Virtualized Traffic: *Reconstructing Traffic Flows from Discrete Spatio-Temporal Data*

Jason Sewall, Jur van den Berg, Ming Lin, Dinesh Manocha

Abstract—We present a novel concept, *Virtualized Traffic*, to reconstruct and visualize continuous traffic flows from discrete spatio-temporal data provided by traffic sensors or generated artificially to enhance a sense of immersion in a dynamic virtual world. Given the positions of each car at two recorded locations on a highway and the corresponding time instances, our approach can reconstruct the traffic flows (i.e. the dynamic motions of multiple cars over time) in between the two locations along the highway for immersive visualization of virtual cities or other environments. Our algorithm is applicable to high-density traffic on highways with an arbitrary number of lanes and takes into account the geometric, kinematic, and dynamic constraints on the cars. Our method reconstructs the car motion that automatically minimizes the number of lane changes, respects safety distance to other cars, and computes the acceleration necessary to obtain a smooth traffic flow subject to the given constraints. Furthermore, our framework can process a continuous stream of input data in real time, enabling the users to view virtualized traffic events in a virtual world as they occur. We demonstrate our reconstruction technique with both synthetic and real-world input.

Index Terms—Animation, Virtual Reality, Kinematics and dynamics

1 Introduction

With better sensing and scene reconstruction technology and more on-line software tools, such as Google Maps and Virtual Earth, for visualizing urban scenes, there is a growing need to introduce realistic street traffic in virtual worlds. One natural approach is to incorporate a traffic simulator in a virtual environment. There are numerous techniques to simulate macro- and microscopic traffic [1], including agent-based methods [2], [3], cellular automata [4], [5], mathematical modeling for continuous flows [6], [7], [8], [9], [10], [11], etc. While some simulate low-level behaviors and some aim to capture high-level flow appearance, the resulting simulations, however, usually do not correlate to the real traffic on the street level.

On the other hand, the current trend in addressing urgent problems due to traffic congestion in urban environments encourages increasingly more traffic monitoring mechanisms, ranging from various forms of traffic sensors (cameras, road sensors, GPS) to the use of mobile phones for car tracking. Inspired by *Virtualized Reality* [12], we propose a novel concept of *Virtualized Traffic* that generates a continuous traffic flow from discrete spatio-temporal data to create a realistic visualization of highway and street-level traffic for synthetic environments. The resulting visualization automatically reflects and correlates to the real-world traffic and also enables possibly new VR applications that can benefit from visual analysis of actual traffic events (e.g. accidents) based on sensor data.

Main Results: Given two locations along a highway, say A and B, we assume that the velocity and the lane of each car is known at two corresponding time instances. The challenge is to reconstruct the continuous motion of multiple cars on the stretch of the highway in between the two given locations. We formulate it as a multi-robot planning problem, subject to spatial and temporal constraints. There are several key differences, however, between the traditional multi-robot planning problem and our formulation. First of all, we need to take into account the geometric, kinematic and the dynamic constraints of each car (though a subset of specialized algorithms have also considered these issues [13]). Second, in our formulation, not only the start time, but the arrival time of the cars is also specified. In contrast, the objective of previous literature has been for the robots to arrive at the goal location as soon as possible. Third, the domain that is dealt with here is an *open system*, i.e. the number of cars is not fixed. Instead, new cars can continuously enter the stretch of the highway to be visualized. This aspect requires incremental update to the current solution as new cars arrive at the given location.

In this paper, we present a prioritized approach that assigns priorities to each car based on the relative positions of the cars on the road: cars in front have a higher priority. Then, in order of decreasing priority, we compute trajectories for the cars that avoid cars of

- *J. Sewall, M. Lin and D. Manocha are with the Department of Computer Science, University of North Carolina at Chapel Hill E-mail: {sewall,lin,dm}@cs.unc.edu*
- *J. van den Berg is with the Department of Electrical Engineering and Computer Science, University of California at Berkeley E-mail: berg@berkeley.edu*

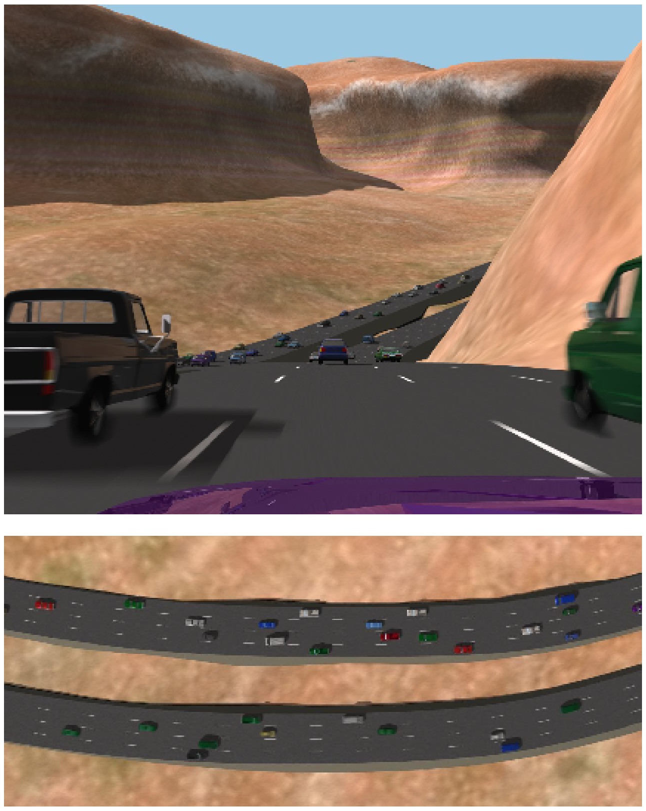

Fig. 1: Images of highway traffic synthesized by our method. Our method computes trajectories one by one for a continuous stream of cars (of possibly high-density). The trajectories fit the boundary conditions at the sensor points, and obey the geometric, kinematic and dynamic constraints on the cars. The number of lane changes and the total amount of (de-)acceleration are minimized and the distance to other cars is maximized to obtain smooth and plausible motions.

higher priority for which a trajectory has already been determined.

To make the search space for each car tractable, we constrain the motions of the car to a pre-computed roadmap, which is a reasonable assumption as each car typically has a pre-determined location to travel to. The roadmap provides links for changing lanes and encodes the car's *kinematic* constraints. Given such a roadmap, and a start and final state-time on the roadmap, we compute a trajectory on the roadmap that is compliant with the car's *dynamic* constraints and avoids collisions with cars of higher priority. At each time step, the car either accelerates maximally, maintains its current velocity, or decelerates maximally. This approach discretizes the set of possible velocities and the set of possible positions as well, enabling us to compute in three-dimensional state-time grids along the links of the roadmap. Our algorithm searches for a trajectory that minimizes the number of lane changes and the amount of (de-)acceleration, and maximizes the distance to other cars to obtain smooth and realistic motions. We show that this approach can successfully reconstruct traffic flows for a large number of cars efficiently, and examine the performance of our method on a set of real-world traffic flow data. Fig. 1 shows one of the challenging scenarios synthesized and visualized by our method.

Organization: The rest of this paper is organized as follows. First, we discuss related work in Section 2. In Section 3, we formally define the problem and a car's geometric, kinematic and dynamic constraints. In Section 4, we discuss the details of our approach and present experimental results in Section 5. Finally, we conclude and discuss future work in Section 6.

2 Related Work

In this section, we give a brief review of prior work first in traffic simulation, then in multi-agent planning as we extend some of the algorithms from robotics and adapt them here for our problem.

2.1 Traffic Simulation

The growing ubiquity of vehicle traffic in everyday life has generated considerable interest in models of traffic behavior, and a large body of research in the area has appeared in the last 60 years. The problem of traffic simulation has been very prominent in several fields — given a road network, a behavior model, and initial car states, how does the traffic in the system evolve? Such methods are typically designed to explore specific phenomena, such as jams and unstable, stop-and-go patterns of traffic, or to evaluate network configurations to aid in real-world traffic engineering.

Our approach does not address the classical problems of traffic simulation but instead *traffic reconstruction*, in which both the begin and end states of its cars are given. To better contrast our work against previous work, we give a brief overview of the commonly known methods for traffic simulation. For a more thorough review of the state of the art, see Helbing's extensive survey [1].

One popular category of traffic simulation techniques is broadly termed *microscopic* simulation. This classification includes discrete agent-based methods, wherein each car is treated as a discrete autonomous agent with arbitrarily complex rules governing their behavior. Most agent-based methods use some form of the "car-following" set of rules as described in [2] and [3]. Some of the public-domain traffic simulation systems, such as NETSIM [14], INTEGRATION [15], and MITSIM [16], are implemented using the agent-based modeling framework.

Nagel and Schreckenberg [4] applied cellular automata to the problem traffic simulation. The efficiency and simplicity of these models has led to much interest and extensions to the Nagel-Schreckenberg model (see the survey in Chowdhury et al. [5] for a detailed review).

Traffic may also be treated as a continuum and its evolution in time described by partial differential equations; this class of simulation methods is often called *macroscopic* simulation. Lighthill and Whitham [6] and Richards [7] were able to accurately capture a large number of traffic-related phenomena with a simple scalar nonlinear conservation law, and subsequent improvements by Payne [8] and Whitham [9] were able to describe more complicated states of traffic. Recently, the techniques described by Aw and Rascle [10] and Zhang [11] address some of the shortcomings of the Payne-Whitham model and provide concise description of traffic evolution. Unfortunately, these methods can be numerically challenging to handle due to the presence of discontinuities in the solution.

A third class of simulation methods, called *mesoscopic* methods, uses a continuum representation of traffic but uses Boltzmann-type mesoscale equations to traffic dynamics. This approach was pioneered by Prigogine and Andrews [17] and improved upon by Nelson et al. [18], Shvetsov and Helbing [19] and others.

There is also considerable work on using virtual environments for driving simulation [20], [21] and methods for modeling the vehicle behavior and navigable paths [22], [23], [24].

2.2 Multi-Robot Planning and Coordination

Existing approaches to multi-robot planning can roughly be divided into two categories: *coordinated* planning and *prioritized* planning. Coordinated approaches compute a path in the composite configuration space of the robots, which is formed by the Cartesian product of the configuration spaces of the individual robots [25], [26], [27]. They allow for complete planners, but their running time is exponential in the number of robots. The performance can be increased by constraining the configuration space of the individual robots to a preplanned path or roadmap [28], but the running time remains exponential in the number of robots.

Prioritized approaches incrementally construct a solution [29], [30]. Each of the robots is assigned a priority, and in order of decreasing priority the robots are selected. For each selected robot a trajectory is planned, avoiding collisions with the previously selected robots, which are considered as moving obstacles. Prioritized approaches are not complete, but the running time is only linear in the number of robots.

For the objective of traffic reconstruction, a coordinated approach cannot be applied. Not only would it be computationally unfeasible, but coordinated approaches are difficult to apply in a setting where new robots (or cars) continuously enter the scene without affecting motions of cars in the far past. A prioritized approach on the other hand, is well-suited for our application. Priorities can be naturally assigned based on the relative positions of the cars on the road, as it is reasonable to assume that cars only react to cars in front of them.

3 Problem Definition

Given as input a stretch of a highway between two points A and B of length L that has N lanes of a certain width. This highway is traversed by a continuous stream of cars. For each car i the sensors provide a tuple $(t_i^A, \ell_i^A, v_i^A, t_i^B, \ell_i^B, v_i^B)$ as data input, where $t_i^A \in \mathbb{R}$ is the time at which car i passes point A, $\ell_i^A \in 1 \dots N$ is the lane in which car i is at point A, and $v_i^A \in \mathbb{R}^+$ is the velocity of car i at point A (and similarly for point B).

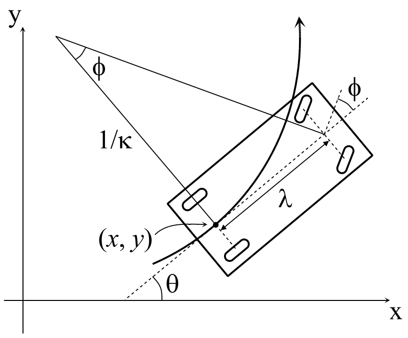

Fig. 2: The kinematic model of a car; (x, y) and θ are the position and the orientation of the car, λ is the distance between the front and rear axle, ϕ is the car's steering angle and κ is the curvature of the traversed path.

The task is to compute trajectories for the cars on the highway starting and arriving in the given lanes, at the given times, and at the given velocities. The trajectories should be computed such that the cars respect geometric constraints (e.g. respecting safety distance with each other), and such that the kinematic and dynamic constraints on the cars are enforced (see Section 3.1). Further, we want the reconstructed trajectories to look realistic; the cars should stay in their lane wherever possible, maintain sufficient distance to each other, and not unnecessarily accelerate or decelerate.

3.1 Kinematics and Dynamics of a Car

A car can be conceptualized as a rectangle moving in the 2-D plane. Its configuration is defined by a position (x, y), and an orientation θ (see Fig. 2). Let λ be the distance between the rear axle and the front axle of the car. The configuration transition equations of the car, in terms of path length s, are given by:

$$\begin{aligned} x'(s) &= \cos\theta && (1)\\ y'(s) &= \sin\theta && (2)\\ \theta'(s) &= \frac{\tan\phi}{\lambda} = \kappa, \qquad |\phi| \le \phi_{\max} && (3) \end{aligned}$$

where ϕ is the car's steering wheel angle, and κ the curvature of the traversed path. The steering wheel angle is bounded to reflect the car's minimum turning radius.

The above equations are the *kinematic constraints* on a car. They describe the traversal paths of a car. The *dynamic* constraints describe how such paths may be traversed over time t:

$$\begin{aligned} s'(t) &= v, \qquad 0 < v \le v_{\max} && (4)\\ v'(t) &= a, \qquad |a| \le a_{\max} && (5)\\ \phi'(t) &= \omega, \qquad |\omega| \le \omega_{\max} && (6) \end{aligned}$$

where v is the velocity of the car, a its acceleration and ω the speed with which the steering wheel is turned. The velocity of the car is bounded from below such that it can only move forward (which is realistic on a highway). The acceleration and the speed with which the steering wheel can be turned are bounded as well. Because of the discretization that is applied below, we choose symmetric bounds on the acceleration.

3.2 Discretization

To implement our traffic reconstruction method, we extend the approach presented by Van den Berg and Overmars in [31] that plans a trajectory for a robot under kinodynamic constraints in environments with *multiple moving obstacles*. We adapt the same discretization of the search space. We review that discretization here, and describe it in terms of our problem definition. The first discretization step is to construct a roadmap for the car's configuration space that encodes the kinematic constraints on the car. Constraining the cars to move along the edges of the roadmap ensure that the car's kinematic constraints are enforced. To comply with the car's dynamic constraints, we have to consider the *state space* of the car. To avoid the other cars in the environment, we extend the state space to the *state-time space*. In Section 4.1, we discuss how we construct a roadmap for the case of highway traffic reconstruction. Here, we describe how the state-space and the state-time space are discretized.

Let us first assume that the roadmap consists of a single path. The *state space* of the car then consists of pairs $\langle s, v\rangle$, where s is the position of the car along the path, and v the car's velocity. The state space is discretized into a grid by choosing a small time step Δt. At each time step, the car is allowed to change its velocity by choosing from a finite set of acceleration options A_n. If we choose to allow just three accelerations, we have the following state transition equations:

$$\begin{aligned} a &\in \{-a_{\max}, 0, a_{\max}\} = A_3 && (7)\\ v(t+\Delta t) &= v(t) + a\Delta t && (8)\\ s(t+\Delta t) &= s(t) + v(t)\Delta t + \tfrac{1}{2}a\Delta t^2 && (9)\end{aligned}$$

This results in a regular two-dimensional grid of reachable states (see Fig. 3), where the spacing in the grid is $\Delta v = a_{\max}\Delta t$ along the v-axis, and $\Delta s = \frac{1}{2}a_{\max}\Delta t^2$ along the s-axis. From a given state $\langle s, v\rangle$, three other states are reachable: $\langle s + (2\frac{v}{\Delta v} + 1)\Delta s, v + \Delta v\rangle$, $\langle s + 2\frac{v}{\Delta v}\Delta s, v\rangle$ and $\langle s + (2\frac{v}{\Delta v} - 1)\Delta s, v - \Delta v\rangle$, each one associated with a different acceleration. This defines a directed graph in the discretized state space which is called the *state graph*.

We are free to choose a from a different set of accelerations than that in Eq. (7) — it is possible and sometimes advantageous to give the search a finer-grained choice. Our formulation assumes an acceleration of the form

$$\begin{aligned} A_{2n+1} = \{&-a_{\max}, -a_{\max}2^{-n+2}, -a_{\max}2^{-n+1}, \ldots, 0,\\ &a_{\max}2^{-n+1}, a_{\max}2^{-n+2}, \ldots, a_{\max}\} \quad (10)\end{aligned}$$

where $n \in \mathbb{N}_1$.

The branching factor of our search increases with n and leads to longer compute and greater memory usage; however, the wider array of acceleration options can be useful for reconstructing some inputs. In practice, only A_3 and $A_5 = \{-a_{\max}, -a_{\max}/2, 0, a_{\max}/2, a_{\max}\}$ are practical for most real-time applications due to the exponential expense that comes with increases in n. In general, for acceleration set A_{2n+1}, the spacing on the v-axis should be $\Delta v = a_{\max}2^{-n+1}\Delta t$, the spacing on the s-axis should be $\Delta s = \frac{1}{2}a_{\max}2^{-n+1}\Delta t^2$, and there are $2n+1$ states reachable from a given state $\langle s, v\rangle$.

To define the state graph for the entire roadmap rather than a single path, state grids along each of the edges of the roadmap are connected at the vertices of the roadmap, such that the car can choose among all of the outgoing edges when it encounters a vertex. As can be seen in Fig. 3, only half of the states in the state grid are reachable. In order to connect the state grids smoothly at the vertices, each of the edges of the roadmap is subdivided into an *even* number of steps. As a result, there is a finite number of reachable positions in the roadmap. For all of these positions, the velocity is bounded by Equation (4). If the roadmap edge has curvature, the upper bound of the velocity may be further restricted by the dynamic constraint of Equation (6). States outside the velocity bounds are defined not to be part of the state graph. As a result, the total state graph contains a finite number of states, but—in contrast to [31]—we do *not* construct the state graph explicitly.

To avoid collisions with other cars while planning on the state graph, the time dimension is added to the discretized state space, forming a three-dimensional *state-time space* along each of the edges of the roadmap (see Fig. 3). It consists of pairs $\langle q, t\rangle$, where $q = \langle s, v\rangle$ is a state contained in the state graph, and t a time value. The time axis is discretized by the time step Δt. Other cars moving on the highway transform to static obstacles in the state-time space. They are *cylindrical* along the v-dimension, as the car's velocity does not influence its collision status.

Like the state graph is defined on the discretized state space, the *state-time graph* is defined on the discretized state-time space. It is a directed acyclic graph, that contains a transition from state-time $\langle q, t\rangle$ to $\langle q', t + \Delta t\rangle$ if q' is a successor of q in the state graph. The task is to compute a trajectory through the state-time graph from a given start state-time $\langle q_{\text{start}}, t_{\text{start}}\rangle$ to a given goal state-time $\langle q_{\text{goal}}, t_{\text{goal}}\rangle$. The state-time graph is explored *implicitly* during the search for a trajectory.

4 Reconstructing Traffic

In this section we discuss how we reconstruct the traffic from the acquired sensor data, given the discretization of the search space as defined in Section 3.2.

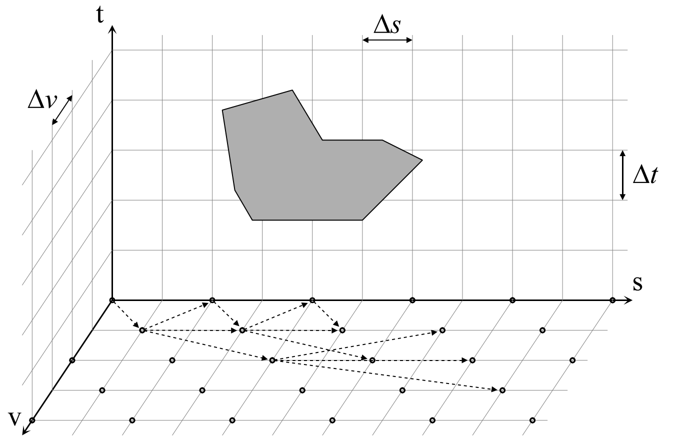

Fig. 3: The three-dimensional state-time grid along a single edge of the roadmap. Obstacles (gray) are cylindrical along the v-dimension. A part of the state graph (or equivalently, the projection of the state-time graph) is shown using dashed arrows on the sv-plane. Only the grid points marked by the dots are reachable. Each transition takes one time step.

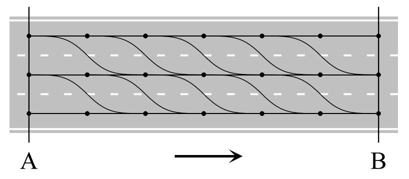

Fig. 4: A roadmap constructed for a highway with three lanes. The highway was subdivided into six segments. The thick dots are the vertices of the roadmap. Only lane changes to the right of the length of two segments are shown here.

4.1 Constructing the Roadmap

As explained in Section 3.2, the cars are constrained to move over a preprocessed roadmap to make the configuration space of a car tractable. We construct this roadmap as follows. First, we subdivide the highway into M segments of equal length. For each lane of the highway, we place a roadmap vertex at the end of each segment (see Fig. 4). This gives a $M \times N$ grid of roadmap vertices, where N is the number of lanes. Each vertex (i, j) is connected by an edge to the next vertex $(i+1, j)$ in the same lane. These edges allow cars to stay in their lane and move forward. To allow for lane changes, we also connect vertices of neighboring lanes. Each vertex (i, j) is connected to vertices $(i+a, j+1), \ldots, (i+b, j+1)$ and $(i+a, j-1), \ldots, (i+b, j-1)$. Here a and b denote the minimum and maximum length (in number of segments) of a lane change, respectively. The short lane changes are useful at lower velocities, the longer ones at higher velocities.

When adding the edges for lane changes, we have to make sure that they are "realistic". That is, they should obey the kinematic constraints of a car and should be traversable without abrupt steering wheel motions. Let

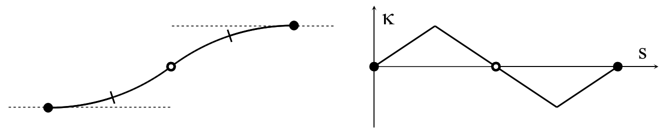

Fig. 5: A lane change curve (left) between two points consists of four clothoid curves, i.e. curves with constant curvature derivative (right).

us look more closely at the constraint on the speed with which the steering wheel is turned given in Equation (6). It translates into the following bound on the *curvature* derivative:

$$|\phi'(t)| \leq \omega_{\max} \Leftarrow |\kappa'(t)| \leq \frac{\omega_{\max}}{\lambda} \Leftrightarrow |\kappa'(s)| \leq \frac{\omega_{\max}}{v\lambda} \Leftrightarrow$$
$$v \leq \frac{\omega_{\max}}{|\kappa'(s)|\lambda} \quad (11)$$

In other words: the smaller the curvature derivative (with respect to path length s), the higher the velocity with which this path can be traversed. Hence, we look for lane-change curves with the smallest possible (absolute) curvature derivative. Let us look at a lane change to the left (see Fig. 5). Note that a lane-change curve between two points is symmetric in its midpoint. At its midpoint, the curvature (and the steering wheel angle) must be zero, as it is the reversal point from steering to the left to steering to the right. The curvature is also zero at its starting point and end point. Hence, the curve in-between the starting point and the midpoint consists of two curves, one with maximal positive curvature derivative, the other with maximal negative curvature derivative. A curve with constant curvature derivative is well known to be a *clothoid*, so one lane-change edge consists of four clothoid curves.

The roadmap resulting from the above method is valid for cars with any value of λ, so we need to construct a roadmap only once, and can use it for all cars.

4.2 Trajectory for a Single Car

Given a roadmap as constructed above and the state-time graph as defined in the previous section, we describe how we can compute a trajectory for a single car, assuming that the other cars are moving obstacles of which we know their trajectories. How we reconstruct the traffic flows for multiple cars is discussed in below.

A straightforward approach for searching a trajectory in the state-time graph is the A*-algorithm. It builds a *minimum cost tree* rooted at the start state-time and biases its growth towards the goal. To this end, A* maintains the leaves of the tree in a priority queue Q, and sorts them according to their f-value. The function $f(\langle q, t\rangle)$ gives an estimate of the cost of the minimum cost trajectory from the start to the goal via $\langle q, t\rangle$. It is computed as $g(\langle q, t\rangle) + h(\langle q, t\rangle)$ where $g(\langle q, t\rangle)$ is the cost it takes to go from the start to $\langle q, t\rangle$, and $h(\langle q, t\rangle)$ a lower-bound estimate of the cost it takes to reach the goal from $\langle q, t\rangle$. A* is initialized with the start state-time in its priority queue, and in each iteration it takes the state-time with the lowest f-value from the queue and *expands* it. That is, each of the state-time's successors in the state-time graph is inserted into the queue if they have not already been reached by a lower-cost trajectory during the search. This process repeats until the goal state-time is reached, or the priority queue is empty. In the latter case, no valid trajectory exists. During the search we keep track of a "backpointer" $\mathrm{bp}(\langle q,t\rangle)$ that maps each traversed state to its ancestor so as to be able to reconstruct the trajectory if one is found. The algorithm is given in Algorithm 1.

Algorithm 1 A*$(q_{\mathrm{start}}, t_{\mathrm{start}}, q_{\mathrm{goal}}, t_{\mathrm{goal}})$

```
1:  $g(\langle q_{\mathrm{start}}, t_{\mathrm{start}}\rangle) \leftarrow 0$
2:  Insert $\langle q_{\mathrm{start}}, t_{\mathrm{start}}\rangle$ into $Q$
3:  while $Q$ is not empty do
4:      Pop the element $\langle q,t\rangle$ with lowest $f$-value from $Q$
5:      if $q = q_{\mathrm{goal}}$ and $t = t_{\mathrm{goal}}$ then return success!
6:      for all successors $q'$ of $q$ in the state graph do
7:          $c \leftarrow$ cost of edge between $\langle q,t\rangle$ and $\langle q', t+\Delta t\rangle$
8:          if $g(\langle q', t+\Delta t\rangle) > g(\langle q,t\rangle) + c$ then
9:              $\mathrm{bp}(\langle q', t+\Delta t\rangle) \leftarrow \langle q,t\rangle$
10:             $g(\langle q', t+\Delta t\rangle) \leftarrow g(\langle q,t\rangle) + c$
11:             Insert or update $\langle q', t+\Delta t\rangle$ in $Q$
12: Trajectory does not exist; return failure
```

In [31] the A*-algorithm was used to find a *minimal-time trajectory*. That is, only a goal state is specified, and the task is to arrive there as soon as possible. This makes it easy to focus the search towards the goal; the cost of a trajectory is simply defined as its length (in terms of time). However, in our case the arrival time is specified as well, so we know in advance how long our trajectory will be. Therefore, we cannot use time as a measure in our cost function. Instead, we let the cost of a trajectory T depend on the following criteria, in order to obtain smooth and realistic trajectories:

- The number of lane changes $X(T)$ in the trajectory.
- The total amount $A(T)$ of acceleration and deceleration in the trajectory.
- The accumulated cost $D(T)$ of driving in closer proximity than a preferred minimum $d_{\mathrm{limit}} > 0$ to other cars.

More precisely, the total cost of the trajectory T is defined as follows:

$$\mathrm{cost}(T) = c_X X(T) + c_A A(T) + c_D D(T) \tag{12}$$

where c_X, c_A and c_D are weights specifying the relative importance of each of the criteria. $A(T)$ and $D(T)$ are defined as follows:

$$A(T) = \int_T |v'(t)|\, dt \tag{13}$$

$$D(T) = \int_T \max\left(\frac{d_{\mathrm{limit}}}{d(t)} - 1, 0\right) dt \tag{14}$$

where $v(t)$ is the velocity along the trajectory as a function of time, and $d(t)$ is the distance (measured in terms of time) to the nearest other car on the highway as a function of time.

The distance $d(t)$ to other cars on the highway given a position s in the roadmap and a time t is computed as follows. Let t' be the time closest to t at which a car configured at s would be in collision with another car, given the trajectories of the other cars. Then, $d(t) = |t - t'|$. We obtain this distance efficiently by – prior to determining a trajectory for the car – computing for all positions in the roadmap during what time intervals it is in collision with any of the other cars. Now, $d(t)$ is simply the distance between t and the nearest collision interval at s. If t falls within an interval, the car is in collision and the distance is zero. As a result, the above cost function would assume infinite value.

In the A*-algorithm, we evaluate the cost function per edge of the state-time graph that is encountered during the search. The edge is considered to contain a lane change if a lane-change edge of the roadmap is entered. The total cost $g(\langle q,t\rangle)$ of a trajectory from the start state-time to $\langle q,t\rangle$ is maintained by accumulating the costs of the edges the trajectory consists of. The lower bound estimate $h(\langle q,t\rangle)$ of the cost from $\langle q,t\rangle$ to the goal state-time $\langle q_{\mathrm{goal}}, t_{\mathrm{goal}}\rangle$ is computed as follows:

$$v_{\mathrm{avg}} = \frac{x(q) - x(q_{\mathrm{goal}})}{t_{\mathrm{goal}} - t} \tag{15}$$

$$\begin{aligned} h(\langle q,t\rangle) = {} & c_X|\mathrm{lane}(q) - \mathrm{lane}(q_{\mathrm{goal}})| + \\ & c_A(|v(q) - v_{\mathrm{avg}}| + |v(q_{\mathrm{goal}}) - v_{\mathrm{avg}}|) \end{aligned} \tag{16}$$

where v_{avg} is the average velocity of the trajectory from $\langle q,t\rangle$ to $\langle q_{\mathrm{goal}}, t_{\mathrm{goal}}\rangle$, and $x(q)$, $\mathrm{lane}(q)$ and $v(q)$ are respectively the the position along the highway, the lane and the velocity at state q. If $v_{\mathrm{avg}} > v_{\mathrm{max}}$, we define $h(\langle q,t\rangle) = \infty$.

An advantage of the goal time being specified is that we can apply a bidirectional A*, in which a tree is grown from both the start state-time and the goal state-time in the reverse direction until a state-time has been reached by both searches. This greatly reduces the number of states explored and hence the running time.

Streaming: Let us assume that we acquire data from each of the sensors A and B whenever a car passes by. Obviously, for each car, we first acquire data from A and then from B. We order the cars in a *planning queue* sorted by the time at which the cars pass sensor A. The queue continuously grows when new sensor data arrives from sensor A. Now, continually, we compute a trajectory for the car at the front of the queue when its data from sensor B has arrived. To this end, we use the algorithm of the previous section, such that the car avoids other cars for which a trajectory has previously been computed (which is initially none). The start state-time and the goal state-time are directly derived from the data acquired at sensor A and B respectively. They are rounded to the nearest point in the discretized state-time space. This procedure repeats indefinitely.

Streaming Property: The reconstructed trajectories can be regarded as a "movie" of the past, or as a function $R(t)$ of time. As new trajectories are continually computed, the function $R(t)$ changes continuously. However, the above scheme guarantees that $R(t)$ is *final* for time t if $(\forall i : t_i^A < t : t_i^B < t_{\mathrm{cur}})$, where t_{cur} is the current "real world" time. "Final" means that $R(t)$ will not change anymore for time t when trajectories are determined for new cars. In other words, we are able to "play back" the reconstruction up till time t as soon as all cars that passed sensor A before time t have passed sensor B. We call this the *streaming* property; it allows us to stream the reconstructed traffic at a small delay.

Real Time Requirements: In order for our system to run in *real time*, that is, so that the computation does not lag behind new data arriving (and the planning queue grows bigger and bigger), we need to make sure that reconstruction takes on average no more time than the time in between arriving cars. For instance, if a new car arrives every second, we need to be able to compute trajectories within a second (on average) in order to have real-time performance.

4.3 Qualitative Analysis

Prioritization: The above scheme implies a static prioritization on the cars within a given pair of sensor locations. Cars are assigned priorities based on the time they passed sensor A, and in order of decreasing priority trajectories are calculated that avoid cars of higher priority (for which trajectories have previously been determined). This is justified as follows: in real traffic drivers mainly react to other cars in *front* of them, hardly to cars behind. This is initially the case: a newly arrived car i has to give priority to all cars in front of it. On the other hand, car i may overtake another car j, after which it still has to give priority to j. However, it is not likely that once car i has overtaken car j that both cars will 'interact' again, and that car j influences the the remainder of the trajectory of car i. This can be seen as follows. If we assume that cars travel on a trajectory with a constant velocity (this is what we try to achieve by the optimization criteria of Equation (13)), each pair of cars only interact (i.e. one car overtakes the other) at most *once*.

In fact, in a real-world application it is to be expected that multiple consecutive stretches of a highway are being reconstructed, each bounded by a pair of a series of sensors $A, B, C, \ldots$ placed along the highway. If car i overtakes car j in stretch AB, then for reconstructing the stretch BC, car i has gained priority over car j. So, when regarded from the perspective of multiple consecutive stretches being reconstructed, there is an implicit *dynamic* prioritization at the resolution of the length of the stretches.

Traffic Phenomena: This viewpoint of multiple connected stretches is very important when analyzing the traffic behavior seen in the reconstructions and streaming real-time data. For each stretch individually, our algorithm attempts to reconstruct as smooth a motion as possible. So, it is unlikely to see traffic jam phenomena emerge at resolutions lower than the stretch length in the reconstructions. However, in the more global scale over multiple stretches, these phenomena are observable, as the algorithm tries to fit the data. This can be viewed analogous to the Nyquist-Shannon sampling theorem, stating that frequencies higher than the sampling resolution cannot be captured.

Noise Sensitivity: Our method is hardly sensitive to noise in the data. This can be understood by the fact that sensed passing times and velocities at the sensors are rounded to the nearest point on the discretized time-axis and velocity-axis respectively to initialize the reconstruction algorithm.

5 Experimental Results

We have implemented our method and experimented on various challenging scenarios.

5.1 Quantitative Results

In our first experiments we use a highway with $N = 4$ lanes of $L = 1000$ meters length. Lane change curves are 50 meters long. For the cars, we set $a_{\max} = 3m/s^2$, $v_{\max} = 35m/s$ (close to 80 MPH), $d_{\mathrm{limit}} = 1s$ and $\omega_{\max} = 1\mathrm{rad}/s$. We set the time step Δt at $0.5s$, which, for A_3, gives $\Delta v = 1.5m/s$ and $\Delta s = 0.375m$ for the discretization of the state space. As a result, when using A_3, the roadmap consists of 41570 discrete positions. For several of these scenarios, we have repeated the experiments with the five- and seven-cardinality acceleration sets A_5 and A_7. These result in smaller Δs and Δv and therefore result in more detailed roadmaps and larger search spaces, in addition to the larger branching factor during the search. Where appropriate, we have run the experiments with these accelerations for a subset of the independent parameters due to their long running times.

To stress test our work on various scenarios, the data were randomly generated. For each car i we pick a random start time t_i^A from the interval $[t_{i-1}^A, t_{i-1}^A + 2/(\rho N)]$, where ρ is the traffic density (i.e. the number of cars per second per lane). The end time t_i^B is selected as $t_i^A + L/V$, where V is the average velocity randomly picked between 20 and $30m/s$. The start and end lane are randomly chosen as well, and the start and end velocities are fixed at $22.5m/s$.

Dense Traffic: In our first experiment, we set $\rho = 1/2$, which gives fairly dense traffic (a new car enters the four-lane highway every $1/(\rho N) = 1/2$ seconds on average). Given the fact that the average velocities are relatively high and far between (between 20 and $30m/s$) and the start and end lanes are randomly chosen, this is a challenging example. Such data is not likely to occur in practice. We compute trajectories for 50 cars. In the

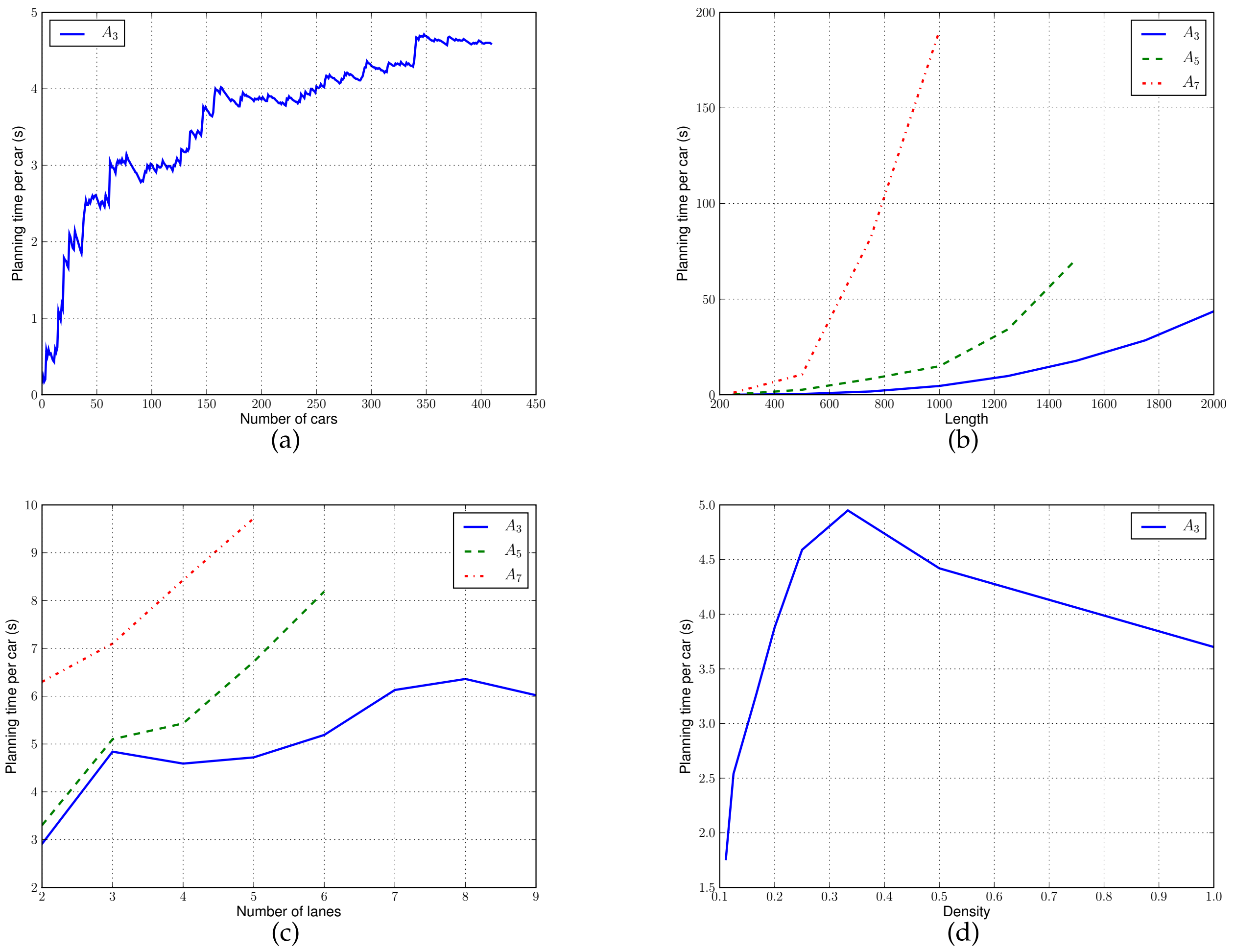

Fig. 6: (a) The average compute time of the first x cars in our experiment ($L = 1000m, N = 4, \rho = 1/2$). (b) The average compute time as a function of the highway length ($N = 4, \rho = 1/2$). (c) The average compute time as a function of the number of lanes ($L = 1000m, \rho = 1/2$). (d) The average compute time as a function of the density ($L = 1000m, N = 4$).

supplementary video, the reconstructed traffic can be viewed. Because of the relatively large differences in average velocities of the cars, it is interesting to see that some fast cars aggressively race through the traffic to reach the "goal" (i.e. the end of the highway) in time.

Performance: In Fig. 6(a) we plot the average running time of the first x cars for this experiment. What can be seen from the chart is that the compute time does not increase much when more cars have previously been considered. Only the running times for the very first cars are faster, because they do not have to avoid any other cars. For the rest of the cars, the traffic density is more or less equal. In this *worst* case scenario, the average running time over all 500 cars was 4.6 seconds. For real-time data streaming, the reconstruction is faster and can be done at interactive rates. However, the search space (the state-time space) is big, and focusing the A*-search to the goal can be hard as we are not searching for a time-minimal trajectory. In general, a low-cost trajectory is found quickly, whereas a high-cost trajectory can take more time before it is found. This is because the A*-search first exhausts all the possible low-cost trajectories before it expands leaves of the search tree with a high cost.

Effect of Road Length: In our subsequent experiments, we varied the major parameters, while keeping the others equal. In Figs. 6(b), (c) and (d) we see how the compute time varies with the highway length, the number of lanes, and the traffic density, respectively. We see that the reconstruction time clearly increases as the length of the highway increases. In fact, the curves shown are nearly perfect cubic functions (i.e. polynomials of degree 3). This can be explained as follows. As we keep the average velocity constant, the length (in terms of time) of the trajectories increases with the length of the highway. As the A* algorithm searches in a *three*-dimensional state-time space (see Fig. 3), the volume of the search tree is expected to grow cubically with the depth of the tree (i.e. the number of time steps). This cubic growth trend is observed for A_5 and A_7, although the scale grows with

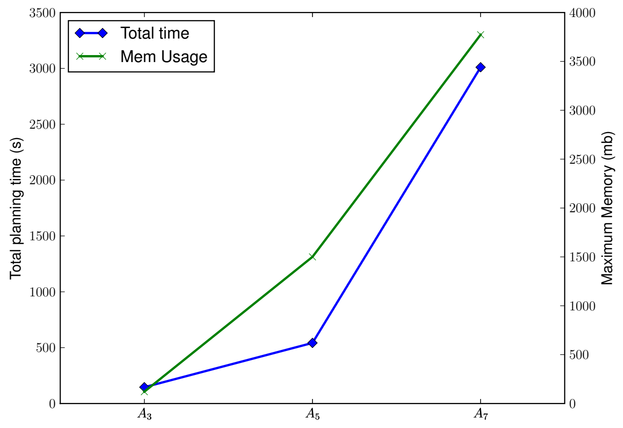

Fig. 7: Total running time and approximate peak memory usage for the acceleration choices A_3, A_5, A_7. The experiment ran for each is a straight four-lane, 1000-meter highway with 50 cars.

the number of acceleration options.

The Number of Lanes: We see that the reconstruction time increases little as the number of lanes increases. In principle, twice the number of lanes gives twice as large a search space. However, the lengths of trajectories in terms of time remain constant regardless of the number of lanes. Also, an increase in lanes gives more space to find a low-cost trajectory, which are found quicker than high-cost trajectories. The larger branching factors of A_5 and A_7 result in slightly worse scaling in the larger search space afforded by the increase in lanes.

Traffic Density: The density of the traffic seems to have a more or less linear relationship with the compute time: the lower the density, the lower the compute time. When there is hardly any traffic, each car can find a low-cost trajectory quickly. However, for very high density the compute time seems to decrease. This is due to the fact that the quantity of traffic introduces many collision constraints, which in turn limits the branching factor of the search.

Impact of Time Steps: We note that over all experiments, we have kept the time step Δt constant at a low 0.5s, but we note that the running time decreases *quartically* (i.e. $\sim 1/\Delta t^4$) when the time step increases. This is because that the search space is three-dimensional, and the spacing in the discretized grid is Δt for the time axis, $O(\Delta t)$ for the v-axis, and $O(\Delta t^2)$ for the s-axis (see Fig. 3). So, for instance, for a time step of $\Delta t = 1s$, which is fine for most practical situations, the compute times are ~ 16 times less than the ones reported for these experiments.

Impact of A_i: With the minimum number of three acceleration options (A_3), reasonable results can be achieved while keeping the branching factor of the search low. However, it is sometimes desirable to expand the number of acceleration choices available to vehicles to suit a particular problem. The qualitative cost of increasing the number of search options in the A^* algorithm is clear — increasing the branching factor of a search at all nodes generally incurs an exponential increase in memory usage and running time. We quantitatively have observed these effects for A_3, A_5, and A_7; see Fig. 7. Accelerations sets A_9 and larger did not fit in core memory for these experiments and were omitted.

Real-time Data Streaming: Given a density of ρ, the real-time requirement (see Section 4.2) states we need to calculate within $1/(\rho N)$ time on average per car. The time step Δt can be tuned to achieve this requirement. We note that for $\Delta t = 1s$, the experiments with $L = 1000m, \rho = 1/2$ and $N = 4$ can be run in real time. The time step should obey $\Delta t < 1/\rho$ to capture high density traffic. Otherwise the time value of multiple cars entering the same lane of the highway will be rounded to the same point on the time-axis.

5.2 Scenarios

We further applied our method to two specific scenarios. One is a cloverleaf highway interchange (see Fig. 8). In this case, we have a sensor at each of the four arms of the intersection. Cars can enter and leave the intersection at any sensor point and our algorithm computes their trajectories accordingly. In our example, we used highways of $1000m$ length with four lanes and a density of $\rho = 1/2$. As can be seen in Fig. 8 and the supplementary video, the reconstruction gives plausible and smooth traffic even in the case of a cloverleaf intersection.

The next scenario actually consists of multiple consecutive stretches, as we discussed in Section 4.3. In our example, we place four sensors A, B, C and D along a linear highway with four lanes such that the stretch AB is $400m$, BC is $200m$ and CD is $400m$ long. We generated the data such that the average velocity of the cars in the first and the last section was $20m/s$ and in the middle $5m/s$ to simulate a traffic jam scenario. The

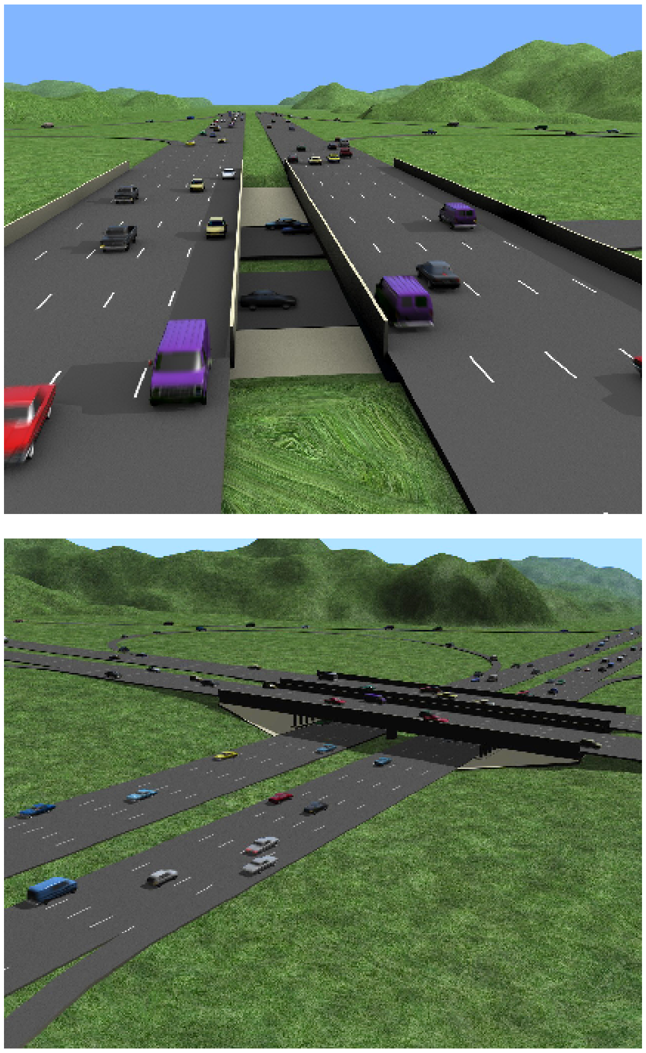

Fig. 8: Images from our cloverleaf scenario ($L = 4 \times 1000m, N = 4, \rho = 1/2$). There are sensors at each of the arms of the cloverleaf intersection. Cars can enter and leave the intersection at any sensor and our algorithm computes their trajectories accordingly.

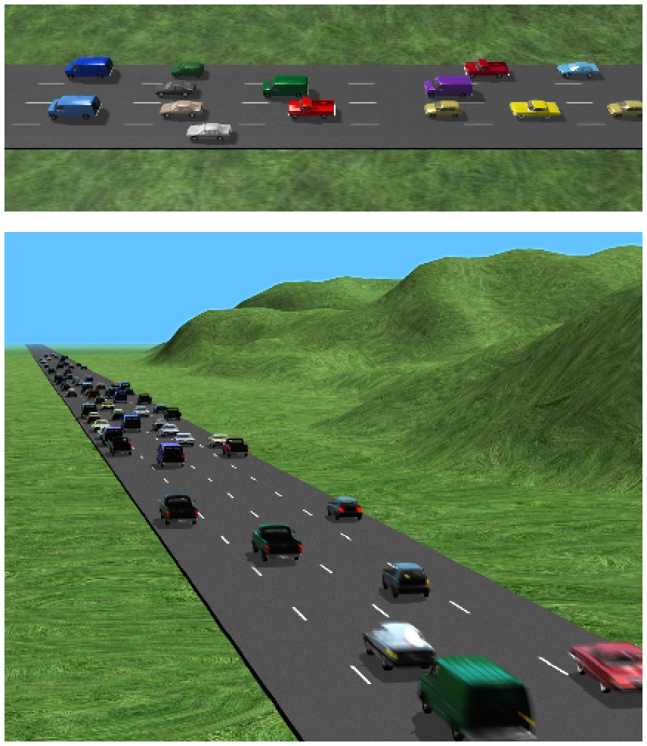

Fig. 9: Images from our traffic jam scenario ($L = \{400m, 200m, 400m\}, N = 4, \rho = 1/2$). The traffic of three consecutive stretches of a highway are reconstructed independently, and afterwards concatenated in a single visualization. In order to simulate a traffic jam, we generated the data such that the average velocity in the middle section was much less than in the other two.

traffic was reconstructed independently for each section of the road, and afterwards concatenated together in a single visualization. As can be seen in Fig. 9 and the supplementary video, the traffic jam can be clearly reconstructed by our method.

5.3 Validation

The aforementioned scenarios consist of synthetic but representative road segments with procedurally-generated input. It is important to consider how our technique performs on input from real traffic. To test the applicability of our method to real-world problems, we have extracted the relevant start/end pairs from a set of vehicle trajectory data and given them as input to our system.

5.3.1 Data

The vehicle trajectory data were obtained through the U.S. Federal Highway Administration's Next-Generation Simulation (NGSIM) project [32] and contains trajectory data for every vehicle traveling along a segment of I-80 passing through Emeryville, CA for a fifteen-minute interval. The segment of road in consideration is six lanes of northbound highway approximately 370 meters long. An on-ramp adjoins and feeds traffic to the rightmost lane near the beginning of the segment in consideration, tapering off over the first 250 meters of road segment.

The vehicle trajectory data contains samples for the position, velocity, and acceleration of each vehicle at regular intervals each separated by $1/10\text{th}$ of a second. The dataset also contains information about the length, width, and 'type' of each vehicle — motorcycles, freight trucks, and consumer-type automobiles all appear in the dataset under consideration.

5.3.2 Approach

Our approach uses a discrete roadmap to determine the path each vehicle travels in space; this includes where vehicles may change lanes and the behavior of merging traffic. While the I-80 vehicle trajectory data is very detailed, there is no explicit information about the geometry of the road nor the connectivity of the lanes. Clearly, the configuration of the underlying road has a significant impact on any traffic flow. For our technique to operate, we require a detailed roadmap.

To compute a roadmap for this validation test, we took advantage of the quantity and detail of the vehicle trajectory data. While real-world vehicles have a tendency to drift slightly as they travel along a lane (and this was reflected in the input data), there was a suitable quantity of data to determine that the stretch of I-80 under consideration was very nearly linear. With further analysis of the trajectories of vehicles entering the highway via the on-ramp, we were able to determine its shape and relationship with the neighboring lanes of the main highway.

The trajectory data required some processing to be suitable for our technique; for each vehicle, the time-series of position/velocity needed to be examined to find the starting and ending time/lane/velocity. Furthermore, the entrance and exit points were not consistent across all vehicles; some appear/leave lanes a few meters closer/further along the road than others. To fit our roadmap model, the starting and ending values were 'clipped' to accommodate all of these paths. This clipping simply has the effect of narrowing the region we are considering by a few meters but owing to the high-resolution nature of the trajectory data, we were able to accurately interpolate the clipped starting information.

5.3.3 Performance

Recall that our technique makes a best-effort search for paths rather than search the entire space of all possible vehicle paths. Input vehicles treat already-planned vehicles (ahead of them) as obstacles, and there are cases where no path can be found for a vehicle given the prior planning. Ideally, given real-world data, we would like our method to produce a path for every input vehicle. However, there are certain vehicle behaviors present in real-world data that our technique does not model, and that are likely to cause difficulties in achieving a 100% success rate in planning. For example,

- Our model assumes that vehicles travel only along lanes or on certain lane-change path. In California, the practice of "lane-splitting" is legal — motorcycles are free to travel in between cars in adjacent lanes. This occurs in the I-80 dataset, and presents a challenge for our method, which must try to find a path around such obstacles and force each vehicle to precisely follow a single lane.
- We assume discrete, symmetric options for acceleration — e.g. for a 3-acceleration version of our technique, a vehicle may decelerate maximally, not accelerate, or accelerate maximally at any given moment. Trajectories in the I-80 dataset exhibit a continuous range of accelerations. While our technique is capable of representing the same *range* of accelerations, expanding the search space to more than 7 discrete acceleration options quickly becomes impractical in both running time and memory usage.

In addition, due to sensor noise and uncertainties, we have observed the trajectories of some cars recorded from the real-world traffic to be spatially and temporally incoherent, i.e. the sampled positions of some vehicles seem to "jump around" over time. Of the 2052 vehicles present in the I-80 vehicle set, the three-acceleration variant A_3 of our method successfully reconstructed 1686 of the vehicles (82.2%). Our method was able to reconstruct the 15 minutes of real-world traffic from the data in 6.64 minutes; representative frames from the original validation data and reconstruction data are shown in Fig. 10.

6 Discussion and Future Work

In this paper, we have presented a novel concept of *Virtualized Traffic*, in which traffic is reconstructed from discrete data obtained by sensors placed alongside a highway or street. We have presented an algorithm to determine the trajectories for multiple cars that also allows *streaming* real-world traffic data in *real time* to visualize traffic as data is recorded. We have adapted a prioritized search method to compute trajectories and examined how our technique operates on real-world traffic data.

Our current approach strikes a trade-off between the quality of the reconstructed traffic and the overall performance of the approach for interactive applications. Further investigations can be made to improve the quality of traffic reconstruction and visualization for other non-real-time applications. For example, while we support several discrete acceleration options for vehicles, we are constrained to symmetric velocities; in reality, a vehicle is generally more able to break (decrease velocity) than accelerate (increase velocity). To model this aspect would significantly increase the cost of reconstruction in both runtime performance and storage requirements.

In our current discretization of the state-time space, we choose a fixed time step, which gives a discrete set of reachable positions and velocities. However, traffic usually involves high-speed motion, so to increase the resolution of the discretization at large velocities, we may instead consider a fixed amount of traversed distance, and derive the velocities and times accordingly.

Our validation experiments with real-world data have been promising, but refinements to the structure of our roadmaps are necessary for our technique to be able to best describe all of the features present in real-life vehicle motion — for example, to properly be able to describe the motion of motorcycles traveling between lanes.

We have shown in this paper that our framework is applicable to complex highway scenarios, including cloverleaf intersections and traffic jams. An interesting extension would be to allow for intersections with traffic lights or stop signs, and entire roadmaps of streets in urban/suburban environments.

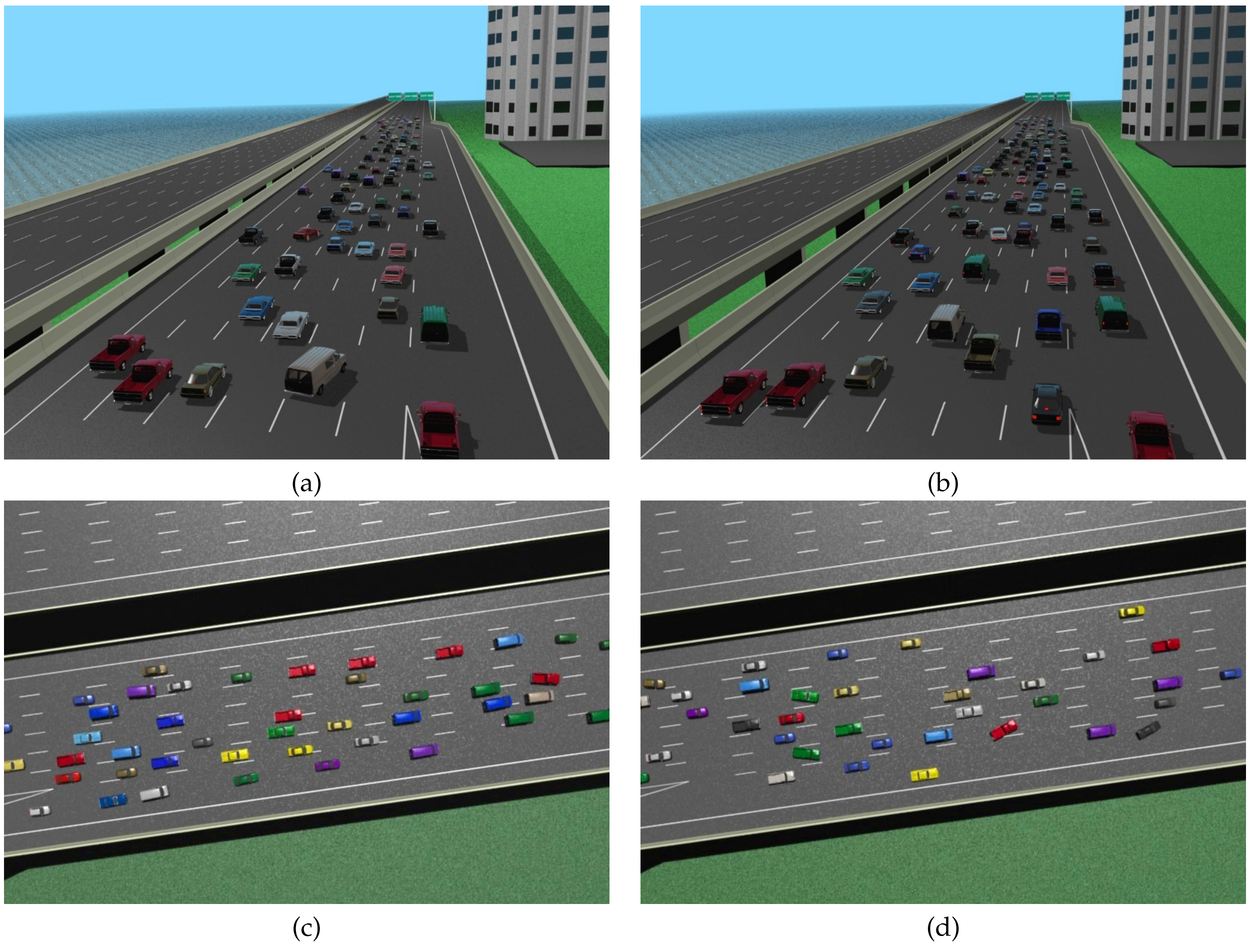

Fig. 10: a) Original I-80 vehicle trajectory data. b) Reconstructed I-80 vehicles c) Original I-80 vehicle trajectory data d) Reconstructed I-80 vehicles

Acknowledgment

We would like to thank Prof. Alexandre Bayen and Ryan Herring from the Institute of Transportation Studies at the University of California at Berkeley for their assistance in obtaining real-world trajectory data.

This research is supported in part by the Army Research Office, National Science Foundation, RDECOM, and Intel Corporation.

References

[1] D. Helbing, "Traffic and related self-driven many-particle systems," *Reviews of Modern Physics*, vol. 73, no. 4, pp. 1067–1141, 2001.

[2] D. L. Gerlough, "Simulation of freeway traffic on a general-purpose discrete variable computer," Ph.D. dissertation, UCLA, 1955.

[3] G. Newell, "Nonlinear effects in the dynamics of car following," *Operations Research*, vol. 9, no. 2, pp. 209–229, 1961.

[4] Kai Nagel and Michael Schreckenberg, "A cellular automaton model for freeway traffic," *Journal de Physique I*, vol. 2, no. 12, pp. 2221–2229, dec 1992. [Online]. Available: http://dx.doi.org/doi/10.1051/jp1:1992277

[5] D. Chowdhury, L. Santen, and A. Schadschneider, "Statistical Physics of Vehicular Traffic and Some Related Systems," *Physics Reports*, vol. 329, p. 199, 2000.

[6] M. J. Lighthill and G. B. Whitham, "On kinematic waves. ii. a theory of traffic flow on long crowded roads," *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences (1934-1990)*, vol. 229, no. 1178, pp. 317–345. [Online]. Available: http://journals.royalsociety.org/content/q23th31837k884h0

[7] P. I. Richards, "Shock waves on the highway," *Operations research*, vol. 4, no. 1, p. 42, 1956, doi: pmid:.

[8] H. J. Payne, *Models of freeway traffic and control*, 1971, iD: 29690330.

[9] G. B. Whitham, *Linear and nonlinear waves*. New York: Wiley, 1974, iD: 815118.

[10] A. Aw and M. Rascle, "Resurrection of "second order" models of traffic flow," *SIAM Journal of Applied Math*, vol. 60, no. 3, pp. 916–938, 2000.

[11] H. M. Zhang, "A non-equilibrium traffic model devoid of gas-like behavior," *Transportation Research Part B: Methodological*, vol. 36, no. 3, pp. 275–290, March 2002.

[12] T. Kanade, P. Rander, and P. Narayanan, "Virtualized reality: Constructing virtual worlds from real scenes," *IEEE MultiMedia*, vol. 4, no. 1, pp. 34–47, 1997.

[13] C. M. Clark, T. Bretl, and S. Rock, "Applying kinodynamic randomized motion planning with a dynamic priority system to multi-robot space systems," *IEEE Aerospace Conference Proceedings*, vol. 7, pp. 3621–3631, 2002.

[14] A. Byrne, A. de Laski, K. Courage, and C. Wallace, "Handbook of computer models for traffic operations analysis," Washington, D.C., Tech. Rep. FHWA-TS-82-213, 1982.

[15] S. Algers, E. Bernauer, M. Boero, L. Breheret, C. D. Taranto, M. Dougherty, K. Fox, and J. F. Gabard, "Smartest project: Review of micro-simulation models," *EU project No: RO-97-SC*, vol. 1059, 1997.

[16] Q. Yang and H. Koutsopoulos, "A Microscopic Traffic Simulator

for evaluation of dynamic traffic management systems," *Transportation Research Part C*, vol. 4, no. 3, pp. 113–129, 1996.

[17] I. Prigogine and F. C. Andrews, "A Boltzmann like approach for traffic flow," *Operations Research*, vol. 8, no. 789, 1960.

[18] P. Nelson, D. Bui, and A. Sopasakis, "A novel traffic stream model deriving from a bimodal kinetic equilibrium," in *Proceedings of the 1997 IFAC meeting, Chania, Greece*, 1997, pp. 799–804.

[19] V. Shvetsov and D. Helbing, "Macroscopic dynamics of multilane traffic," *Physical Review E*, vol. 59, no. 6, pp. 6328–6339, 1999.

[20] J. Kuhl, D. Evans, Y. Papelis, R. Romano, and G. Watson, "The iowa driving simulator: An immersive research environment," *Computer*, vol. 28, no. 7, pp. 35–41, 1995.

[21] S. Bayarri, M. Fernandez, and M. Perez, "Virtual reality for driving simulation," *Commun. ACM*, vol. 39, no. 5, pp. 72–76, 1996.

[22] H. Wang, J. Kearney, J. Cremer, and P. Willemsen, "Steering behaviors for autonomous vehicles in virtual environments," in *Proc. IEEE Virtual Reality Conf.*, 2005, pp. 155–162.

[23] J. Cremer, J. Kearney, and P. Willemsen, "Directable behavior models for virtual driving scenarios," *Trans. Soc. Comput. Simul. Int.*, vol. 14, no. 2, pp. 87–96, 1997.

[24] P. Willemsen, J. Kearney, and H. Wang, "Ribbon networks for modeling navigable paths of autonomous agents in virtual environments," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 3, pp. 331–342, 2006.

[25] S. LaValle and S. Hutchinson, "Optimal motion planning for multiple robots having independent goals," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 912–925, 1998.

[26] P. Švestka and M. Overmars, "Coordinated path planning for multiple robots," *Robotics and Autonomous Systems*, vol. 23, no. 3, pp. 125–152, 1998.

[27] G. Sánchez and J. Latombe, "Using a PRM planner to compare centralized and decoupled planning for multi-robot systems," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2002, pp. 2112–2119.

[28] K. Kant and S. Zucker, "Toward efficient planning: the path-velocity decomposition," *International Journal of Robotics Research*, vol. 5, no. 3, pp. 72–89, 1986.

[29] M. Erdmann and T. Lozano-Pérez, "On multiple moving objects," *Algorithmica*, vol. 2, pp. 477–521, 1987.

[30] J. van den Berg and M. Overmars, "Prioritized motion planning for multiple robots," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2005, pp. 2217–2222.

[31] ——, "Kinodynamic motion planning on roadmaps in dynamic environments," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2007, pp. 4253–4258.

[32] "Next generation simulation program," June 2008, http://www.ngsim.fhwa.dot.gov/.

Jason Sewall is an Alumni Fellow and Ph.D. candidate in the Department of Computer Science at the University of North Carolina at Chapel Hill. He is a research assistant in the GAMMA group, and his research interests include computational fluid dynamics and traffic simulation. Jason has collaborated on papers with over 20 fellow researchers in his six years of graduate study.

Jur van den Berg received his M.S. degree from the University of Groningen, The Netherlands, and his Ph.D. degree from Utrecht Univerisity, The Netherlands, in 2003 and 2007, respectively. From 2007 to 2009 he was a postdoctoral researcher at the University of North Carolina at Chapel Hill. Currently he is a postdoctoral researcher at the University of California at Berkeley. His research interests include motion and path planning, navigation of virtual characters, and medical robotics.

Ming C. Lin is currently the Beverly W. Long Distinguished Professor of Computer Science at the University of North Carolina at Chapel Hill. Her research interests include physically-based modeling, haptics, robotics, real-time 3D graphics for virtual environments, geometric computing, and distributed interactive simulation. She has (co-)authored more than 190 refereed scientific publications, co-edited/authored three books, including "Applied Computation Geometry" by Springer-Verlag, "High-Fidelity Haptic Rendering" by Morgan-Claypool, and "Haptic Rendering: Foundations, Algorithms and Applications" by A.K. Peters.

She has received several honors and awards, including the NSF Young Faculty Career Award in 1995, Honda Research Initiation Award in 1997, UNC/IBM Junior Faculty Development Award in 1999, UNC Hettleman Award for Scholarly Achievements in 2002, Carolina Women's Center Faculty Scholar in 2008, Carolina's WOWS Scholar 2009–2011, and 6 best paper awards.

She has served as a program committee member for over 90 leading conferences on virtual reality, computer graphics, robotics, haptics and computational geometry and co-chaired over 20 international conferences and workshops. She is the Associate Editor-in-Chief of IEEE Transactions on Visualization and Computer Graphics (TVCG). She also has served as an associate editor and guest editor of over 15 journals and magazines.

Dinesh Manocha is currently a Phi Delta Theta/Mason Distinguished Professor of Computer Science at the University of North Carolina at Chapel Hill. He was selected an Alfred P. Sloan Research Fellow, received NSF Career Award in 1995 and Office of Naval Research Young Investigator Award in 1996, Honda Research Initiation Award in 1997, and Hettleman Prize for scholarly achievement at UNC Chapel Hill in 1998. He has also received more than 13 best paper and panel awards at many leading conferences.

Manocha's research interests include geometric computing, interactive computer graphics, Physics-based simulation and robotics. He has published more than 270 papers in these areas. Some of the software systems developed by his group on collision and geometric computations, interactive rendering, and GPU-based algorithms have been widely downloaded and used by leading commercial vendors. Manocha has served as a program committee member or program chair for more than 75 leading conferences and also served as a guest editor or member of editorial board of ten leading journals. He has supervised 40 MS and Ph.D. students. He has served as a PI or Co-PI on more than 55 grants. His research has been sponsored by AMD/ATI, ARO, DARPA, Disney, DOE, Honda, Intel, Microsoft, NSF, NVIDIA, ONR, RDECOM and Sloan Foundation.