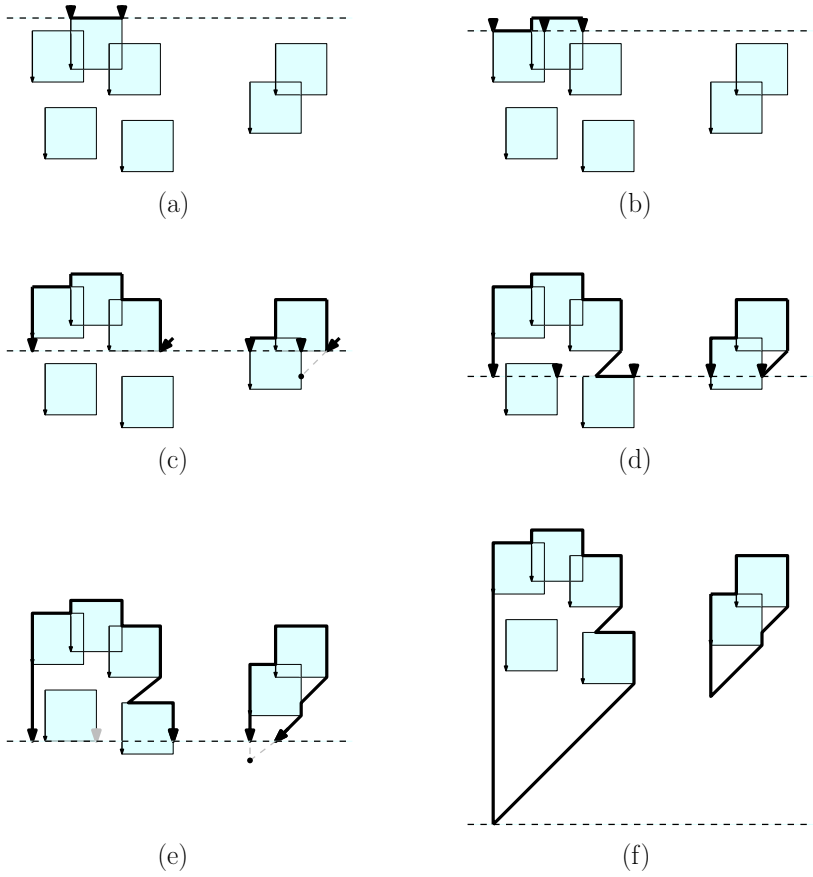


Figure 12: A sweep line merging obstacles and creating collision zones. Note: these illustrations do not show every step in the sweep line process. Some are skipped in order to save space. (a) Encountering the first top edge and adding an interval to the sweep line status. (b) Encountering the next top edge, which increases the interval size. (c) Encountering bottom edges changes the rightmost slope of the collision zone. Notice on the right that an internal right edge is stored in the status. (d) Sloped edges the top of an unprocessed obstacle and the rightmost edge of an obstacle in an interval. (e) Encountering the bottom edge of an internal obstacle. It's rightmost edge is deleted from the sweep line status. (f) Reaching the point of convergence for a collision zone. The interval is deleted from the sweep line status.

The initial population of the event list occurs in $O(n \log n)$ time as it requires the sorting of the top and bottom edges of the obstacles. As the sweep line progresses through the obstacle space, it must add and remove the right edges of obstacles to the appropriate intervals. These lists of right edges are built incrementally in sorted order, so adding to and removing from them requires only $O(\log n)$ time. Finally, as there are a constant number of possible events per obstacle (at most, each obstacle has 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.

4.2 Movement Planning

Once the obstacles have been merged and grown appropriately, we need to find speed profiles for each vehicle that allow them to safely cross the intersection. This is done using the same obstacle filled space we have been working with thus far, though with a small change in perspective. Currently, vehicles are only allowed horizontal movement and obstacles only move vertically. Instead, we will treat the obstacles as static objects and add a vertical component to the vehicles equal to the obstacles' speed. So, for example, a vehicle moving at the maximum speed will actual follow a path with a slope of $\frac{s_n}{\delta_{max}}$ where as a stationary vehicle will travel vertically. Again, though, we have scaled our axes so that this ratio is 1, imposing on the vehicle monotonic movement with a slope in the interval $[1, \infty]$. With this understanding, we can now easily find a path through the obstacles while obeying the speed constraints of the vehicles.

The vehicle will travel at its minimum slope (equivalent to its maximum speed) until it either reaches its goal position or encounters an obstacle. If the vehicle would collide with an obstacle, it comes to a stop and waits for the obstacle to pass before proceeding. Once this occurs, the vehicle continues on its way at its maximum speed until it has covered the distance to its goal (measured horizontally, as vertical movement no longer represents spatial translation).

The path created by the above behavior can be efficiently found through the use of another line sweep. First, notice that every edge that is locally to the left of an obstacle (referred to as a *left edge*) is a vertical line segment. Since the vehicles move monotonically, they will only ever encounter an obstacle at one of these left edges. So, to find a path for each vehicle traveling at speed δ_{max}, a sweep line perpendicular to the vehicles' trajectories is created and swept from the upper-right to the lower-left (see Fig. 13(a)). This perpendicular line's status will maintain a list of obstacle occlusions with respect to the vehicles' direction of travel by adding an interval for each obstacle as it is encountered during the sweep. More specifically, it stores the point where the sweep line first encountered the obstacle's left edge, the horizontal position of the left edge, and the point where the sweep line last encountered the edge.

During the sweep, a tree is built representing a set of all paths through the obstacle field that encounter an obstacle. Vehicles will either encounter an obstacle in the tree or are free to travel at full speed without collision until their goal is reached. Each obstacle is a vertex in the tree and edges represent the path taken after encountering this obstacle. The edge will either lead to an encounter with another obstacle or will lead to the root. The root is the only vertex which does not represent an obstacle but instead signifies an open path to the goal.

The event list for the sweep line is populated with the upper and lower ends of each left edge. Whenever an upper end is encountered, it is inserted into the list of intervals in the sweep line status and the obstacle is inserted into the path tree. If the insertion point does not lie within an

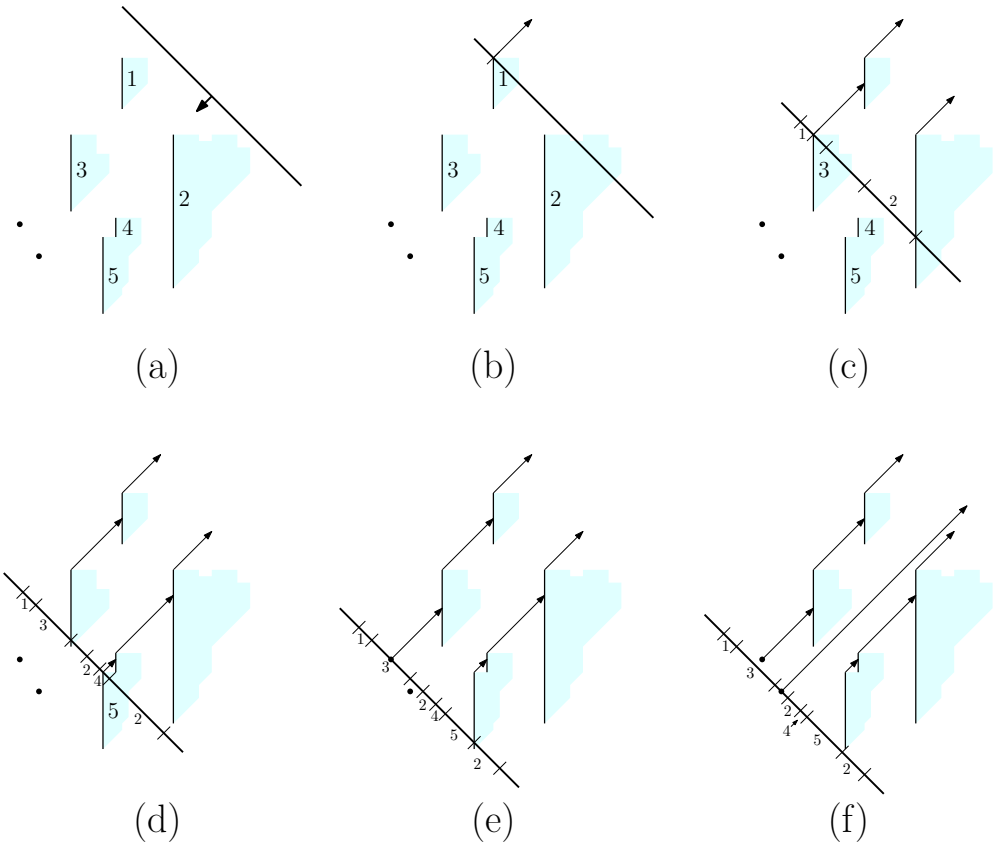


Figure 13: (a) A sweep line for path finding, traveling perpendicular to the direction of travel of a vehicle moving at speed δ_{max}. (b) The sweep line encountering vertical edge 1. As there is no interval on the sweep line where it occurs, this line's path goes directly to the goal at speed δ_{max}. Edges to the root of the path tree are represented by arrows going off to infinity. (c) The sweep line encountering edge 3. This encounter lies in the interval for edge 1. (d) Encountering edge 5, creating a path from it to edge 4. (e) Encountering the first vehicle, which lies in the interval for edge 3. Thus, the final path for the vehicle is to travel at maximum speed until it reaches edge 3, wait for the edge to pass, travel to 1, wait, and finally travel to the goal position. (f) The sweep line encountering the second vehicle at an open interval. Thus, this vehicle can travel at speed δ_{max} until it reaches its goal position.

existing interval, then an edge between the obstacle and the root is created (see Fig. 13(b)). If the insertion point lies within an interval, that interval is split by the inserted point and an edge between the new obstacle and the interval's obstacle is added (see Fig. 13(c)).

Whenever the lower end point of an obstacle's left edge is encountered, the interval ending for that obstacle is added to the list. If an event occurs before an interval has completed, the intervals intermediate size can be determined using the position of the sweep line, the start point of the interval, and the position of the obstacle's left edge (This occurs in Fig 13 between (c) and (d)). Finally, when a vehicle is encountered, its position along the sweep line determines its path. If it is an interval, then its path begins by traveling to the associated obstacle and, using the path tree, travels to that obstacle's parent obstacle, repeating this process until it has reached its goal position.

So, in the example in Fig 13(e), the upper vehicle encounters obstacle 3, waits for it to pass (i.e., travels vertically till the end is reached), moves at the maximum speed until it encounters obstacle 1, then continues on until it reaches its goal position. The lower vehicle, having been inserted into the interval list in between intervals, is free to travel at the maximum speed until its goal position is reached (see Fig. 13(f)).

4.3 Different Length Vehicles

While we have made the assumption that all vehicles must be of the same length, this solution proposed above can easily tolerate differing lengths in the invariant vehicles in N. In fact, no change is required at all. Neither the sweep line nor the movement planning policy have any dependencies on the vertical length of the obstacles. However, the problem becomes harder when the vehicles in W have differing lengths. This is due to the fact that the contraction/expansion of the problem space can not be shared among all of the vehicles. The naive solution is to simply rerun this step for every vehicle in W, leading to disappointing complexity on the order of $O(n^2 \log n)$. Of course, in reality having n cars does not imply n different lengths. Instead, cars could be bucketed according to their length as there are likely only a handful of meaningful lengths that need to be considered. Given this, the sweep line process would only need to occur a constant number of times (equal to the number of buckets) with no impact on the asymptotic complexity.

References

[1] Tsz-Chiu Au and Peter Stone. Motion planning algorithms for autonomous intersection management. In *Bridging the Gap Between Task and Motion Planning*, 2010.

[2] Florian Berger and Rolf Klein. A traveller's problem. In *Proceedings of the Twenty-sixth Annual Symposium on Computational Geometry*, SoCG '10, page 176182, New York, NY, USA, 2010. ACM.

[3] Dustin Carlino, Stephen D. Boyles, and Peter Stone. Auction-based autonomous intersection management. In *Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on*, page 529534. IEEE, 2013.

[4] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, August 1964.

[5] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, October 1959.

[6] Kurt Dresner and Peter Stone. Multiagent traffic management: A reservation-based intersection control mechanism. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, page 530537. IEEE Computer Society, 2004.

[7] Kurt Dresner and Peter Stone. Multiagent traffic management: An improved intersection control mechanism. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, page 471477. ACM, 2005.

[8] Kurt M. Dresner and Peter Stone. A multiagent approach to autonomous intersection management. *J. Artif. Intell. Res.(JAIR)*, 31:591656, 2008.

[9] David Fajardo, Tsz-Chiu Au, S. Travis Waller, Peter Stone, and David Yang. Automated intersection control: Performance of future innovation versus current traffic signal control. *Transportation Research Record: Journal of the Transportation Research Board*, 2259(-1):223–232, December 2011.

[10] R. Fenton, G. Melocik, and K. Olson. On the steering of automated vehicles: Theory and experiment. *IEEE Transactions on Automatic Control*, 21(3):306–315, June 1976.

[11] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, July 1998.

[12] S. Petti and T. Fraichard. Safe motion planning in dynamic environments. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005. (IROS 2005)*, pages 2210–2215, August 2005.

[13] Rajesh Rajamani. *Vehicle Dynamics and Control*. Springer Science & Business Media, December 2011.

[14] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, March 1987.

[15] Mark VanMiddlesworth, Kurt Dresner, and Peter Stone. Replacing the stop sign: Unmanaged intersection control for autonomous vehicles. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, page 14131416. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[16] Peter R. Wurman, Raffaello D'andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *The AI magazine*, 29(1):9–19, 2008.

[17] Jingjin Yu and Steven M. LaValle. Multi-agent path planning and network flow. *arXiv: 1204.5717 [cs]*, April 2012.

A Remarks

Remark Since vehicle dynamics and timing differences are difficult to understand in static images, some visualization conventions are used throughout this paper to convey these time dependent properties. First, delays in a vehicle's movement are visualized by displacing the vehicle by a distance equivalent to the delay. For example, a vehicle placed 1 distance unit behind its starting position represents a delay of 1 time unit (see Fig. 14(a)). This positional change is equivalent to a 1 unit delay as it takes the vehicle this long to reach its original position when traveling at the maximum speed (which the vehicles must do in order to reach their goals in time). Additionally, we can visualize a vehicle's path in relation to another vehicle traveling perpendicularly by projecting one of the vehicles along the resultant vector of their combined motions (thus the shaded region in Fig. 14(b)). The deflection of this band is based on the ratio of the two vehicles' speeds. If this shaded band intersects both vehicles then a collision will occur between them.

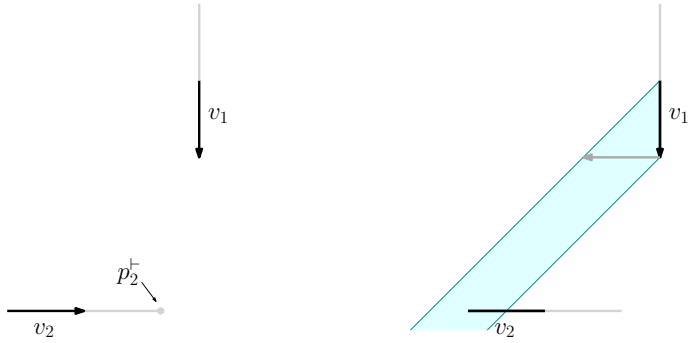


Figure 14: (a) A pair of vehicles traveling toward each other. Both v_1 and v_2 have an allowable delay of 1 time unit, though only v_2 has actually done so. (b) The motion of v_1 projected forward in time. Notice that a collision will occur with v_2, which would have been avoided if v_2 had not delayed.