Fidelity and Performance of State Fast-forwarding in Microscopic Traffic Simulations

PHILIPP ANDELFINGER, Oak Ridge National Laboratory, USA
YADONG XU, TUMCREATE, Singapore
DAVID ECKHOFF, TUMCREATE, Singapore & Technical University of Munich, Germany
WENTONG CAI, Nanyang Technological University, Singapore
ALOIS KNOLL, Technical University of Munich, Germany & Nanyang Technological University, Singapore

Common car-following models for microscopic traffic simulation assume a time advancement using fixed-sized time steps. However, a purely time-driven execution is inefficient when the states of some agents are independent of other agents and thus predictable far into the simulated future. We propose a method to accelerate microscopic traffic simulations based on identifying independence among agent state updates. Instead of iteratively updating an agent's state throughout a sequence of time steps, a computationally inexpensive "fast-forward" function advances the agent's state to the time of its earliest possible interaction with other agents. We present an algorithm to determine independence intervals in microscopic traffic simulations and derive fast-forward functions for several well-known traffic models. In contrast to existing approaches based on reducing the level of detail, our approach retains the microscopic nature of the simulation. An evaluation is performed for a synthetic scenario and on the road network of Singapore. At low traffic densities, maximum speedup factors of about 2.6 and 1.6 are achieved, while at the highest considered densities, only few opportunities for fast-forwarding exist. We show that the deviation from purely time-driven execution is reduced to a minimum when choosing an adequate numerical integration scheme to execute the time-driven updates. Verification results show that the overall deviation in vehicle travel times is marginal.

CCS Concepts: • **General and reference** → **Performance**; • **Computing methodologies** → **Modeling methodologies**; **Agent/discrete models**; **Continuous models**;

Majority of research performed while author was at Nanyang Technological University and TUMCREATE, Singapore.

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

This work was financially supported by the Singapore National Research Foundation under its Campus for Research Excellence and Technological Enterprise (CREATE) program.

Authors' addresses: P. Andelfinger, Oak Ridge National Laboratory; email: andelfingepj@ornl.gov; Y. Xu, TUMCREATE, Singapore; email: yadong.xu@tum-create.edu.sg; D. Eckhoff, TUMCREATE, Singapore & Technical University of Munich, Germany; email: david.eckhoff@tum-create.edu.sg; W. Cai, Nanyang Technological University, Singapore; email: david.eckhoff@tum-create.edu.sg; A. Knoll, Technical University of Munich, Germany & Nanyang Technological University, Singapore; email: knoll@in.tum.de.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.

1049-3301/2020/03-ART10 $15.00
https://doi.org/10.1145/3366019

Additional Key Words and Phrases: Fidelity, performance, microscopic traffic simulation, approach

ACM Reference format:
Philipp Andelfinger, Yadong Xu, David Eckhoff, Wentong Cai, and Alois Knoll. 2020. Fidelity and Performance of State Fast-forwarding in Microscopic Traffic Simulations. *ACM Trans. Model. Comput. Simul.* 30, 2, Article 10 (March 2020), 26 pages.
https://doi.org/10.1145/3366019

1 INTRODUCTION

Microscopic traffic simulation models represent the traffic in a road network on the level of individual vehicles that update their accelerations, velocities, and positions according to properties of their environment and nearby vehicles [27]. Often, simulation proceeds in a *time-driven* manner by the updating vehicles' states at fixed time steps. When considering scenarios spanning the road traffic of an entire city, this detailed simulation approach incurs substantial computational demands and long runtimes. Instead of the commonly applied time-driven execution, some previous works have considered an *event-driven* execution of agent-based simulations [5, 19, 34, 44]. In event-driven simulations, the model time is advanced across those points in model time where agent state changes occur, skipping intervals without state changes. Short intervals between state updates in common microscopic traffic simulations are required to accurately represent close interactions among vehicles, whereas vehicles isolated on the road behave in a predictable fashion.

Based on this observation, we propose an approach for accelerating independent agent state updates in microscopic traffic simulations using a computationally inexpensive *fast-forward function*, which updates the agent state to the first possible point in model time where an interaction may occur, skipping the intermediate updates. The approach retains the microscopic nature of the simulation. To apply the fast-forward function without violating the correctness of the simulation results, intervals in model time are identified during which agent interactions can be ruled out. A reduction in simulation execution time is achieved if the time spent on identifying such *independence intervals* is smaller than the time saved through the reduction in time steps. In essence, the proposed approach performs event-driven state updates for isolated vehicles, whereas time-driven updates are maintained for all other vehicles.

We propose an algorithm that predicts independence intervals efficiently for microscopic traffic simulations on road networks represented by graphs. Further, we derive a fast-forward function for several well-known car-following models that govern the acceleration behavior of the simulated vehicles: the Intelligent Driver Model [45], Gipps' model [17], Wiedemann's model [48], the Optimal Velocity Model [2], and Krauss' model [28]. The benefits of the proposed approach are evaluated in the city-scale microscopic traffic simulator CityMoS [52], both on a synthetic road network and on a representation of the road network of the city of Singapore.

The present article is an extended version of our previous conference publication [1]. Beyond the previous results, we demonstrate the generality of the approach by deriving scanning and fast-forward functions for several well-known car-following models, we investigate the fidelity achievable under several numerical integration schemes for the time-driven updates, and we explore the performance when extending the scanning mechanism towards a finer spatial granularity.

The remainder of this article is organized as follows: Section 2 sketches the technical background of our work. In Section 3, we describe the proposed fast-forwarding approach and derive scanning and fast-forward functions. Section 4 provides verification and performance evaluation results. Section 5 describes remaining limitations and potential enhancements of the approach. Section 6 discusses related work. Section 7 provides a summary of our results and concludes the article.

2 PRELIMINARIES

2.1 Agent-based Modeling and Simulation

In agent-based simulation, entities called agents are situated in an environment within the simulation space. An agent's environment is composed of static elements and nearby agents. At each point in model time, each agent has a *state* defined by a set of state variables. During a *state update*, an agent applies *update functions* to update the state variables according to the sensed environment. A *sensing range* limits the distance up to which the environment is considered. We refer to a state update that reads the state variables of nearby agents as an *interaction*.

Execution mechanisms for agent-based models can be classified into two categories: In a *time-driven* execution, the simulation proceeds in cycles. In each cycle, each agent may carry out a state update, which advances its state by a fixed delta in model time. In an *event-driven* execution, the model time advances only to those points in model time where state changes occur [37]. The proposed fast-forwarding approach combines time-driven and event-driven execution.

2.2 Microscopic Traffic Simulation

In microscopic traffic simulations, agents called driver-vehicle-units (DVUs) move through the simulation space according to models of the state and behavior of a human driver as well as of the vehicle operated by the driver. Typically, the simulation space is a road network modeled as a directed graph $G = (V, E)$, where edges represent roads with one or more lanes and vertices represent intersections. At each point in model time, each DVU is situated at a specific position on a lane within an edge.

DVUs perform state updates according to a car-following model (e.g., References [17, 45]) and a lane-changing model (e.g., References [18, 26]). Car-following models determine the acceleration of a vehicle according to the characteristics of the driver, the vehicle, and the surrounding traffic conditions. Commonly, the acceleration is chosen according to a desired safety gap to the vehicle ahead. Lane-changing models decide whether a DVU should change lanes, e.g., based on the current velocity and vehicles on other lanes. The distance up to which nearby DVUs are considered is limited by the sensing range. For simplicity, we refer to DVUs as agents or vehicles throughout the remainder of the article.

3 FAST-FORWARDING APPROACH

As introduced in the previous section, in an agent-based simulation, agents update their states according to their current environment and the states of neighboring agents within their sensing range. When an agent is spatially isolated from others, the current state update depends only on the environment, which may be static or highly predictable. The proposed approach is based on the observation that if it can be guaranteed that the agent remains isolated up to a certain point in model time, the agent's state can be updated to this point immediately in an event-driven fashion. By computing such updates using a fast-forward function that is less computationally expensive than a sequence of regular state updates, the overall execution time of the simulation can be reduced.

3.1 Problem Definition

In this section, we formally describe state updates in agent-based simulations using time-driven updates and the proposed fast-forward function. We loosely follow the formalization by Scheutz et al. [42], who studied agent interactions to limit state updates to those required for reaching the simulation's termination criterion or to decrease communication costs in distributed simulations. In contrast to their approach, which still relies on time-driven agent updates, fast-forwarding accelerates the simulation by avoiding state updates that are guaranteed to be independent of any other agents' states.

Let τ be the time step size of the simulation. An agent state update from model time t to $t + \tau$ can be represented as applying a *state update function* f_τ: $S_a^{t+\tau} = f_\tau(S_a^t, E_a^t, N_a^t)$, where S_a^t is the state of agent a at simulation time t. We differentiate between the environment E_a^t surrounding agent a at t and the set of neighboring agents N_a^t that are sensed by a at t. Let f_τ^k denote applying the state update function k times:

$$f_\tau^k\left(S_a^t, E_a^t, N_a^t\right) = f_\tau\left(f_\tau\left(\dots\left(f_\tau\left(S_a^t, E_a^t, N_a^t\right), E_a^{t+\tau}, N_a^{t+\tau}\right), \dots\right), E_a^{t+(k-1)\tau}, N_a^{t+(k-1)\tau}\right).$$

We introduce a *fast-forward function* F, which approximates the result of iteratively applying f_τ given $N_a^{t+i\tau} = \emptyset$, $i \in \{0, \dots, k-1\}$: $|F(k\tau, S_a^t, E_a^t) - f_\tau^k(S_a^t, E_a^t, \emptyset)| = \epsilon$, where ϵ is the approximation error. If agent a does not sense any other agent within the next k time steps, then the fast-forward function F successfully approximates the final state for agent a as if iteratively applying the state update function f_τ. However, since the sensing relation may not be symmetric and F does not yield the intermediate states in $(t, t + k\tau)$ required for sensing a, other agents' state updates may deviate when applying F for a. Thus, avoiding deviations across all agents' states requires mutual independence:

$$\forall i \in \{0, \dots, k-1\} : \left(\left(N_a^{t+i\tau} = \emptyset\right) \wedge \left(\forall a' \in A \setminus \{a\} : a \notin N_{a'}^{t+i\tau}\right)\right),$$

where A is the set of agents in the simulation, and $\wedge$ denotes logical conjunction. We refer to any interval $[t, t + k\tau)$ during which the above holds as an *independence interval.*

3.2 Overview of Our Approach

Algorithm 1 shows pseudo code of the main simulation loop when using fast-forwarding. The execution scheme maintains time-driven updates for agents that interact with others, whereas isolated agents are updated to a future point in model time in an event-driven fashion. The set A holds all agents present in the simulation, while at any point during the execution of the simulation, the set U holds only those agents that have not been fast-forwarded to a future point in model time. An event queue q holds events to be executed in timestamp order. Events are aligned to integer multiples of the time step size τ. The event types are associated with the following behavior:

- UPDATEAGENTS performs a state update for all agents in U from the current point in model time by one time step size τ. An execution scheme limited to this event type would coincide with a purely time-driven simulation.
- FASTFORWARDAGENTS periodically determines for each agent in U an independence interval during which interactions with other agents cannot occur. If the interval extends beyond a single time step into the future, then the agent is updated to the end of the independence interval by applying a model-specific fast-forward function. The agent is then removed from U and a REINSERTAGENT event is scheduled at the end of the independence interval.
- REINSERTAGENT inserts a fast-forwarded agent into U to allow it to be considered in the next UPDATEAGENTS event.

The execution order for events at the same model time is REINSERTAGENT, FASTFORWARDAGENTS, UPDATEAGENTS.

The fast-forwarding approach neither necessitates nor precludes parallelization of the simulation. In our performance evaluation, we apply trivial parallelization to the identification of independence intervals, whereas all other parts of the simulation are executed sequentially.

ALGORITHM 1: Main simulation loop with fast-forwarding.

```
1: $U \leftarrow A$
2: $Q.pushEvent(type\text{: UPDATEAGENTS}, time\text{: } 0)$
3: $Q.pushEvent(type\text{: FASTFORWARDAGENTS}, time\text{: } scanningPeriod)$
4: while $!terminate()$ do
5:     $ev \leftarrow Q.popEvent()$
6:     if $ev.type = \text{UPDATEAGENTS}$ then
7:         for each $a \in U$ do
8:             $a.updateState()$
9:         $Q.pushEvent(type\text{: UPDATEAGENTS}, ev.time{+}\tau)$
10:    else if $ev.type = \text{FASTFORWARDAGENTS}$ then
11:        $identifyIndependenceIntervals(U)$
12:        for each $a \in U$ do
13:            if $a.independenceIntervalEnd > ev.time + \tau$ then
14:                $a.fastForward(a.independenceIntervalEnd)$
15:                $U \leftarrow U \setminus \{a\}$
16:                $Q.pushEvent(type\text{: REINSERTAGENT}, time\text{: } a.independenceIntervalEnd, agent\text{: } a)$
17:        $Q.pushEvent(type\text{: FASTFORWARDAGENTS}, time\text{: } ev.time + scanningPeriod)$
18:    else if $ev.type = \text{REINSERTAGENT}$ then
19:        $U \leftarrow U \cup \{ev.agent\}$
```

3.3 Identifying Independence Intervals

To allow for the identification of independence intervals, Scheutz et al. define a *translation function* that "determines for a given location the maximum distance an agent can travel within one update" [41]. By determining the area that agents may travel to within the next k updates, independence intervals can be identified. In contrast to the translation function, the proposed fast-forward function determines the *full* agent state after k updates in case of independence from other agents' states. Thus, in contrast to the iterative time steps used by Scheutz et al., the fast-forward function allows for agent state updates across multiple time steps through a single function evaluation. We assume that the fast-forward function is accompanied by a *scanning* function similar to Scheutz' translation function, which yields the time at which an agent first arrives at a given target distance if independence from other agents' states is given.

In this section, we propose methods for identifying opportunities for fast-forwarding. First, we formulate conditions under which agents can be fast-forwarded on individual graph edges. Subsequently, we propose an algorithm to identify fast-forwarding opportunities across sequences of edges. We assume that the simulation space, i.e., the road network, is represented by a graph $G = (V, E)$ composed of a set of directed edges E representing roads and a set of vertices V representing intersections. During an agent's lifetime, the agent traverses a predefined sequence of connected edges. Each edge traversal may require multiple state updates. Interactions with other agents may increase the number of updates required to traverse an edge. Each edge has an assigned weight l representing its length, l being at least the sensing range. For simplicity, in our description, we disregard the spatial extent of the agents themselves, which we do, however, consider in our implementation of the approach.

3.3.1 *Single-link and Sub-link Scanning.*

If an agent a is driving at the speed limit of an edge and is located sufficiently far from all other agents so it is guaranteed that a will remain outside any other agent's sensing range for a certain amount of time, then a is eligible for fast-forwarding. To limit our consideration to the agent's current edge, we ensure that a cannot yet sense the next

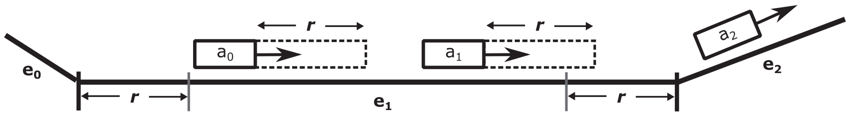


Fig. 1. Identifying independence intervals on a single edge. Two agents a_0 and a_1 are moving along edge e_1 and have already reached the speed limit. Agent a_1 will not interact with any other agent at least until it senses edge e_2. Agent a_0 cannot be sensed from e_0 and will not interact with any agent at least until it senses the position of a_1 at the time of scanning.

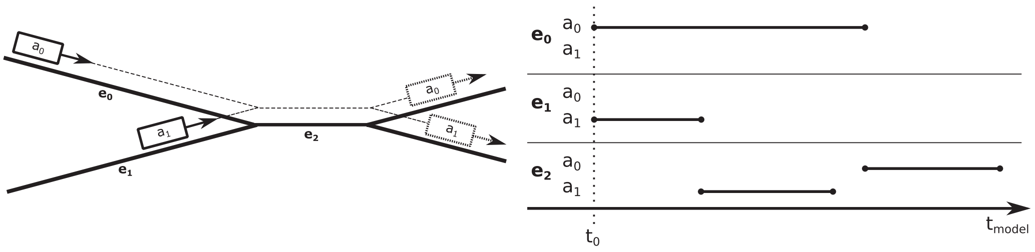


Fig. 2. Identifying independence intervals across multiple edges. Left: two agents a_0 and a_1 passing the same edge e_2. Right: occupancy intervals for edges e_0, e_1, and e_2. Since a_0 and a_1 never share their starting edges with other agents, they can be fast-forwarded across e_0 and e_1, respectively. Further, since a_0 and a_1 never occupy e_2 at the same time, both can be fast-forwarded across e_2. Thus, at minimum, the independence intervals of both a_0 and a_1 extend to the time at which the successor edge to e_2 is sensed.

edge on its route and cannot be sensed from the previous edge. More formally, an independence interval for agent a covers the time interval during which all of the following conditions c_1 to c_4 hold:

$$c_1\colon v(a) = v_{\max}, \qquad c_2\colon p(a) > r,$$
$$c_3\colon p(a) < l - r, \qquad c_4\colon \forall a' \in \bar{A} \setminus \{a\} : |p(a') - p(a)| > r,$$

where $v(a)$ and $p(a)$ are agent a's current velocity and position on the current edge, $\bar{A}$ is the set of agents on the same edge as a, r is the sensing range, l is the length of the current edge, and $v_{\max}$ is the speed limit. Figure 1 illustrates the identification of independence intervals on a single link.

We propose two efficient scanning methods: in *single-link scanning*, we replace c_4 by the stricter condition $\forall a' \in \bar{A} \setminus \{a\} : p(a') < p(a) - r$, i.e., fast-forwarding is allowed for the most ahead vehicle on the edge. The second variant, which we refer to as *sub-link scanning*, applies c_4 without modification, which may expose further fast-forwarding opportunities. In both variants, we only consider the positions of the vehicles at the model time when the scanning is carried out. This guarantees correctness but may not produce independence intervals of the largest possible size.

3.3.2 Multi-link Scanning. We now extend the identification of independence intervals to sequences of graph edges, as shown in Figure 2. Algorithm 1 determines for each agent an interval during which the agent never shares an edge with another agent. The algorithm proceeds in two stages: In the first stage, each agent registers its occupancy intervals at the edges that may be traversed within a configurable *scanning horizon*. The scanning horizon limits the scanning overheads. Each edge stores the earliest time it is sensed by any agent (*occupiedFrom*) with an initial value of ∞. If a registering agent exits the edge earlier than the current value of *occupiedFrom*, then we store the agent as a candidate for fast-forwarding (*earliestAgent*) together with its sensing time and exit time. Otherwise, the registering agent may interact with a previously registered agent; thus, we set *earliestAgent* to *nil*.

ALGORITHM 2: Identifying independence intervals across multiple graph edges.

```
1: procedure stageOne
2:     for each $a \in A$ do
3:         $e \leftarrow a.currentEdge$
4:         $sensingTime \leftarrow currentTime$
5:         $exitTime \leftarrow currentTime + travelTime(a, e, e.length - a.position)$
6:         $register(e, a, sensingTime, exitTime)$
7:         $sensingTime \leftarrow currentTime + travelTime(a, e, e.length - a.position - sensingRange)$
8:         $e \leftarrow a.getSuccessorEdgeOnRoute(e)$
9:         while $e \neq nil$ and $exitTime \leq currentTime + scanningHorizon$ do
10:            $nextSensingTime \leftarrow exitTime + travelTime(a, e, e.length - sensingRange)$
11:            $exitTime \leftarrow exitTime + travelTime(a, e, e.length)$
12:            $register(e, a, sensingTime, exitTime)$
13:            $sensingTime \leftarrow nextSensingTime$
14:            $e \leftarrow a.getSuccessorEdgeOnRoute(e)$
15: procedure stageTwo
16:     for each $a \in A$ do
17:         $a.independenceIntervalEnd \leftarrow currentTime$
18:         $e \leftarrow a.currentEdge$
19:         while $e \neq nil$ do
20:             if $e.earliestAgent \neq a$ then
21:                 break
22:             $e \leftarrow a.getSuccessorEdgeOnRoute(e)$
23:             $a.independenceIntervalEnd \leftarrow e.occupiedFrom$
24: procedure register($e, a, sensingTime, exitTime$)
25:     if $exitTime \geq currentTime + scanningHorizon$ then
26:         $exitTime \leftarrow \infty$
27:     if $exitTime < e.occupiedFrom$ then
28:         $e.earliestAgent \leftarrow a$
29:         $e.earliestAgentExitTime \leftarrow exitTime$
30:     else if $sensingTime \leq e.earliestAgentExitTime$ then
31:         $e.earliestAgent \leftarrow nil$
32:     $e.occupiedFrom \leftarrow min(e.occupiedFrom, sensingTime)$
```

In the second stage, each agent again iterates through the edges reachable within the scanning horizon. Starting at an agent a's current edge, agent a can be fast-forwarded across the longest sequence of edges with $earliestAgent = a$ and with exit times within the scanning horizon.

By limiting fast-forwarding to the agent who first occupies an edge, some opportunities are not exploited. An example is given in Figure 2: Although agent a_0 never occupies edge e_2 at the same time as a_1, agent a_0 will not be fast-forwarded across edge e_2. To limit the costs of the scanning process, we do not consider such situations.

The size of the independence intervals depends on the scanning horizon as well as the period in model time after which single-link and multi-link scanning are repeated, which must be balanced with the incurred scanning overhead. The simulation model and scenario define upper bounds for the independence intervals through the agents' velocity, the sensing range in relation to the lengths of the edges, as well as the density of the traffic. Our implementation evaluated in Section 4 applies single-link and multi-link scanning with a configurable period length in model time. The effects of varying the scanning parameters according to the considered simulation model and scenario are evaluated in Section 4.2.

3.4 Scanning and Fast-forward Functions

Fast-forwarding relies on a scanning and a fast-forward function to identify independence intervals and the future state of an agent based on its current state and the environment.

(1) As described in Section 3.3, the identification of independence intervals relies on a scanning function $S : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R} \times \mathbb{R}$, which given a position p and the current velocity v yields the model time at which the considered vehicle will reach p and its velocity at that time.
(2) To be able to resume time-driven updates after fast-forwarding an agent, we round the time to which the agent can be fast-forwarded to the nearest smaller time step. Given the rounded time t and the current velocity, the fast-forward function $F : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R} \times \mathbb{R}$ yields the position and velocity reached at t.

The focus of car-following models is the acceleration behavior of a following vehicle f depending on the state of a leading vehicle l. In fact, some models require the presence of a leading vehicle to produce plausible acceleration values. For instance, the acceleration value generated by the well-known General Motors model [6] tends towards infinity when the distance to the leading vehicle tends towards infinity. To apply such models in microscopic traffic simulations, the free driving behavior must be specified separately. The scanning and fast-forward functions can then be derived for the equations governing the free driving.

3.4.1 Derivation Procedure. In the following, we derive scanning and fast-forward functions for several well-known car-following models. Many car-following models are specified as differential equations $\frac{dv_\mathrm{f}}{dt} = g(v_\mathrm{f}, v_\mathrm{l}, p_\mathrm{f}, p_\mathrm{l})$, where v_i and p_i are the velocity and position of vehicle i at model time t. The vehicle interactions in common traffic scenarios lead to systems of coupled differential equations, which are usually solved using numerical integration with a fixed time step size.

Some models express the acceleration with respect to discrete time steps of size τ, resulting in the form $a(t + \tau) = g_\tau(v_\mathrm{f}, v_\mathrm{l}, p_\mathrm{f}, p_\mathrm{l})$. However, even for models specified in a continuous manner, the mobility in microscopic traffic simulations is typically simulated by numerical integration of the acceleration function over time and with a fixed time step size. Thus, for our purposes, the two ways of specifying the acceleration behavior are equivalent. The derivation of the scanning and fast-forward functions typically takes the following form:

- We obtain the free driving behavior by determining $\lim_{\Delta p \rightarrow \infty} \frac{dv}{dt}$ with $\Delta p = p_\mathrm{l} - p_\mathrm{f}$, which yields an equation that is independent of the leading vehicle's state: $\frac{dv_\mathrm{f}}{dt} = h(v_\mathrm{f})$.
- After separation of variables, we have $dt = \frac{dv}{h(v)}$, for which direct integration yields a function $t(v) = \hat{t}(v) + C_t$. We correct for the initial velocity v_0 and the constant of integration by setting $C_t = -\hat{t}(v_0)$.
- Solving for v, we obtain $v(t)$, which can be interpreted as the velocity reached at time t.
- Using the relationship $p(v) = \int \frac{v}{h(v)} dv$, we obtain $p(v) = \hat{p}(v) + C_p$ and set $C_p = -\hat{p}(v_0)$.
- Solving for v, we obtain $v(p)$, which is the velocity reached at position p.

Now, the scanning function is $S(p, v_0) = (t(v(p)), v(p))$. The fast-forward function is $F(t, v_0) = (p(v(t)), v(t))$.

As an example, in the trivial case of a constant acceleration a_0, the above procedure yields the functions $v(p) = \sqrt{2a_0 d}$, $p(v) = v^2/(2a_0)$, $t(v) = v/a_0$, and $v(t) = a_0 t$.

In the following, we derive fast-forward and scanning functions for several common car-following models. The derivations were carried out using Mathematica.

3.4.2 Intelligent Driver Model. In the Intelligent Driver Model (IDM), vehicles accelerate according to the following differential equation [45]:

$$\frac{dv}{dt} = a_0\left(1 - \left(\frac{v}{v_\mathrm{d}}\right)^\delta - \left(\frac{s_0 + vT + (v\Delta v)/(2\sqrt{a_0 b_0})}{\Delta p}\right)^2\right).$$

Here, a_0 is the maximum acceleration, v is the current velocity, v_d is the target velocity, s_0 is the minimum desired distance to the vehicle ahead, b_0 is the comfortable braking deceleration, and Δp and Δv are the position and velocity differences to the leading vehicle. δ is a tuning parameter typically set to 4 [45]. We perform our computations for this value.

For $\Delta p \to \infty$, the acceleration is determined solely by the *free road term*:

$$\frac{dv}{dt} = a_0\left(1 - \left(\frac{v}{v_\mathrm{d}}\right)^\delta\right).$$

For $\delta = 4$, integration after separation of variables yields the time required to accelerate from 0m/s to v:

$$\hat{t}(v) = \frac{v_\mathrm{d}}{4a_0}\left(\log(v_\mathrm{d} + v) - \log(v_\mathrm{d} - v) + 2\arctan\left(\frac{v}{v_\mathrm{d}}\right)\right).$$

When accelerating from initial velocity $v_0 > 0$m/s, the time elapsed when velocity v is reached is:

$$t(v) = \hat{t}(v) - \hat{t}(v_0).$$

The distance that the vehicle has traveled when reaching velocity v can be obtained as follows:

$$\hat{p}(v) = \int \frac{v}{a(v)}dv = \frac{v_\mathrm{d}^2}{2a_0\mathrm{arctanh}((\frac{v}{v_\mathrm{d}})^2)}.$$

Solving for v:

$$v(p) = v_\mathrm{d}\sqrt{\tanh\left(\frac{2a_0 p}{v_\mathrm{d}^2}\right)}.$$

We now have the basic functions $v(p)$, $p(v)$, and $t(v)$. However, we do not have a closed form for $v(t)$, which required to formulate the fast-forward function. Since $t(v)$ is twice differentiable, we can postulate $t(v) - t = 0$ and apply Halley's root-finding method [16] to compute v numerically at cubical convergence speed.

An extension to IDM has been proposed to apply decelerations when advancing to a road with a speed limit below the current velocity [27]:

$$\frac{dv}{dt} = -a_0\left(1 - \left(\frac{v_\mathrm{d}}{v}\right)^\delta\right).$$

We derive fast-forward and scanning functions for this situation as well. For $\delta = 4$, integration after separation of variables yields:

$$\hat{t}(v) = -\frac{1}{2a_0}\left(v_\mathrm{d}\left(\arctan\left(\frac{v_\mathrm{d}}{v}\right) - \mathrm{arctanh}\left(\frac{v_\mathrm{d}}{v}\right)\right) + 2v\right).$$

We can obtain the distance at velocity v as follows:

$$\hat{p}(v) = \int \frac{v}{a(v)}dv = \frac{1}{2a_0}\left(\left(v_\mathrm{d}^2 - v^2\right)\mathrm{arctanh}\left(\left(\frac{v_\mathrm{d}}{v}\right)^2\right)\right).$$

We apply Halley's method to obtain $v(p)$ and $v(t)$ and proceed as above. In the evaluation in Section 4.1.1, we set $a_0 = 3.0\frac{m}{s^2}$.

3.4.3 Gipps' Model. The model [17] by Gipps is another widely employed car-following model. It is used in the microscopic traffic simulator AIMSUN [3]. Ciuffo et al. give an overview of the analysis and applications of the model [8].

In contrast to IDM, the model is specified with respect to discrete time steps. However, as discussed above, we can proceed as if the acceleration behavior was given as a differential equation. Two separate equations are given for the free driving behavior and the presence of a leading vehicle. We are interested in the former case, for which the acceleration is specified as follows:

$$\frac{dv}{dt} = a_0 c_1 (1 - v/v_\mathrm{d})\sqrt{c2 + v/v_\mathrm{d}}.$$

By the same process as above, we obtain:

$$\begin{aligned}
\hat{t}(v) &= \frac{2v_\mathrm{d}\mathrm{atanh}(\sqrt{(c_2 v_\mathrm{d} + v)/(c_2 v_\mathrm{d} + v_\mathrm{d})})}{a_0 c_1 \sqrt{c2+1}},\\
v(t) &= v_\mathrm{d}(c_2 T + T - c_2) \ \text{ with } \ T = \tanh^2\left(\frac{a_0 c_1 t\sqrt{c2+1}}{2v_\mathrm{d}}\right),\\
\hat{p}(v) &= \frac{\mathrm{atanh}(\frac{\sqrt{c_2 + v/v_\mathrm{d}}}{\sqrt{1+c_2}}) - 2v_\mathrm{d}^2\sqrt{1+c_2}\sqrt{v/v_\mathrm{d}+c_2}}{a_0 c_1 \sqrt{1+c_2}}.
\end{aligned}$$

Since we do not have a closed form for $v(p)$, we apply Halley's method to solve $p(v) - p = 0$. The evaluation in Section 4.1.1 will rely on the parameters from Reference [17]: $c_1 = 2.5, c_2 = 0.025$. The per-driver constant a_0 is sampled from $N(1.7, 0.09)$.

3.4.4 Wiedemann's Model. An early car-following model proposed by Wiedemann [48] is used in the commercial simulator VISSIM. The acceleration behavior is governed by different equations depending on the distance and velocity of the following and leading vehicle. The free driving behavior relies on the equation: $\frac{dv}{dt} = c_1(c_2 - c_3 v)$, with parameters c_1, c_2, c_3. In this model, the velocity approaches $v_\mathrm{d} = c_2/c_3$. We obtain:

$$\begin{aligned}
v(p) &= \frac{c_2}{c_3}\left(1 + Re\left(W\left(\frac{-e^{-1-(c_1 c_3^2 d)/c_2}}{c_2}\right)\right)\right), & v(t) &= \frac{c_2 - e^{-c_1 c_3 t}}{c_3},\\
\hat{t}(v) &= -\frac{\log(c_2 - c_3 v)}{c_1 c_3}, & \hat{p}(v) &= -\frac{c_3 v + c_2 \log(c_2 - c_3 v)}{c_1 c_3^2}.
\end{aligned}$$

Due to the logarithm in $p(v)$, the function $v(p)$ is expressed in terms of the Lambert W function, which is defined implicitly and must thus be evaluated numerically [9]. For simplicity, we apply Halley's method to determine v as the root of $p(v) - p$.

In contrast to most other models, once v_d has been reached, slight deviations from this speed are modeled using a random acceleration $-b_\mathrm{null}$ or $+b_\mathrm{null}$ at each time step, wherein b_null itself may be a random variable.

During scanning and fast-forwarding, we assume that once the driver has reached v_d, this velocity is maintained exactly. Thus, the fidelity of a vehicle's velocity and position after fast-forwarding compared to time-driven state updates depends on the distribution of b_null. Our approach ignores the stochasticity in the model during fast-forwarding. By drawing from the sampling distribution of the sum of random variables with the distribution of b_null, it would be possible to instead retain the variance introduced by b_null.

The evaluation in Section 4.1.1 will rely on the parameters from Reference [23], in which a fixed b_{null} is used: $b_{\text{null}} = 0.22, c_1 = 0.004, c_2 = c_3 v_{\text{d}}, c_3 = 0.496$.

3.4.5 Optimal Velocity Model. The Optimal Velocity Model was proposed by Bando et al. [2]. We consider the form relayed by Helbing and Tilch [22]: $\frac{dv}{dt} = k_1(V(\Delta p) - v)$ with $V(\Delta p) = V_1 + V_2 \tanh(k_2 \Delta p - k_3)$. For $\Delta p \to \infty$, $V(\Delta p)$ tends towards $V_1 + V_2$, and $a(v)$ tends towards $k_1(V_1 + V_2 - v)$. This is a special case of the free driving behavior of Wiedemann's model with $c_1 = k_1, c_2 = V_1 + V_2, c_3 = 1, c_4 = 0$. The evaluation in Section 4.1.1 will rely on the parameters from Reference [22]: $k_1 = 0.85, V_1 = 6.75, V_2 = 7.91$.

3.4.6 Krauss' Model. Krauss investigates a class of models of the following discretized form [28]:

$$\begin{aligned}
v_{\text{safe}}(t+\tau) &= v_{\text{f}}(t) + \frac{\Delta p - \Delta p_{\text{d}}}{\tau_{\text{b}} + \tau}, \\
v_{\text{d}}(t+\tau) &= \min(v_{\max}, v_{\text{f}}(t) + \tau a(v_{\text{f}}(t)), v_{\text{safe}}(t)), \\
v_{\text{f}}(t+\tau) &= \max(0, v_{\text{d}}(t) - \eta),
\end{aligned}$$

where Δp and Δp_{d} are the current and desired gaps between the leader and follower, τ is the driver reaction time, $\tau_{\text{b}} = \frac{v_{\text{f}} + v_{\text{l}}}{2b}$, the constant b is the maximum deceleration. The function $a(v)$ yields the acceleration at a given velocity v. The random variable η models the inaccuracies in the driver's acceleration behavior.

Since v_{safe} tends towards infinity for $\Delta p \to \infty$, the desired velocity v_{d} tends towards $\min(v_{\max}, v_{\text{f}} + \tau a(v_{\text{l}}))$. Until $v_{\max}$ has been reached and assuming $v_{\text{d}}(t) - \eta > 0$, the expectation of the acceleration with $\tau = 1$ is thus $a(v_{\text{f}}) - \mathbb{E}(\eta)$. As with Wiedemann's model, the sampling distribution of sums of random variables with the distribution of η could be computed to retain the stochasticity of the model instead.

If $a(v_{\text{f}})$ is a constant function, then fast-forward and scanning functions are trivially derived as described in Section 3.4.1. The evaluation in Section 4.1.1 will rely on parameters from Reference [28]: $v_{\max} = 36m/s, a(v) = 0.8m/s^2$. The value of η is sampled from $U(0, 0.4\tau)$.

3.5 Discussion

Most car-following models are defined by a time-continuous differential equation specifying a vehicle's acceleration behavior. Time-driven microscopic traffic simulations approximate the specified behavior by calculating new acceleration values at each time step and updating the vehicles' velocities and positions accordingly. Smaller time step sizes increase the fidelity of the approximation but are associated with higher computational cost. In contrast, the fast-forward function produces a "smooth" acceleration behavior without discretization to intermediate time steps. As such, while the results when using fast-forwarding deviate from those of a purely time-driven execution, the former can in fact be considered more in line with the model specification. A further potential source of deviations is due to determining the occupancy intervals using the scanning function. The intervals are therefore affected by deviations as well. When a vehicle does not approach a road segment using the fast-forward function but using iterative time steps, the predicted occupancy interval may slightly deviate from the observed interval during which the vehicle actually occupies the road segment. Thus, it is possible that a vehicle is fast-forwarded based on an erroneous prediction that the vehicle will be isolated on the road segment. In the next section, we study the error introduced both by individual fast-forwarding operations and in full simulations on two different road networks.

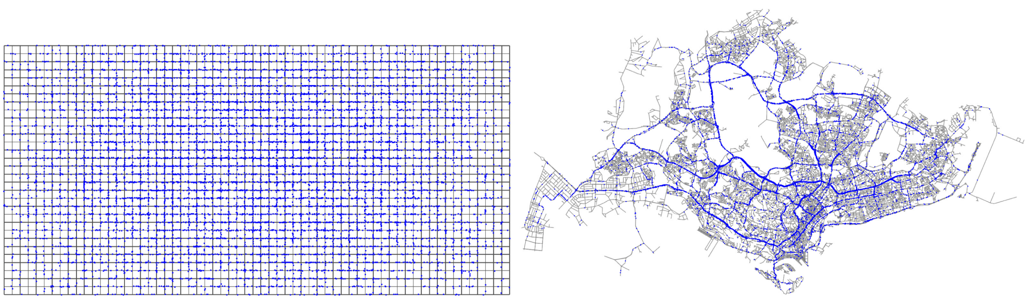

(a) Grid road network, around 10 000 vehicles. (b) Singapore road network, around 19 500 vehicles.

Fig. 3. Considered road networks with traffic, blue dots denoting vehicles.

4 EVALUATION

In this section, we aim to answer the following questions:

- *How large is the deviation in the simulation results between a purely time-driven execution and the proposed fast-forwarding approach?*
- *In which scenarios and to which degree can fast-forwarding accelerate the simulation?*

For the evaluation of the fidelity of individual fast-forwarding operations, we compare results when simulating a single vehicle on a single-lane road for the car-following models analyzed in Section 3. Since the fidelity depends strongly on the numerical integration scheme used for the time-driven updates, we compare the results when using four common integration schemes.

Our evaluation using full simulation runs is performed using the city-scale microscopic traffic simulator CityMoS [52]. The implementation of the scanning procedure follows the description in Section 3.3, with trivial parallelization across the agents using OpenMP. Compared to the implementation used in our previous experiments [1], we carried out minor code optimizations and introduced sub-link scanning (cf. Section 3.3.1).

Two road networks are considered: a synthetic grid-shaped road network (cf. Figure 3(a)) and a representation of the road network of Singapore (cf. Figure 3(b)). The grid network is composed of 64×32 rectangles, each edge being 200 m in length. There are two edges between two adjacent vertices with opposite traffic directions, resulting in a total length of 1600 km. In the Singapore network, the average edge length is 93.2m. In comparison to the experiments in our previous publication [1], the road network has been processed to merge redundant edges. In both scenarios, origin and destination pairs are chosen uniformly at random on the road network. Route planning is based on Dijkstra's algorithm, using the edges' lengths and speed limits as their weights. Agents start their trips at points in model time chosen uniformly at random. In the grid scenario, we started the measurements after a warm-up phase of 1,800s to achieve roughly constant agent populations of 500, 2,000, and 10,000. After the warm-up phase, each measurement continued for 1 h of model time. In the Singapore scenario, we used a warm-up phase of 1,800 s and subsequently measured the performance for 1 h of model time while the agent population increases to about 4,800, 12,600, and 24,700 agents, respectively. Vehicles accelerate according to the Intelligent Driver Model [45] and perform lane changes according to the rules described in Reference [49]. We configured a sensing range of 40m facing forward. We varied the following algorithm parameters:

- **Single-link and multi-link scanning period**: the identification of independence intervals and the fast-forwarding are performed periodically. Since the scanning overhead

depends on whether individual graph edges or sequences of graph edges are considered, the period length for each variant is varied separately.
- **Scanning horizon**: the overhead of scanning and the size of independence intervals both depend on the maximum delta in model time that agents may be fast-forwarded.

We repeated the performance measurements using sub-link scanning, which is applied when single-link scanning fails to identify a fast-forwarding opportunity. For the grid scenario, we performed a parameter sweep to study the effect of different parameter combinations on the simulation performance. The levels in seconds of model time were $\{0.5, 2, 8, 32\}$ and $\{0.5, 2, 8, 32, 128\}$ for the single-link and multi-link scanning periods, and $\{16, 64, 256\}$ for the scanning horizon. In the Singapore scenario, we applied a simple auto-tuning approach to select and vary parameter combinations at runtime: a sequence of preconfigured parameter combinations is set one after the other, measuring the simulation progress per unit wall-clock time for each combination. The simulation then proceeds with the best-performing parameter combination. The auto-tuning process is repeated once either after 1,200s of model time have passed or the simulation performance has changed by more than a factor of 2. We allow the auto-tuning to choose from the following combinations of single-link scanning period, multi-link scanning period, and scanning horizon, each in seconds of model time: (2, 32, 64), (2, 128, 64), (4, 32, 64), (4, 128, 64), and (2, ∞, 64). Each simulation run was repeated at least three times. All performance measurements were performed on a 3.00 GHz Intel i5-7400 CPU with 16 GiB of RAM running Ubuntu 16.04 and using GCC 5.4.0 for compilation.

4.1 Verification

After a fast-forwarding operation, an agent's velocity and position may deviate from the results of purely time-driven state updates. Since the numerical integration used in time-driven updates only approximates the true value of the acceleration equation integrated over time, an error is introduced. The magnitude of the error depends on the numerical integration scheme.

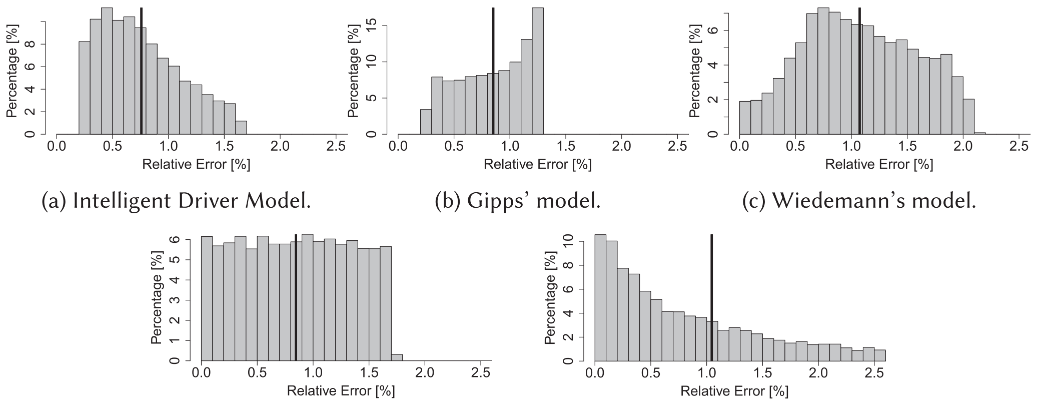
In the following, we first quantify the deviation of a vehicle's position after a fast-forwarding operation when compared to time-driven state updates based on different numerical integration schemes. We then compare aggregated statistics over all vehicles in full simulation runs.

4.1.1 Individual Fast-forwarding Operations. We study the absolute deviation in a vehicle's position after driving for 30s on a single-lane road using fast-forwarding compared to time-driven state updates. Since, on multi-lane roads, isolated agents are free to change to their preferred lane, the selection of the correct lane during fast-forwarding is trivial and not verified separately. The starting velocity in m/s was drawn uniformly at random from the interval $[0, 30]$. Where the model parameters allow us to set the speed limit directly, it was set to 36m/s. The time step size τ was set to 0.1s and 0.5s. For each combination of model, integration method, and time step size, $10,000$ repetitions were performed.

The deviations between fast-forwarding and time-driven updates are due to the error introduced by the numerical integration used for the time-driven updates. Thus, given each model's acceleration behavior specified by a function f, we evaluate the deviations when performing the time-driven updates using each of the four numerical integrations studied by Treiber and Kanagaraj [46]:

(1) Forward Euler:

$$a(t+\tau) = f(v(t)), \qquad v(t+\tau) = v(t) + a(t+\tau)\tau, \qquad p(t+\tau) = p(t) + v(t+\tau)\tau.$$



(a) Intelligent Driver Model. (b) Gipps' model. (c) Wiedemann's model.

(d) Optimal Velocity model. (e) Krauss' model.

Fig. 4. Relative deviation between fast-forwarding and time-driven updates using the forward Euler scheme with $\tau = 0.5$s. A vertical line shows the mean.

(2) Ballistic update [25]:

$$a(t+\tau) = f(v(t)), \quad v(t+\tau) = v(t) + a(t+\tau)\tau, \quad p(t+\tau) = p(t) + 1/2(v(t) + v(t+\tau))\tau.$$

(3) Trapezoid rule:

$$\begin{aligned} v_1 &= v(t), & a_1 &= f(v_1), & v(t+\tau) &= v(t) + 1/2(a_1 + a_2)\tau, \\ v_2 &= v + \tau a_1, & a_2 &= f(v_2), & p(t+\tau) &= p(t) + 1/2(v_1 + v_2)\tau. \end{aligned}$$

(4) Fourth-order Runge-Kutta (RK4):

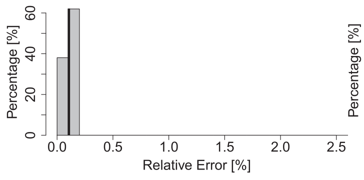
$$\begin{aligned} v_1 &= v(t), & a_1 &= f(v_1), & v(t+\tau) &= v(t) + 1/6(a_1 + 2a_2 + 2a_3 + a_4)\tau, \\ v_2 &= v + 1/2a_1\tau, & a_2 &= f(v_2), & p(t+\tau) &= p(t) + 1/6(v_1 + 2v_2 + 2v_3 + v_4)\tau. \\ v_3 &= v + 1/2a_2\tau, & a_3 &= f(v_3), & & \\ v_4 &= v + a_3\tau, & a_4 &= f(v_4), & & \end{aligned}$$

Per time step, the schemes rely on one, two, or four evaluations of the differential equation to approximate the integral over time. As discussed in Section 3, Wiedemann and Krauss introduce stochasticity into their models by prescribing discrete-time updates based on the forward Euler scheme and drawing random numbers at each update. Since adaptations to these models would be needed to support update schemes that require multiple evaluations per time step, we evaluate these two models only for the forward Euler and ballistic update schemes.

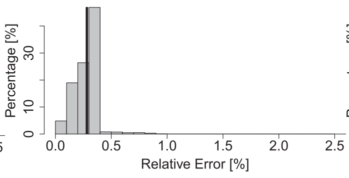
Treiber and Kanagaraj previously evaluated the fidelity achieved by the above set of integration schemes [46]. Since the movement of interacting vehicles is analytically tractable only for trivial scenarios, they compare the results of numerical integration with τ between 0.002s and 2.4s to those achieved by a reference simulation using $\tau = 10^{-4}$s. In contrast, since fast-forwarding applies to isolated vehicles only, the reference result can be determined directly from the continuous formulation of the acceleration behavior.

The total distance driven after 30s differs among the models depending on the respective acceleration behavior, with averages of about 700m for IDM and Krauss' model, 855m for Gipps' model, 430m for OVM, and 920m for Wiedemann's model.

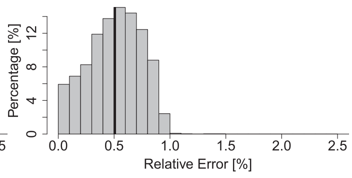
Figure 4 shows the verification results for the forward Euler scheme with a time step size τ of 0.5s. The mean relative error is around 1% for all car-following models. Due to the stochastic



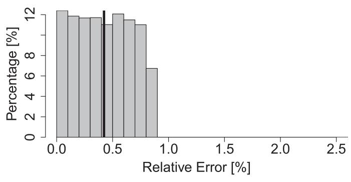
(a) Intelligent Driver Model.



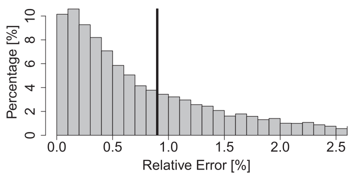
(b) Gipps' model.



(c) Wiedemann's model.



(d) Optimal Velocity model.



(e) Krauss' model.

Fig. 5. Relative deviation between fast-forwarding and time-driven updates using the ballistic update scheme with $\tau = 0.5$s. A vertical line shows the mean.

Table 1. Relative Deviation [%] between Fast-forwarding and Time-driven Updates after 30s, Aggregated across All Considered Car-following Models, Excluding Krauss' and Wiedemann's Models for Trapezoid Rule and RK4

	Euler		**Ballistic**		**Trapezoid**		**RK4**	
τ	**Mean**	**Max.**	**Mean**	**Max.**	**Mean**	**Max.**	**Mean**	**Max.**
0.1	0.23	4.06	0.14	4.73	8.56×10^{-5}	3.46×10^{-3}	9.31×10^{-8}	2.79×10^{-7}
0.5	0.91	8.21	0.44	8.67	2.03×10^{-3}	7.73×10^{-2}	1.37×10^{-6}	1.48×10^{-4}

elements of Krauss' model, the relative error is heavy-tailed. Figure 5 shows the results for ballistic update. As before, τ is set to 0.5s. Although the integration scheme also relies on only one evaluation of the acceleration function per time step, the relative error is strongly reduced for most of the models. Due to the strong stochastic influence in Krauss' model under the chosen parameters, the relative error for Krauss' model is roughly the same with ballistic update as with the forward Euler scheme.

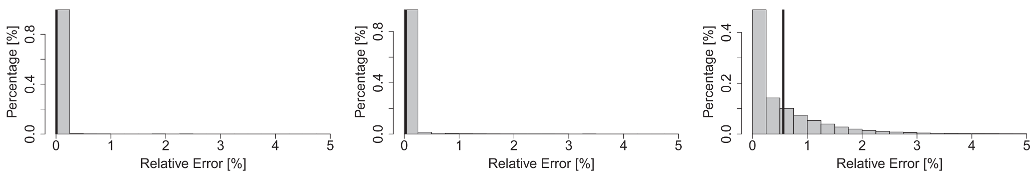
Table 1 shows verification results for the different numerical integration schemes aggregated across the car-following models. Since the Krauss' and Wiedemann's models assume one function evaluation per time step, the results for the trapezoid rule and RK4 exclude these two models. The results show that both the mean and maximum relative deviation depend strongly on the integration scheme. With the forward Euler scheme, the largest observed absolute deviations with $\tau = 0.1$s after 30s of driving was 14.4m for Krauss' model. Excluding the stochastic models, the largest deviation with the forward Euler scheme was 1.99m for Gipps' model. With RK4, the maximum absolute error is reduced immensely to only 1.35×10^{-6}m $= 1.35\mu$m for Gipps' model.

These results show that the fast-forward functions accurately represent the acceleration behavior specified in the car-following models. The remaining deviation is due to the inaccuracy introduced by the integration scheme used in the time-driven updates. Still, we consider the outcomes of time-driven simulations our reference results. Since the ballistic update scheme enables higher fidelity than forward Euler, while still relying on only one function evaluation per time step, our experiments on entire road networks rely on the ballistic update scheme.

4.1.2 Grid and Singapore Scenarios. For the grid and Singapore scenarios, we conduct the verification with respect to the trip duration, which is a commonly studied metric in transportation

Table 2. Average Trip Durations [s] in Time-driven and Fast-forwarding Runs

	Grid			**Singapore**		
Peak agent count	**500**	**2,000**	**10,000**	**4,800**	**12,600**	**24,700**
Time-driven	371.3	370.5	377.4	783.2	874.0	1003.0
Fast-forwarding	371.3	370.5	375.7	786.2	876.8	1006.1

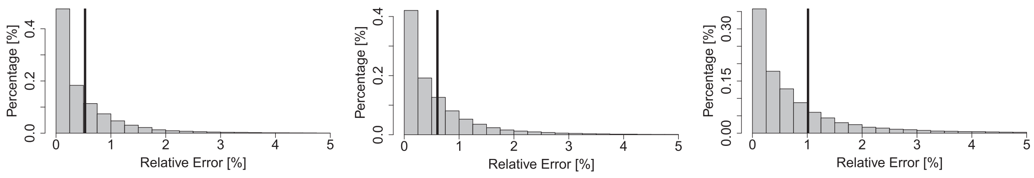


(a) Peak agent count: 500.
Avg.: 0.01%. 99.9%-quantile: 0.09%.

(b) Peak agent count: 2 000.
Avg.: 0.03%. 99%-quantile: 0.65%.

(c) Peak agent count: 10 000.
Avg.: 0.56%. 99%-quantile: 3.49%.

Fig. 6. Relative deviation between fast-forwarding and time-driven updates in the grid scenario.



(a) Peak agent count: 4 800.
Avg.: 0.53%. 99%-quantile: 3.55%.

(b) Peak agent count: 12 600.
Avg.: 0.60%. 99%-quantile: 3.84%.

(c) Peak agent count: 24 700.
Avg.: 1.01%. 99%-quantile: 8.23%.

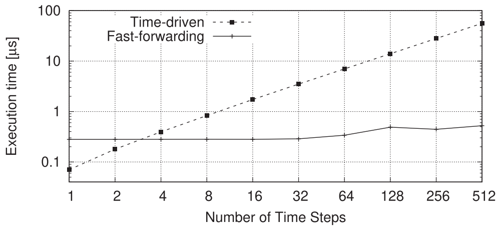
Fig. 7. Relative deviation between fast-forwarding and time-driven updates in the Singapore scenario.

engineering. Table 2 compares the average trip duration. Sub-link scanning was enabled during these experiments. For the grid scenario, we performed a parameter sweep across the scanning parameters. The verification results are given for the parameter combinations resulting in the lowest execution times. For the Singapore scenario, the parameters were configured at runtime using auto-tuning. The time step size was 0.1s. We observe that in both scenarios, the deviation in the average trip durations between the time-driven execution and fast-forwarding is less than 1%.

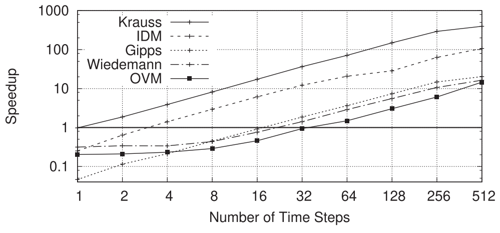
Figures 6 and 7 show histograms of the relative deviation introduced by fast-forwarding on a per-vehicle basis. We observe that the relative deviation increases with larger amounts of traffic. With larger traffic densities, it becomes more likely that a small difference in the progress of a vehicle is amplified through subsequent interactions with another vehicle. For instance, a slight delay in a vehicle's advancement to a new road may allow another vehicle to enter the road first, causing a braking maneuver and a substantial increase in the overall trip duration compared to the time-driven simulation. Among the considered scenarios, the maximum average deviation is about 1%. We also report the 99%-quantile, which is less than 10% in all scenarios.

4.2 Performance Measurements

4.2.1 Fast-forward Function. To understand the potential for performance gains using the proposed approach, we first compare the computational cost of iterative time-driven agent updates and updates using the fast-forward function for the Intelligent Driver Model. We simulate a single vehicle on a road segment of 10km length. Initially, the velocity is 0km/h. The vehicle accelerates to the speed limit of 100km/h. In Figure 8, we compare the wall-clock time required to execute a



(a) Wall-clock execution time of IDM.



(b) Speedup over purely time-driven execution.

Fig. 8. Execution time required for simulating a certain number of steps of 0.1 s using time-driven execution compared to fast-forwarding on a single road segment.

certain number of time steps to the time required to advance a vehicle by the same distance using fast-forwarding. Each measurement was repeated 10^7 times in the time-driven case and 100,000 times for fast-forwarding. The figure shows averages over the repetitions.

Figure 8(a) shows for the Intelligent Driver Model that, as expected, the execution time of the fast-forward function is roughly constant, whereas the execution time of time-driven updates depends approximately linearly on the number of steps. Although the fast-forward function is associated with higher computational cost than an individual time-driven state update, fast-forwarding outperforms time-driven updates beyond 3 consecutive steps. Thus, assuming no additional overheads, fast-forwarding is beneficial when the average number of skipped time steps is larger than 3.

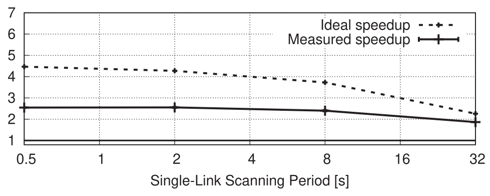
Reducing the time step size of the simulation allows for a larger number of skipped time steps per fast-forwarding operation. However, due to the overall increase in time steps, the proportion of skipped time steps across the simulation run will remain roughly the same.

Figure 8(b) shows the speedup of fast-forwarding by a given number of time steps over a time-driven execution for all of the considered car-following models. As expected, we observe near-linear speedup for all models. Some variation in the speedup is introduced by the numerical steps during fast-forwarding: as discussed in Section 3.4, we rely on Halley's method to iteratively solve for the vehicle's final velocity when a closed form is not available. The number of iterations required is dependent on the initial estimate and the shape of the acceleration function of the considered model. Given the speed limit $v_{\max}$, we used an initial estimate of $0.8v_{\max}$. In the experiments on entire road networks, we used the current velocity as the initial estimate.

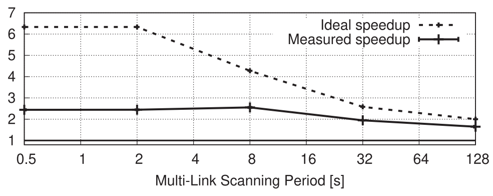
The break-even point between fast-forwarding and time-driven updates for Gipps' and Wiedemann's model as well as OVM lies between 16 and 32 steps, corresponding to 1.6s to 3.2s of model time. As described in Section 3.4, fast-forwarding of Krauss' model is possible using closed forms alone. Accordingly, the speedup is the largest for this model. In fact, for Krauss' model, the execution times of a single time-driven update and a fast-forwarding operation are roughly identical.

Note that the small-scale benchmark simulation using a single road on a single lane presented in this section may offer more opportunities for compiler optimizations than simulations on entire road networks, although the values of parameters such as the speed limit and the number of time steps were configured at runtime and could thus not be leveraged for optimizations. We repeated the measurements with the lowest optimization level (`-O0`) supported by the C++ compiler from the GNU compiler collection, observing the same trends as described above. In the next sections, we evaluate the performance gains in simulations of entire road networks.

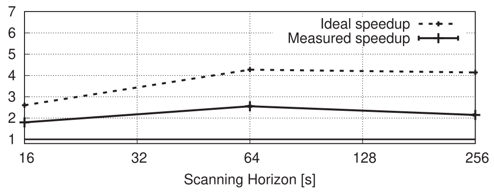
4.2.2 Grid Scenarios. Figures 9 to 11 show the overall speedup achieved using the fast-forwarding approach with sub-link scanning enabled compared with a time-driven execution for



(a) Multi-link period 8s, horizon 64s.

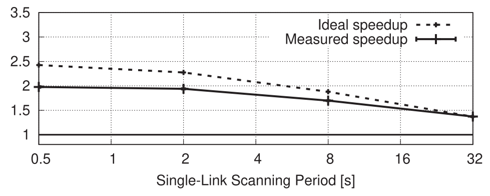


(b) Single-link period 2s, horizon 64s.

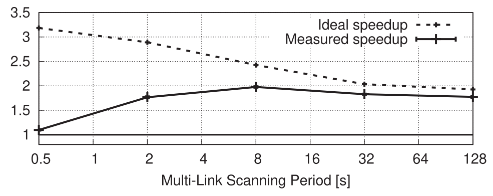


(c) Single-link period 2s, multi-link period 8s.

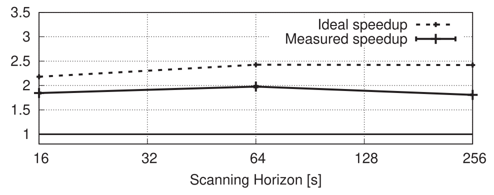
Fig. 9. Ideal and measured speedup with 500 agents in the grid scenario.



(a) Multi-link period 8s, horizon 64s.



(b) Single-link period 0.5s, horizon 64s.

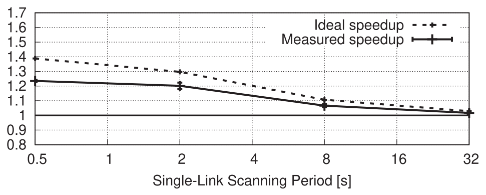


(c) Single-link period 0.5s, multi-link period 8s.

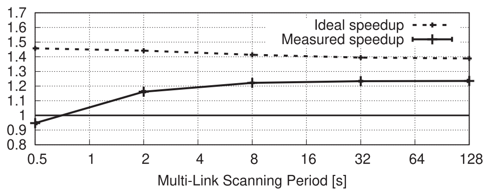
Fig. 10. Ideal and measured speedup with 2,000 agents in the grid scenario.

the grid scenario for 500, 2,000, and 10,000 agents, respectively. In addition, we plot the relative reduction in state updates, which indicates the ideal speedup through fast-forwarding when disregarding the overhead for identifying independence intervals and the evaluation of the fast-forward function. For instance, if the number of time steps is reduced by 50%, the ideal speedup is 2. We show three plots for each number of agents, each varying one of the parameter's single-link scanning period, multi-link scanning period, and scanning horizon, while keeping the other two parameters fixed at the values that achieved the largest speedup.

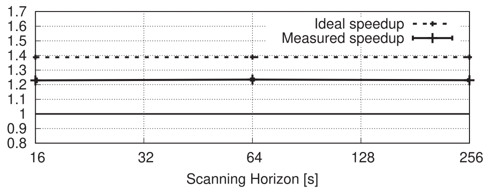
We can observe in Figure 9 that due to substantial opportunities for fast-forwarding with only 500 agents, frequent single-link, sub-link, and multi-link scanning as well as a large scanning horizon are beneficial. The best performance was achieved with single-link and multi-link scanning periods of 2s and 8s and a scanning horizon of 64s.



(a) Multi-link period 0.5s, horizon 64s.



(b) Single-link period 0.5s, horizon 64s.



(c) Single-link period 0.5s, multi-link period 128s.

Fig. 11. Ideal and measured speedup with 10,000 agents in the grid scenario.

Table 3. Performance in the Grid Scenarios with the Best-performing Scanning Parameter Combinations

Peak agent count	**500**		**2,000**		**10,000**	
Sub-link scanning	**off**	**on**	**off**	**on**	**off**	**on**
Execution time [s]	3.5	3.6	24.0	24.0	408.4	404.3
Fast-forwarding overhead [%]	12.1	12.2	10.8	11.3	3.4	3.7
Steps skipped [%]	76.4	76.6	58.0	58.8	25.8	28.0
Steps skipped per fast-forwarding	187.5	184.5	90.3	82.9	52.0	39.1
Speedup over time-driven	**2.56**	**2.55**	**1.98**	**1.98**	**1.22**	**1.24**

Figure 10 demonstrates the trade-off between the scanning overhead and the performance gains through fast-forwarding: While a small multi-link scanning period reveals substantial opportunities for fast-forwarding, the overall speedup increases with larger multi-link scanning periods.

In Figure 11, we can see that in the most congested scenario with 10,000 agents, only few opportunities for fast-forwarding exist. In this scenario, performance gains are mostly achieved through single-link scanning, so a large multi-link scanning period can be chosen.

Overall, the measurements show that the opportunities for fast-forwarding decrease when increasing the traffic density. Further, we can conclude that to limit the overhead for identifying independence intervals, it is important to choose appropriate values for the scanning parameters.

Table 3 shows the effects of fast-forwarding in detail, comparing the results of using only single-link and multi-link scanning to results with sub-link scanning enabled. The measured “fast-forwarding overhead” includes both the scanning and the fast-forwarding operations. As expected, we observe that the percentage of skipped time steps decreases with increasing traffic density. While the absolute execution time spent on scanning and fast-forwarding increases, the overhead relative to the overall execution time decreases. Sub-link scanning enables a minor increase in the percentage of skipped time steps. However, due to the fine-grained scanning, the individual skipping operations cover a smaller number of time steps each. In Figure 8, we have seen that the performance gain is higher when skipping larger number of time steps. Overall, in the grid scenario, sub-link scanning does not achieve a significant increase in speedup.

Table 4. Performance in the Singapore Scenario with Scanning Parameters Configured Using Auto-tuning

Peak agent count	**4,800**		**12,600**		**24,700**	
Sub-link scanning	**off**	**on**	**off**	**on**	**off**	**on**
Execution time [s]	110.6	108.7	659.0	651.3	1,650.0	1,643.1
Fast-forwarding overhead [%]	3.3	3.7	1.3	1.5	0.8	1.0
Steps skipped [%]	36.7	38.14	22.1	23.1	14.1	14.8
Steps skipped per fast-forwarding	115.5	100.4	101.1	86.8	94.8	81.5
Speedup over time-driven	**1.54**	**1.57**	**1.12**	**1.14**	**1.05**	**1.05**

4.2.3 Singapore Scenario. The performance results for the Singapore road network using parameter auto-tuning for the selection of the scanning parameters are shown in Table 4. The results follow a similar trend as those for the grid road network. Speedup factors of 1.57, 1.14, and 1.05 are achieved with sub-link scanning for peak traffic amounts of 4,800, 12,600, and 24,700 agents, respectively. The results show that even when discounting the fast-forwarding overhead, the percentage of skipped time steps cannot be translated to speedup directly: Since a time-driven update of an isolated agent does not need to consider any neighboring agents, it is less computationally intensive than the update of an agent on a densely populated road. Thus, avoiding updates for isolated agents contributes less to the speedup than suggested purely by the percentage of skipped steps.

5 DISCUSSION

In this section, we discuss limitations and potential enhancements of the proposed approach.

5.1 Applicability to More Complex Models and Other Domains

In the present article, we assume that the routes of the agents, i.e., the sequences of edges, will not change after the scanning has been performed. Depending on the considered simulation model, agents may change their routes due to interactions with other agents or even spontaneously. For instance, an agent may re-route when it detects a traffic jam. To support such models, we could either terminate the scanning process at the first point in time when a change is possible or determine occupancy intervals for all possible branches. Both approaches may substantially reduce the opportunities for fast-forwarding. If routing decisions are made stochastically, then pre-sampling from the pseudo-random number stream may still enable prediction of the agents' routes. In the extreme case of entirely unpredictable routes, fast-forwarding would be limited to disjoint areas reachable by agents according to their maximum velocity.

From the problem analysis in Section 3.1, we can infer that the applicability of our approach to other types of time-driven agent-based simulations depends on the specific models used. The approach is applicable to models that allow for the prediction of future agent states and to scenarios where independent agent updates occur. For instance, in crowd simulations using predefined routes on a two-dimensional simulation space, the path taken by isolated agents may be fully predictable. With complex models, it must be considered that the performance benefit of the approach depends on the computational cost of the fast-forward function relative to time-driven updates as well as on the costs for determining independence intervals.

5.2 Deviations Compared to Time-driven Execution

As discussed in Section 3 and evaluated in Section 4.1, state updates performed using the fast-forward function deviate slightly from those performed using iterative time steps. Although the deviations are low and it can be argued that the fast-forward function is closer to the behavior specified by the respective car-following model, two undesirable properties emerge: first, as with any state change in an agent-based simulation, deviations may affect other agents and may thus propagate through the road network. Second, the deviations depend on the parametrization of the fast-forwarding approach, i.e., on the frequency of scanning and on the scanning horizon. Thus, the approach interlinks the execution of the simulation and the observed behavior of the simulation. Ideally, to allow modelers to clearly identify cause-and-effect relationships when modifying the model or scenario, these two aspects should be decoupled. A possible solution is to carry out the model development and simulator debugging using a purely time-driven execution, whereas performance-critical and large-scale runs are accelerated using fast-forwarding.

5.3 Influence on Statistics Collection and Visualization

Typically, in an agent-based simulation, statistics are gathered by periodically aggregating the states of the agents in the simulation, e.g., the velocities of the vehicles on a road network. Usually, the period of aggregation is a multiple of the time step size. If some updates are performed in event-driven fashion according to the proposed approach, then agents may be fast-forwarded beyond the point when statistics are to be collected. A simple solution is to limit the scanning horizon to the next statistics gathering time. A similar problem emerges in the context of optimistically synchronized distributed simulations [43].

Visualization tools could apply interpolation to approximate agent states at time steps that have been skipped through fast-forwarding.

5.4 Further Opportunities for Fast-forwarding

One of the main limitations of the approach is the difficulty of identifying fast-forwarding opportunities in congested scenarios, which may limit the benefits of the approach in common traffic engineering scenarios with dense traffic. In road traffic simulations that consider traffic lights, trivial opportunities for fast-forwarding may be given for vehicles stopped at a traffic light. Such vehicles can simply be fast-forwarded to the next state change of the traffic lights. It may also be possible to extend fast-forwarding to clusters of two or more agents. While the prediction of lane changes for isolated agents is trivial, fast-forwarding of agent clusters will require the prediction of lane changes as a result of vehicle interactions. Further, the computational cost of solving the resulting coupled differential equations may limit the additional performance gains.

5.5 Controlling Overhead

A number of ways present themselves to control the scanning overhead: first, the scanning is parametrized with the scanning periods and scanning horizon. We showed that the optimal parameter values depend strongly on the scenario. Thus, we applied a simple auto-tuning scheme to adapt the scanning parameters according to the traffic conditions of the Singapore scenario.

Second, in addition to the variation of congestion across model time, congestion also typically shows variations across the simulated space. For instance, during peak hours a highly congested roadway will provide few opportunities for fast-forwarding, in contrast to sparsely populated roads in residential areas. In such situations, unnecessary computations could be avoided by restricting the scanning to areas outside congested areas. However, further considerations are then required to maintain correctness: Since vehicles may enter or exit congested areas within the considered scanning horizon, excluding agent interactions would require a safety margin around these areas, which could be defined based on static information such as speed limits.

Finally, the scanning operation could be offloaded to a separate processor and carried out concurrently with the simulation. Within the accuracy allowed by the time step size, previously identified occupancy intervals may be outpaced by the simulation's progress, but not invalidated. Thus, after scanning, fast-forwarding could be applied to all agents that have not yet progressed beyond the target time. Further, during scanning, the scanning function is evaluated a number of times for each relevant vehicle independently, providing ample opportunities for parallelization, e.g., on graphics processing units.

6 RELATED WORK

In this section, we first briefly discuss how hybrid modeling approaches relate to fast-forwarding. We subsequently give an overview of previous work focusing on identifying and exploiting independence between state updates for parallelization of discrete-event simulations and for accelerating sequential and parallel time-driven agent-based simulations.

6.1 Hybrid Traffic Simulation

In hybrid traffic simulation [4, 51], microscopic models are combined with mesoscopic or macroscopic models to balance simulation fidelity and performance. Spatial or temporal segments of the simulation are selected in which a reduction in detail and accuracy is acceptable. In these segments, vehicles are considered in aggregate, e.g., as sets of tasks in a queuing network or in terms of fluid dynamics. As a consequence, it is not always possible to study an individual vehicle across its entire route. In contrast, fast-forwarding does not consider agents in aggregate. Hence, each vehicle's trajectory can still be studied individually. Further, fast-forwarding is applied only if it is ensured that, within the accuracy allowed by the time step size, the simulation results are unaffected.

6.2 Exploiting Independent State Updates

Our proposed approach trivially parallelizes the scanning for independence intervals, whereas all other steps of the simulation are executed sequentially. However, periods in model time where interactions among certain segments of the simulation can be ruled out are commonly exploited in the field of parallel and distributed simulation [15]. To reduce the cost of synchronization between processing elements, methods have been proposed to exploit *lookahead*, i.e., the difference in model time between an event's creation and execution time [14]. If a lower bound on the lookahead can be determined [30, 36], then intervals in model time can be identified during which processing elements can compute independently. Some previous works consider the minimum model time required for a sequence of events to propagate to a remote processing element [7, 11, 31, 32, 35, 40, 47]. Similarly to our approach, intervals of independence are derived according to the topology of the modeled system. However, instead of exploiting the identified independence for parallel execution, in our work, we accelerate sequential simulations by performing independent agent state updates using a computationally inexpensive fast-forward function. As the lookahead-based approaches in parallel and distributed simulation, the fast-forwarding approach relies on a degree of predictability of the entities' behavior, which is model-dependent.

In optimistically synchronized parallel and distributed simulations [13, 19, 39, 44], some computations are performed speculatively and rolled back when a violation of the simulation correctness is detected. In our approach, the identification of occupancy intervals can be seen as speculative state updates under the assumption of independence among agents. When independence between the agent updates cannot be guaranteed, the results are discarded. Lees et al. studied the effects of access patterns to shared state variables on the performance of optimistic simulation algorithms [29].

Some previous works consider accelerating time-driven agent-based simulations by identifying independent state updates among agents: Scheutz et al. [21, 41, 42] apply a *translation function* that reflects the furthest possible amount of movement of an agent to determine an *event horizon* in model time. By identifying non-overlapping areas among multiple agents' event horizons, time intervals of mutually independent updates can be identified. Now, in the context of sequential agent-based simulations, agent updates can be prioritized to achieve the simulation's termination criterion with the minimum number of state updates. For instance, if the focus of the simulation study is on one particular agent, only the state updates directly or indirectly affecting this agent must be performed. In distributed agent-based simulations, idle times due to data dependencies can be reduced by prioritizing agent updates according to the data dependencies across processing elements. In contrast to our work, runtime reductions are achieved through changes in the ordering of agent updates, not through accelerating the state updates themselves. Since road traffic simulations are typically executed until all agents have reached a certain point in model time, the approach by Scheutz et al. would not accelerate such simulations.

Buss et al. [5] propose an event-driven modeling approach for scenarios involving movement and sensing. Instead of explicitly updating an entity's location over a sequence of time steps, events are scheduled at points in model time where changes in movement occur. However, determining suitable event scheduling times for sets of interacting vehicles may incur substantial overhead. Thus, in contrast to the purely event-driven approach proposed by Buss et al., our proposed fast-forwarding approach maintains a time-driven execution for all agents currently involved in an interaction. Further, while the work by Buss et al. and another work with a similar focus by Meyer [34] share with ours the general idea of avoiding explicit intermediate state changes, the main challenge lies in determining the points in model time when interactions between entities may occur and in determining the new agent state. In the present article, we address these challenges in the context of microscopic road traffic simulations.

Davidson and Wainer propose a language to formulate cell-based models of road traffic based on the Cell-DEVS formalism allowing for an event-driven execution [10]. In each cell representing a road segment, a user-defined delay function can be applied to model the time required to traverse the segment. Another event-driven traffic model formulation targeting optimistic parallel execution has been proposed by Yoginath and Perumalla [50]. The time to traverse an empty road segment is determined based on the assumption of a fixed acceleration. If there is a vehicle ahead, then the event indicating the departure from the road segment is delayed based on the time of departure of the vehicle ahead. In contrast to these existing works, our goal is to support existing and commonly used models for car-following behavior from the traffic simulation domain. Thus, simulation studies can benefit from the performance gains of fast-forwarding while relying on well-accepted models that have undergone extensive analysis and calibration efforts in the literature.

Less closely related to our approach is the concept of simulation cloning [24]. In this approach, the total execution time of a set of simulation runs is reduced by computing only the divergent state updates across multiple runs. For instance, if a single agent's state is modified across runs, state changes of other agents that are unaffected by this agent are not recomputed [38]. Similarly, in updatable and exact-differential simulation [12, 20], intermediate events of an initial full simulation run are stored. Subsequent simulation runs branch off from this initial run, reusing stored events unaffected by the branching. As in these approaches, fast-forwarding exploits independence between state updates to accelerate simulations. However, instead of avoiding recomputation, the fast-forwarding approach proposed in the present article avoids computation of some updates entirely.

Finally, the term "fast-forwarding" was used in other contexts where existing information is exploited to advance a simulated entity in model time. In updatable simulations [12], some repeated event executions can be avoided, thus "fast-forwarding" the corresponding simulated entity. Mauve et al. [33] use the term "fast forward" to describe the re-execution of events after a rollback in the context of optimistic synchronization for distributed virtual environments.

7 CONCLUSIONS AND OUTLOOK

We propose an approach to accelerate microscopic traffic simulation by identifying intervals of independent state updates and performing such independent updates using a computationally inexpensive fast-forward function. The approach maintains the microscopic nature of the simulation. We derived fast-forward functions for several well-known models of car-following behavior and evaluated the approach for a synthetic scenario and the road network of the city of Singapore. Our verification results show that the deviation from a purely time-driven execution is marginal. The performance benefit of the approach depends strongly on the level of agent density in the scenarios: for scenarios with sparse traffic, speedup factors of 2 and more were achieved, whereas with dense traffic, the reduced amount of opportunities for fast-forwarding allowed for only limited performance gains. We explored an additional fine-grained scanning scheme, which slightly increases the fast-forwarding opportunities in dense traffic. A possible direction for future work lies in exploring the joint fast-forwarding of clusters of vehicles. Further, the fast-forwarding approach could be extended to models with more complex agent movement behaviors such as crowd models.

REFERENCES

[1] Philipp Andelfinger, Yadong Xu, Wentong Cai, David Eckhoff, and Alois Knoll. 2018. Fast-forwarding agent states to accelerate microscopic traffic simulations. In *Proceedings of the Conference on Principles of Advanced Discrete Simulation*. ACM, New York, NY, 113–124.
[2] Masako Bando, Katsuya Hasebe, Akihiro Nakayama, Akihiro Shibata, and Yuki Sugiyama. 1995. Dynamical model of traffic congestion and numerical simulation. *Phys. Rev. E* 51, 2 (1995), 1035.
[3] Jaime Barceló and Jordi Casas. 2005. *Dynamic Network Simulation with AIMSUN*. Springer, Boston, MA, 57–98.
[4] Wilco Burghout, Haris Koutsopoulos, and Ingmar Andreasson. 2005. Hybrid mesoscopic-microscopic traffic simulation. *Transport. Res. Rec. J. Transport. Res. Board* 1934, 1 (2005), 218–255.
[5] Arnold H. Buss and Paul J. Sánchez. 2005. Simple movement and detection in discrete event simulation. In *Proceedings of the Winter Simulation Conference*. IEEE Press, Piscataway, NJ, 992–1000.
[6] Robert E. Chandler, Robert Herman, and Elliott W. Montroll. 1958. Traffic dynamics: Studies in car following. *Op. Res.* 6, 2 (1958), 165–184.
[7] Moo-Kyoung Chung and Chong-Min Kyung. 2006. Improving lookahead in parallel multiprocessor simulation using dynamic execution path prediction. In *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society, Washington, DC, 11–18.
[8] Biagio Ciuffo, Vincenzo Punzo, and Marcello Montanino. 2012. Thirty years of Gipps' car-following model: Applications, developments, and new features. *Transport. Res. Rec.* 2315, 1 (2012), 89–99.
[9] Robert M. Corless, David J. Jeffrey, and Donald E. Knuth. 1997. A sequence of series for the Lambert W function. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*. ACM, New York, NY, 197–204.
[10] A. Davidson and G. Wainer. 2000. Specifying truck movement in traffic models using cell-DEVS. In *Proceedings of the Simulation Symposium*. Society for Computer Simulation International, 66–73. DOI: https://doi.org/10.1109/SIMSYM.2000.844902
[11] Ewa Deelman, Rajive Bagrodia, Rizos Sakellariou, and Vikram Adve. 2001. Improving lookahead in parallel discrete event simulations of large-scale applications using compiler analysis. In *Proceedings of the Workshop on Parallel and Distributed Simulation*. IEEE Computer Society, Washington, DC, 5–13.
[12] Steve L. Ferenci, Richard M. Fujimoto, Mostafa H. Ammar, Kalyan Perumalla, and George F. Riley. 2002. Updateable simulation of communication networks. In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation*. IEEE Computer Society, Washington, DC, 107–114.
[13] Richard Fujimoto. 2015. Parallel and distributed simulation. In *Proceedings of the Winter Simulation Conference*. IEEE Press, Piscataway, NJ, 45–59.
[14] R. M. Fujimoto. 1988. Lookahead in parallel discrete event simulation. In *Proceedings of the International Conference on Parallel Processing, Vol. 3*. Pennsylvania State University Press, Philadelphia, PA, 34–41.
[15] Richard M. Fujimoto. 2000. *Parallel and Distributed Simulation Systems*. Wiley, New York, NY.

[16] Walter Gander. 1985. On Halley's iteration method. *Amer. Math. Month.* 92, 2 (1985), 131–134.

[17] Peter G. Gipps. 1981. A behavioural car-following model for computer simulation. *Transport. Res. Part B: Methodol.* 15, 2 (1981), 105–111.

[18] Peter G. Gipps. 1986. A model for the structure of lane-changing decisions. *Transport. Res. Part B: Methodol.* 20, 5 (1986), 403–414.

[19] B. Kaan Gorur, Kayhan Imre, Halit Oguztuzun, and Levent Yilmaz. 2016. Repast HPC with optimistic time management. In *Proceedings of the 24th High Performance Computing Symposium.* Society for Computer Simulation International.

[20] Masatoshi Hanai, Toyotaro Suzumura, Georgios Theodoropoulos, and Kalyan S. Perumalla. 2015. Exact-differential large-scale traffic simulation. In *Proceedings of the Conference on Principles of Advanced Discrete Simulation.* ACM, New York, NY, 271–280.

[21] Jack Harris and Matthias Scheutz. 2012. New advances in asynchronous agent-based scheduling. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications.* CSREA Press, Athena, Georgia, 1.

[22] Dirk Helbing and Benno Tilch. 1998. Generalized force model of traffic dynamics. *Phys. Rev. E* 58, 1 (1998), 133.

[23] Bryan Higgs, Montasir M. Abbas, and Alejandra Medina. 2011. Analysis of the Wiedemann car following model over different speeds using naturalistic data. In *Proceedings of the International Conference on Road Safety and Simulation.* 1–22.

[24] Maria Hybinette and Richard M. Fujimoto. 2001. Cloning parallel simulations. *ACM Trans. Model. Comput. Simul.* 11, 4 (2001), 378–407.

[25] Arne Kesting and Martin Treiber. 2013. *Traffic Flow Dynamics: Data, Models and Simulation.* Springer Berlin.

[26] Arne Kesting, Martin Treiber, and Dirk Helbing. 2007. General lane-changing model MOBIL for car-following models. *Transport. Res. Rec. J. Transport. Res. Board* 1999, 1 (2007), 86–94.

[27] Arne Kesting, Martin Treiber, and Dirk Helbing. 2008. Agents for traffic simulation. In *Multi-agent Systems Simulation and Applications*, Adelinde M. Uhrmacher and Danny Weyns (Eds.). CRC Press, Boca Raton, Florida, Chapter 11, 325–356.

[28] Stefan Krauß. 1998. *Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics.* Ph.D. Dissertation. Universität zu Köln.

[29] Michael Lees, Brian Logan, and Georgios Theodoropoulos. 2008. Using access patterns to analyze the performance of optimistic synchronization algorithms in simulations of MAS. *Simulation* 84, 10–11 (2008), 481–492.

[30] Y.-B. Lin and E. D. Lazowska. 1990. Exploiting lookahead in parallel simulation. *IEEE Trans. Parallel Distrib. Syst.* 1, 4 (1990), 457–469.

[31] Jason Liu and David M. Nicol. 2002. Lookahead revisited in wireless network simulations. In *Proceedings of the Workshop on Parallel and Distributed Simulation.* IEEE Computer Society, Washington, DC, 79–88.

[32] Boris D. Lubachevsky. 1989. Efficient distributed event-driven simulations of multiple-loop networks. *Commun. ACM* 32, 1 (1989), 111–123.

[33] Martin Mauve, Jürgen Vogel, Volker Hilt, and Wolfgang Effelsberg. 2004. Local-lag and timewarp: Providing consistency for replicated continuous applications. *IEEE Trans. Multim.* 6, 1 (2004), 47–57.

[34] Ruth Meyer. 2014. Event-driven multi-agent simulation. In *Proceedings of the International Workshop on Multi-agent Systems and Agent-based Simulation.* Springer, Boston, MA, 3–16.

[35] Richard A. Meyer and Rajive L. Bagrodia. 1999. Path lookahead: A data flow view of PDES models. In *Proceedings of the Workshop on Parallel and Distributed Simulation.* IEEE, Piscataway, NJ, 12–19.

[36] D. M. Nicol and J. H. Saltz. 1988. Dynamic remapping of parallel computations with varying resource demands. *IEEE Trans. Comput.* 37, 9 (1988), 1073–1087.

[37] J. Kent Peacock, Johnny W. Wong, and Eric G. Manning. 1979. Distributed simulation using a network of processors. *Comput. Netw. (1976)* 3, 1 (1979), 44–56.

[38] Philip Pecher, Michael Hunter, and Richard Fujimoto. 2015. Efficient execution of replicated transportation simulations with uncertain vehicle trajectories. *Proc. Comput. Sci.* 51 (2015), 2638–2647.

[39] Kalyan S. Perumalla, Mohammed M. Olama, and Srikanth B. Yoginath. 2016. Model-based dynamic control of speculative forays in parallel computation. *Electron. Notes Theoret. Comput. Sci.* 327 (2016), 93–107.

[40] Patrick Peschlow, Andreas Voss, and Peter Martini. 2009. Good news for parallel wireless network simulations. In *Proceedings of the International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems.* ACM, New York, NY, 134–142.

[41] Matthias Scheutz and Jack Harris. 2010. Adaptive scheduling algorithms for the dynamic distribution and parallel execution of spatial agent-based models. *Parallel Distrib. Computat. Intell.* 269 (2010), 207–233.

[42] Matthias Scheutz and Paul Schermerhorn. 2006. Adaptive algorithms for the dynamic distribution and parallel execution of agent-based models. *J. Parallel Distrib. Comput.* 66, 8 (2006), 1037–1051.

[43] Vinoth Suryanarayanan and Georgios Theodoropoulos. 2013. Synchronised range queries in distributed simulations of multiagent systems. *ACM Trans. Model. Comput. Simul.* 23, 4, Article 25 (Nov. 2013), 25 pages.

[44] Vinoth Suryanarayanan, Georgios Theodoropoulos, and Michael Lees. 2013. PDES-MAS: Distributed simulation of multi-agent systems. *Proc. Comput. Sci.* 18 (2013), 671–681.

[45] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. 2000. Congested traffic states in empirical observations and microscopic simulations. *Phys. Rev. E* 62, 2 (Feb. 2000), 1805–1824.

[46] Martin Treiber and Venkatesan Kanagaraj. 2015. Comparing numerical integration schemes for time-continuous car-following models. *Phys. A: Statist. Mech. Applic.* 419 (2015), 183–195.

[47] Jun Wang, Zhenjiang Dong, Sudhakar Yalamanchili, and George Riley. 2013. Optimizing parallel simulation of multicore systems using domain-specific knowledge. In *Proceedings of the Conference on Principles of Advanced Discrete Simulation.* ACM, New York, NY, 127–136.

[48] Rainer Wiedemann. 1974. *Simulation des Strassenverkehrsflusses.* Habilitation Thesis. University of Karlsruhe, Germany.

[49] Yadong Xu, Wentong Cai, Heiko Aydt, Michael Lees, and Daniel Zehe. 2017. Relaxing synchronization in parallel agent-based road traffic simulation. *ACM Trans. Model. Comput. Simul.—Spec. Issue PADS 2015* 27, 2 (2017), 14:1–24.

[50] Srikanth B. Yoginath and Kalyan S. Perumalla. 2009. Reversible discrete event formulation and optimistic parallel execution of vehicular traffic models. *Int. J. Simul. Proc. Modell.* 5, 2 (2009), 104–119.

[51] Daniel Zehe, David Grotzky, Heiko Aydt, Wentong Cai, and Alois Knoll. 2015. Traffic simulation performance optimization through multi-resolution modeling of road segments. In *Proceedings of the Conference on Principles of Advanced Discrete Simulation.* ACM, New York, NY, 281–288.

[52] Daniel Zehe, Suraj Nair, Alois Knoll, and David Eckhoff. 2017. Towards CityMoS: A coupled city-scale mobility simulation framework. In *Proceedings of the 5th GI/ITG KuVS Fachgespräch Inter-Vehicle Communication Conference.* FAU Erlangen-Nuremberg, Erlangen, Germany, 26–28.

Received December 2018; revised May 2019; accepted October 2019