From Effects to Causes: Reversible Simulation and Reverse Exploration of Microscopic Traffic Models

Philipp Andelfinger
TUMCREATE Ltd and
Nanyang Technological University
pandelfinger@ntu.edu.sg

Jordan Ivanchev
TUMCREATE Ltd
jordan.ivanchev@tum-create.edu.sg

David Eckhoff
TUMCREATE Ltd and
Technische Universität München
david.eckhoff@tum-create.edu.sg

Wentong Cai
Nanyang Technological University
aswtcai@ntu.edu.sg

Alois Knoll
Technische Universität München and
Nanyang Technological University
knoll@in.tum.de

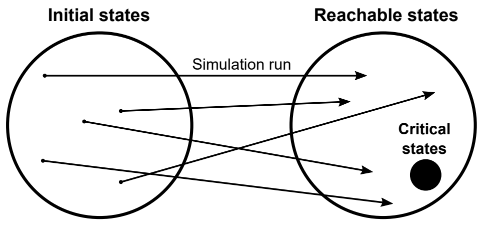
ABSTRACT

We propose an approach for reverse-in-time exploration of the state space of microscopic traffic simulations starting from a user-specified class of outcomes. As a basis for our approach, we present a reversible execution scheme applicable to common car-following and lane-changing models from the traffic simulation literature. The execution scheme permits perfect reversal of a previous forward simulation, which to our knowledge has not been attempted previously in the context of established traffic simulation models. Further, we perform reverse state space explorations directly from user-specified simulation states, i.e., reverse-in-time model checking. By exploring all sequences of possible previous states from a final state, reachability questions can be answered more conclusively than purely through forward simulations. In a case study, reverse exploration is used to identify conditions that lead to specified accident situations, with running time reductions by factors of more than 20 compared to traditional forward exploration.

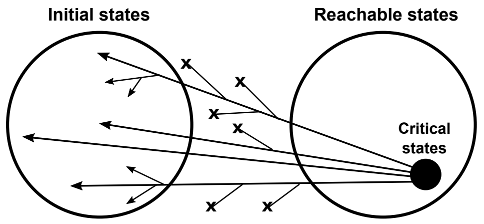
ACM Reference Format:
Philipp Andelfinger, Jordan Ivanchev, David Eckhoff, Wentong Cai, and Alois Knoll. 2019. From Effects to Causes: Reversible Simulation and Reverse Exploration of Microscopic Traffic Models. In *SIGSIM Principles of Advanced Discrete Simulation (SIGSIM-PADS '19), June 3–5, 2019, Chicago, IL, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3316480.3322891

1 INTRODUCTION

Microscopic traffic simulation, which represents vehicles as individual entities, is an established approach to evaluate scenarios in transportation planning and traffic engineering. Often, simulation studies take the form of a "what if" analysis: given the initial traffic conditions, a simulation run is executed. By repeating the simulation for different starting conditions, statistical results can be obtained. This approach is suitable when studying aggregated metrics such as vehicle flows at main roads or mean travel times.



(a) Forward simulations: initial conditions are sampled from the space of initial simulation states to identify state trajectories ending in a critical state.



(b) Reverse exploration: the reachable previous states are explored in reverse directly from the critical states. Unreachable states (signified by an x) are discarded.

Figure 1: Schematic view of a reachability analysis through forward simulations and reverse exploration.

If we are interested in the reachability of critical final simulation states, e.g., accidents or traffic jams, repeated simulation runs can be performed until a critical state has been reached or a sufficient level of confidence is achieved that such a state is unreachable [37, 42]. However, given the vast input state space of most simulations, this approach will rarely suffice to gather definite reachability results.

In the present paper, we propose an approach to enable reverse-in-time state space explorations of microscopic traffic simulations. From a user-specified simulation state, all possible previous simulation states are explored to conclusively answer reachability questions and to determine initial states leading to the configured outcome (cf. Figure 1). Thus, the approach enables a reverse model checking of microscopic traffic simulation models. As a basis for the approach, we first present an execution scheme that enables perfect reverse execution of forward simulation runs. While existing works have proposed reversible simulation models for traffic simulation [44] with the aim of supporting optimistic parallel execution, to our knowledge we are the first to present a reversible simulation scheme applicable to microscopic car-following and lane-changing models that are well-accepted and commonly used in the traffic engineering literature.

Our main contributions are as follows:

- We propose an approach for perfect reverse execution of microscopic traffic simulations.
- We propose a reverse-in-time exploration scheme that enables state space exploration of traffic models.
- We evaluate the memory consumption and performance of the proposed approaches and present a case study of the reverse exploration.

Our prototypical implementation is publicly available[1].

The remainder of the paper is structured as follows: In Section 2, we outline the considered traffic models as well as concepts fundamental to our approach. Sections 3 and 4 describes our proposed approaches for reversible simulation and reverse exploration. In Section 5, we evaluate the performance and scalability of our prototypical implementation. In Section 6, we apply the reverse exploration scheme to study the reachability of critical traffic situations. In Section 7, we discuss limitations and future work. In Section 8, we relate our work to existing approaches from the literature. Section 9 summarizes our results and concludes the paper.

2 FUNDAMENTALS

In the following, we first introduce the microscopic traffic simulation models used to demonstrate our approach and discuss the use of microscopic traffic simulation in the context of safety studies. Further, we briefly outline the concepts of reversible computing.

2.1 Microscopic traffic simulation

In microscopic traffic simulation, road traffic is modeled on the level of individual vehicles. Typically, two models are applied at each time step of size τ based on the state of the currently examined vehicle (ego vehicle) and neighboring vehicles: a car-following model updates the vehicle's velocity, and a lane-changing model determines whether the vehicle changes to one of the adjacent lanes. The car-following model considered in the present paper is the Intelligent Driver Model (IDM) [34]. The model is specified as a differential equation in continuous time. In practice, numerical integration is applied to solve the differential equation over discretized time. We describe the model with respect to the forward Euler scheme for integration. The IDM takes the following form:

$$\dot{v}[n] = a\left(1 - \left(\frac{v[n]}{v_0}\right)^4 - \left(\frac{s_0 + v[n]T + \frac{v[n](v[n]-v_{\text{leader}}[n])}{2\sqrt{ab}}}{p_{\text{leader}}[n] - p[n]}\right)^2\right)$$

where a, b, s_0,v_0, T are model parameters. p_{leader} and v_{leader} are the position and velocity of the vehicle ahead (leader). Using $\dot{v}[n]$, a state update from time step n to $n+1$ is carried out using $v[n+1] = v[n] + \dot{v}[n]\tau$ and $p[n+1] = p[n] + v[n+1]\tau$.

[1]https://doi.org/10.5281/zenodo.2591930
https://github.com/pandelfinger/ReversibleTrafficSim

The MOBIL lane-changing model [15] used in the present paper places the vehicle in the hypothetical situation of already being on the candidate lane and evaluates the utility u of this situation. It also considers the utilities experienced by other vehicles on the road, weighted by a politeness factor $\pi \in \mathbb{R} : \pi \in [0, 1]$. The utility of the ego vehicle being on an adjacent lane is defined as: $u = \tilde{\dot{v}} - \dot{v} + \pi\left(\tilde{\dot{v}}_h^f - \dot{v}_h^f + \tilde{\dot{v}}_c^f - \dot{v}_c^f\right) - \Delta a_{\text{th}}$, where Δa_{th} is the utility improvement threshold that must be exceeded for a vehicle to perform a lane change, $\dot{v}_c^f$ and $\dot{v}_h^f$ are the accelerations of the current and hypothetical follower, and a tilde represents the acceleration if the ego vehicle were to be placed on the respective adjacent lane. The accelerations of the vehicles are computed according to the IDM. The utilities of the possible lane changes are compared and the option with the highest positive utility is chosen.

2.2 Traffic simulation for safety analysis

Traffic simulations can be performed on various levels of modeling detail. The most detailed simulations include sophisticated models of aspects such as vehicle physics and sensor inputs, whereas less detailed simulations consider vehicles in aggregate in the form of queuing networks or fluid equations. Multi-fidelity safety assessment frameworks make use of models on a spectrum of detail levels [31]. Since microscopic simulation still considers vehicles individually, it can be considered the highest level of abstraction still meaningful for safety analysis. The approach for reverse-in-time state space exploration of the present paper targets a safety analysis using microscopic traffic simulation. While on the considered level of detail, the behavior generated by the car-following and lane-changing models cannot reflect complex real-world conditions, we argue that the model detail is sufficient to study high-level strategies for lane-changing, distance-keeping, and platooning.

2.3 Reversible computing

Reversible computing is concerned with the efficient reversed execution of computational tasks, motivated by use cases with benefits in energy efficiency, failure tolerance, and performance. An introduction to the field is given in [24]. While for some operations, such as the addition of a constant value to a variable, a unique reverse operation can be found, many operations such as XOR are inherently destructive: if the underlying function is not injective, multiple elements of the domain map to the same value of the codomain, so that a unique reversal must rely on additional information for disambiguation. A common solution from the literature is to *embed* the original function in a new function with an extended domain. So-called *garbage bits* hold the information required to disambiguate among the domain elements. The problem of efficiently determining minimal embeddings is subject of active research (e.g., [47]), typically targeting functions represented as logic tables. Thus, a constant number of output bits is commonly assumed. In the present paper, we define an embedding for the function determining the velocity of vehicles. A variable number of outputs bits is used to store the minimum number of bits required for disambiguation of each function invocation.

3 REVERSIBLE SIMULATION SCHEME

In the following, we propose an execution scheme supporting perfect reversal of microscopic traffic simulations. Our main ideas can be summarized as follows:

(1) **Number representation:** Due to the numerical issues arising from a floating point representation of velocities and positions, we quantize the velocity function using a fixed-point number representation.
(2) **Transition tables:** From the quantized velocity function, we create an embedding that supports perfect reversal. Forward and backward *transition tables* define mappings that enable vehicle state updates forward and backward in time. The table size is independent of the number of vehicles and is constant over the course of the simulation. The reliance on transition tables allows us to support arbitrary car-following models in a generic manner. For disambiguation of transitions, vehicle updates append zero or more garbage bits to a last in, first out data structure. In Section 4, the transition tables will form the basis of our proposed reverse state space exploration scheme.
(3) **Encoding of lane changes:** A simple encoding scheme is presented to compactly store lane-changing decisions for reverse execution.

3.1 Number representation

To achieve a reversible simulation, it must be possible to determine from the state of the simulation at time $n+1$ the state at time step n. It is known that a floating point number representation poses challenges to reversibility [24]. In the following, we show that in the considered problem, a representation of positions and velocities as IEEE 754 floating point numbers does not permit a perfect reversal of vehicle state updates. We assume that during forward simulation, the position change of a vehicle from time step n to $n+1$ depends only on the velocity $v[n+1]$. This assumption is satisfied by the forward Euler scheme: $p[n+1] = p[n] + v[n+1]\tau$. Suppose that during a reverse simulation $p[n+1]$ and $v[n+1]$ are known. Now, we should be able to simply compute $p[n] = p[n+1] - v[n+1]\tau$. However, since floating point arithmetic is not associative [10], it is possible that $(a+b)+c \neq a+(b+c)$. By setting $a = p[n]$, $b = v[n+1]\tau$, and $c = -v[n+1]\tau$, we see that the following inequality may be satisfied:

$$(p[n] + v[n+1]\tau) - v[n+1]\tau \neq p[n] + (v[n+1]\tau - v[n+1]\tau)$$

which simplifies to $p[n+1] - v[n+1]\tau \neq p[n]$. Thus, even if the velocity is known with perfect precision, the previous position cannot always be recovered exactly.

To still achieve perfect reversibility, we rely on a fixed point number representation. A fixed spacing between two adjacent numbers across the representable range enables associative arithmetic operations. Our implementation relies on the fixed point arithmetic library libfixmath[2]. Numbers are represented by 32 bits, with 16 bits each for the integer and decimal value, allowing us to represent numbers between -32,768 and +32,768 with 65,536 decimal levels.

[2]https://code.google.com/archive/p/libfixmath/

Table 1: Forward transition table with example entries. Disambiguation through a garbage value g is needed for transitions with identical $v_{\text{leader}}[n]$, $\Delta p[n]$, and $v[n+1]$.

$v[n]$	$v_{\text{leader}}[n]$	$\Delta p[n]$	$v[n+1]$	g
0	0	0	0	0
5	0	0	0	1
10	0	0	0	2
0	5	0	0	0
⋮	⋮	⋮	⋮	⋮

Table 2: Backward transition table with entries corresponding to the forward table above.

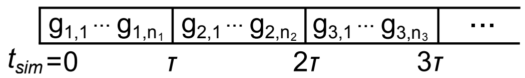
$v[n]$	$v_{\text{leader}}[n-1]$	$\Delta p[n-1]$	g	$v[n-1]$
0	0	0	0	0
0	0	0	1	5
0	0	0	2	10
0	5	0	0	0
⋮	⋮	⋮	⋮	⋮

3.2 Transition tables

Given the position $p[n+1]$ and velocity of a vehicle $v[n+1]$ at time step $n+1$ on a single-lane road, we are able to compute the previous position $p[n]$. The main challenge of the reverse simulation is to compute the previous velocity $v[n]$. The velocity function f of a car-following model relates a vehicle's new velocity to its old velocity, its distance Δp to a leading vehicle, and the leading vehicle's velocity: $v[n+1] = f(v[n], v_{\text{leader}}[n], \Delta p[n])$. A natural approach to the reversal is to use this relationship to determine a function r with $v[n] = r(v[n+1], v_{\text{leader}}[n], \Delta p[n])$.

Using the function r, we could reverse the transitions of vehicles on a lane step by step starting from the first vehicle on the road, for which we would set $v_{\text{leader}}[n] = 0$ and $\Delta p[n] = \infty$. However, since velocities may be reachable from multiple situations, it may not be possible to determine a function r of the form above. For instance, the velocity function f of the IDM car-following model in its most common parametrization is a fourth degree polynomial. Thus, there may be up to four possible solutions to the equation $f(v[n], v_{\text{leader}}[n], \Delta p[n]) - v[n+1] = 0$, allowing for up to four possible previous velocities. We carried out numerical experiments and observed up to two possible previous velocities. While in some cases, only one previous velocity was plausible, e.g., non-negative, model knowledge would be required to formulate criteria for selecting the correct previous velocity. To construct a general approach to reversible execution of car-following transitions, we extend the definition of f and r by a variable number of garbage bits per argument combination. A forward transition now yields a tuple $(v[n+1], g[n+1])$ comprised of the new velocity and a variable number of bits for disambiguation. A backward transition is executed using $v[n] = r(v[n+1], v_{\text{leader}}[n], \Delta p[n], g[n+1])$.

Of course, it would be possible to rely purely on the garbage bits to encode the chosen forward transition. However, since our aim is to minimize the number of garbage bits per step, we rely on all available state information at $[n+1]$ to minimize ambiguity. Thus,



$g_{1,1} \cdots g_{1,n_1}$
$g_{2,1} \cdots g_{2,n_2}$
$g_{3,1} \cdots g_{3,n_3}$
$\cdots$
$t_{sim}=0$
τ
2τ
3τ

Figure 2: Encoding of garbage bits for a single car-following event. Each forward transitions appends zero or more bits for disambiguation.

instead of encoding the entire transition, the garbage bits serve only to disambiguate among those forward transitions leading to the same combination of $v[n+1]$, $v_{\text{leader}}[n]$, and $\Delta p[n]$.

During the forward and reverse simulation, we represent f and r as lookup tables from which transitions are chosen according to the vehicle states at each time step. Tables 1 and 2 show the forward and reverse transitions tables. We separately store the number of ambiguous transitions for each combination of $v[n+1]$, $v_{\text{leader}}[n]$, and $\Delta p[n]$. The size of the transition tables is independent of the number of vehicles and remains constant over the course of a simulation run. It does, however, depend on the chosen level of granularity in the representation of velocities and positions. In Section 5, we evaluate this relationship and the effect of the granularity on the ambiguity of forward transitions.

Figure 2 illustrates the storage of garbage bits for car-following: each vehicle holds a dynamically sized last-in, first-out structure of garbage bits. During forward simulation, zero or more bits are appended according to the number of ambiguous transitions. Given m ambiguous transitions at time step i, we store $n_i = \lceil \log_2(m) \rceil$ bits. At each step of the reverse simulation, the corresponding number of bits is read to determine the correct backward transition.

3.3 Encoding of lane changes

A trivial scheme to enable the reversal of lane-changing decisions would rely on 2 bits per time step to encode three possible values representing a change to the left or right lane, or no change. We instead apply a simple run-length encoding scheme similar to Golomb coding [11]. Lane changes are stored as events comprised of a time step and a 1-bit value representing a change to the left or right lane. To reduce the number of bits required to represent the time step of the lane change, we divide the simulation time into epochs of configurable length l_e. It is then sufficient to store the time step within the current epoch instead of a global time stamp. The start of an epoch is indicated by a zero-bit. To differentiate between epoch markers and lane-changing events, the garbage bits associated with a lane change are succeeded by a one-bit. Figure 3 illustrates the encoding scheme. The encoding requires 1 bit each for the lane-change marker and lane-changing decision, and $b_e = \lceil \log_2(l_e) \rceil$ bits for the time step within the current epoch. The expected number of garbage bits per time step and vehicle required to encode lane changes in this fashion is given by $g_{\text{lc}}(b_e) = p_{\text{lc}}(2 + b_e) + \frac{1}{l_e}$. Here, p_{lc} is the probability of a lane change per time step. Assuming p_{lc} is known, we can determine the root r of the derivative of g_{lc} to determine the epoch length that minimizes the number of garbage bits: $r = \frac{\log 2}{p_{\text{lc}}}$. To obtain an integer number of bits, we set $b_e = \arg\min_{b \in \{\lfloor r \rfloor, \lceil r \rceil\}} g_{\text{lc}}(b)$.

In Figure 4, we plot g_{lc} for different lane-changing probabilities p_{lc}, comparing the optimal epoch length to a naive encoding using 2 bits per time step and fixed epoch lengths of 4, 16, and 64 steps.

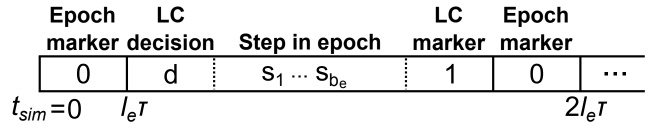


Figure 3: Encoding of garbage bits for lane-changing. Epoch markers are used to indicate that the next epoch has been reached. Within each epoch, b_e bits are used to store the time step of a lane change.

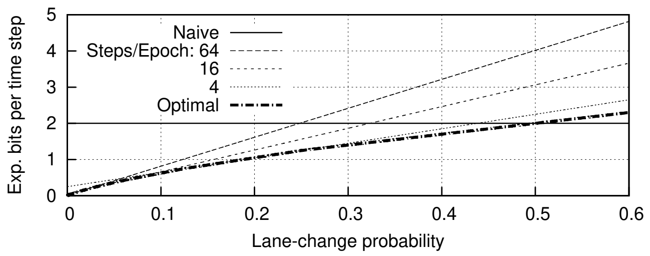


Figure 4: Expected number of garbage bits per time step and vehicle depending on the lane-changing probability p_{lc}.

We can see that the epoch length has a substantial impact on the expected number of garbage bits. A break-even point with the naive encoding is reached at a lane-changing probability of 0.5: if a lane change occurs at more than half of the time steps, it is more efficient to simply store the lane-changing decision at each time step.

4 REVERSE STATE SPACE EXPLORATION

The goal of the reversible simulation scheme described above is to execute a previous forward simulation in reverse. While the challenge of reversibility is the ambiguity in the forward transitions, we can instead exploit the ambiguity at each backward simulation step to answer reachability questions. In a reverse exploration, instead of a single "correct" backward path through the state space of the simulation, all valid paths are explored. Moreover, a forward simulation is not required: we can manually define the vehicle positions and velocities in a "final" simulation state and perform a reverse exploration from this state. The reverse exploration is akin to reverse-in-time model checking, enabling us to determine whether certain simulation states are reachable, and what initial states can lead to a given final state. Case studies demonstrating the use of the reverse exploration are described in Section 6.

Algorithm 1 shows the reverse exploration process with respect to car-following: similarly to the reversible simulation, each step of reverse exploration is performed front-to-back starting with the first vehicle on the considered lane. Instead of following the path through the state space of a forward simulation, we explore all possible transitions. Since each previous state of a leader vehicle may lead to multiple previous states of its successor, reverse exploration is a recursive procedure. It may also occur that one of the previous states of a vehicle does not lead to any possible previous states of its successor, i.e., there is no situation that can lead to the given combination of positions and velocities. Every time a valid state has

Algorithm 1 Reverse exploration of one car-following step.

```
procedure ReverseExplore(currState, prevState, vehIndex)
    veh ← currState[vehIndex]
    vCurr ← veh.v
    veh.p ← veh.p - vCurr × $\tau$
    if vehIndex = 0 then
        vAhead ← 0
        pAhead ← ∞
    else
        vehAhead ← currState[vehIndex - 1]
        vAhead ← vehAhead.v
        pAhead ← vehAhead.p
    k ← GetNumTransitions(veh.v, vAhead, pAhead - veh.p)
    for s ∈ {0, 1, …, k - 1} do
        veh.v ← GetVPrev(vCurr, vAhead, pAhead - veh.p, s)
        prevState.push(veh)
        if vehIndex = numVehicles - 1 then
            StoreValidState(prevState)
        else
            ReverseExplore(currState, prevState, vehIndex + 1)
        prevState.pop()
```

been found for the most behind vehicle on the lane, a valid overall state prior to car-following has been determined.

Note that to limit our description of Algorithm 1 to one time step of reverse exploration, we indicate that all valid previous states are stored. However, to reduce the memory consumption, in our implementation of the approach, we perform a depth-first search in time, i.e., for each valid state at time step n we first explore the states at $n - 1$ before considering the next valid state at time step n.

For each valid state before applying the car-following model, we also determine all possible states before lane-changing. Given k vehicles and the three possible lane-changing decisions of "left", "right", and "no change", there are a total of 3^k possible previous states. Of these, many states are not reachable because the vehicle is already on one of the outer lanes of the road. Other previous states may be unreachable through car-following. Still, the large number of previous lane positions results in a state explosion when exploring backward across multiple time steps. As a heuristic, it is possible to explore the previous states in ascending order of the number of lane changes: since lane changes are only performed when a lane change will result in a better driving situation, it is likely that only a few of the vehicles in a scenario perform a lane change at a given time step. Thus, the exploration can first assume no lane changes, then visit all possible states with one lane change, and so forth. For each number l of lane changes, $3^k \times \binom{k}{l}$ possible previous states are visited. While this heuristic ordering facilitates the identification of valid previous states, exhaustive explorations such as the ones presented in Sections 5 and 6 must explore all possible previous states. In Section 5.3, we evaluate the number of states visited in an example scenario.

5 EVALUATION

Our performance evaluation covers three aspects: firstly, we study the size of the transition lookup tables depending on the number granularity. Secondly, we explore the memory consumption and running time of the forward and backward simulation. Thirdly, we measure the performance of the reverse exploration to give an indication of tractable scenario conditions and search depths. The experiments were executed on a workstation equipped with an Intel Core i5-7400 and 16GiB of RAM running Ubuntu 16.04.5 LTS.

Table 3: Number of argument combinations, unique output velocities, and table sizes depending on the granularity.

Granularity [m], [m/s]	#Forward keys	#Unique backward keys	Total table size
0.25	1,062,982	759,178	57 MiB
0.125	8,346,562	5,962,812	446 MiB
0.0625	66,152,322	47,262,261	3,533 MiB

5.1 Size of transition lookup tables

In the present paper, vehicles exhibit car-following behavior according to the Intelligent Driver Model. The model determines a vehicle's new velocity from its current velocity, position, as well as the distance to the leader and its velocity. The number of possible combinations of these model arguments depends on the minimum number increment (*granularity*) permitted by the number representation. Given a granularity of i and a sensing range r, there are $(\frac{r}{i} + 2)$ possible vehicle distances $\Delta p \in \{0, i, 2i, \ldots, \frac{r}{i}\} \cup \{\infty\}$. With the maximum velocity $v_{\max}$, the overall number of argument combinations is $(\frac{v_{\max}}{i} + 1)^2 \times (\frac{r}{i} + 2)$. Table 3 shows the number of keys of the transitions tables for granularities of 0.25, 0.125, and 0.0625. For the backward transition table, we only count unique combinations of $v[n+1]$, $v_{\text{leader}}[n]$, and $\Delta p[n]$. In all cases, the number of backward keys is about 71.4% of the number of forward keys. If the disambiguation indices are also considered, the number of keys of the forward and backward tables is identical.

We also list the total memory required by the forward and backward transitions tables. The values given are based on the number of argument combinations and the size of the table entries given our fixed-point number representation. A key of the forward transition table is comprised of the velocity of the ego vehicle, the distance to the leader, and the velocity of the leader. A forward table value is comprised of the new velocity and a disambiguation index. Given that we currently rely on 32 bit variables, a key-value pair of the forward transition table requires $5 \times 4 = 20$ bytes of memory. A key and value of the backward transition table require 16 and 4 bytes, respectively. In total, given w forward transitions, each table requires $w \times 20$ bytes of memory. To be able to determine the number of garbage bits to be read at a given backward transition, an additional table stores the number of forward transitions for each key of the backward transition table. This table requires $w \times 16$ bytes of memory.

In practice, the data structures used and the memory allocation scheme of the operating system increase the memory consumption further. In our current implementation, the measured total memory consumption after loading the tables was about 6.7 GiB at a granularity of 0.0625. The table sizes could be reduced substantially by using the minimum possible number of bits to represent table entries. For instance, if there are q unique parameter combinations, a key of the table could be represented by $\lceil \log_2(q) \rceil$ bits. For instance, only 26 bits would be required at a granularity of 0.0625

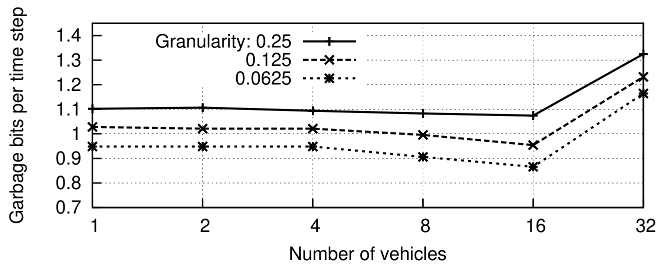


Figure 5: Number of garbage bits per time step and vehicle depending on the number of vehicles in the scenario.

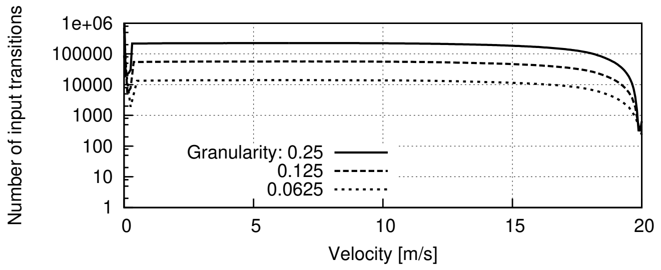


Figure 6: Number of input transitions of the Intelligent Driver Model leading to a given velocity at the next time step.

instead of the 128 bits used in our current implementation. We leave optimization efforts to reduce the memory consumption of the tables to future work.

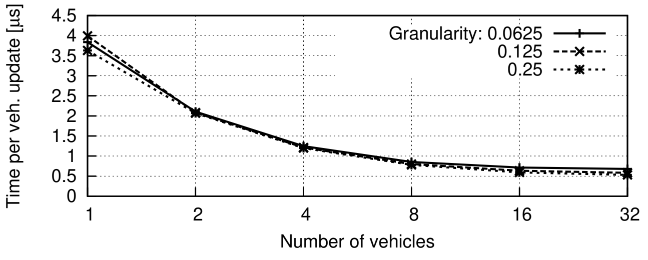
The total table size increases linearly with the number of parameter combinations, i.e., by a factor of $2^3 = 8$ when the granularity is halved. The observed memory consumption at a granularity of 0.0625 is within the capacities of commodity workstations.

5.2 Reversible simulation

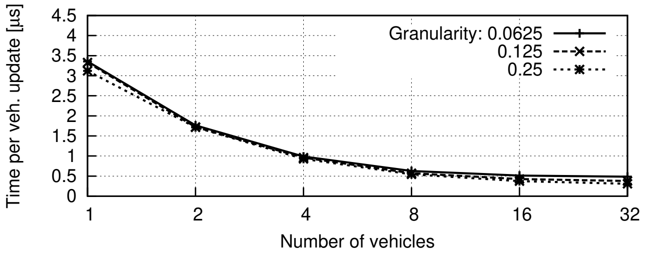
We simulated a scenario with three lanes and vehicle numbers in $\{1, 2, 4, 8, 16, 32\}$, each vehicle being 4.5m in length. Initially, the vehicles are placed on a random lane, and at positions chosen uniformly at random on the first 500m of the road. Each vehicle has a random initial velocity $\leq 20\frac{\text{m}}{\text{s}}$. To increase the probability of lane changes, 100 stationary vehicles obstructing one random lane each are generated with random spacing drawn from [10, 50]m. We repeated the experiment using granularities for velocities and positions of 0.25, 0.125, and 0.0625. For each parameter combination, 10,000 simulations were executed, each for 1,000 time steps of 0.1s. In the encoding of the garbage bits for lane-changing, we configured an epoch length of 4 time steps.

Figure 5 shows the number of garbage bits required per time step and vehicle to enable a reverse simulation. Between around 0.87 and 1.32 bits are needed, with the largest number of bits at 32 vehicles. The majority of the required garbage bits are generated by the car-following model, which for the different parameter combinations requires an average of 0.51 to 0.97 bits per time step and vehicle. We also observe that the granularity of positions and velocities has only a minor effect on the number of garbage bits, with finer granularities requiring fewer bits. A possible reason for the increase in garbage bits with 32 vehicles is a lower average velocity: Figure 6 shows the number of unique state transitions leading to a given velocity, covering all possible previous velocities as well as velocity and position differences. We observe that the number of transitions decreases with higher velocities, reducing the number of required garbage bits.

We now compare the number of garbage bits required in our reversible simulation with the memory requirements of a traditional periodic state saving approach. Given the limited number granularity in our approach, an equivalent representation of velocities up to and including $20\frac{\text{m}}{\text{s}}$ in a state saving approach at a granularity of 0.0625 would require $\lceil \log_2(\frac{20}{0.0625} + 1) \rceil = 9$ bits. According to our scenario parameters, we assume that there are three lanes and that



(a) Forward simulation.



(b) Backward simulation.

Figure 7: Wall-clock time required for a state update of one vehicle in forward and backward simulations.

positions take on values $\leq$ 2500m. Then, a full vehicle state can be represented by $9 + 2 + \lceil \log_2(\frac{2500}{0.0625} + 1) \rceil = 27$ bits. We can now determine the number of time steps of the reversible simulation at which the overall number of garbage bits equals the bits required by one full state saving. Given the 0.51 to 0.97 garbage bits per step of the reversible simulation, an equivalent memory consumption would be reached by saving the full vehicle states after a period of 28 to 53 time steps.

To support a finer granularity, the state saving approach would require additional bits. However, we have seen in Figure 5 that in the reversible simulation, the number of garbage bits per time step in fact decreases at finer granularities. Given this observation, the main concern when aiming for finer granularities is the total size of the lookup tables, which increases eightfold with each doubling (cf. Table 3), but remains constant across the simulation.

Although our current implementation of the reversible simulation has not been heavily optimized for performance, we also measured the running time of the forward and backward execution.

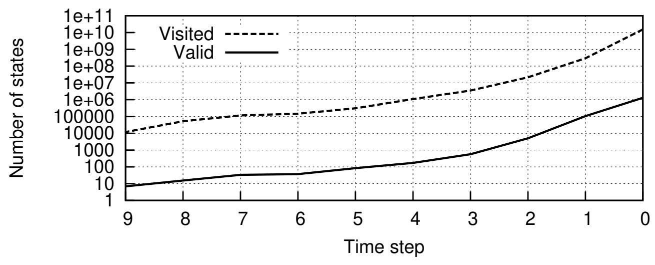


Figure 8: The visited and valid simulation states during reverse exploration of up to 10 time steps of $\tau = 0.1$s each in a scenario with 8 vehicles.

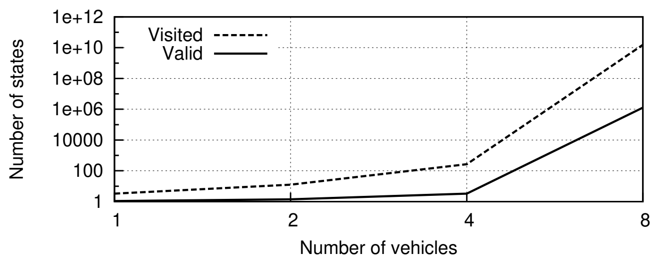


Figure 9: The average visited and valid simulation states after 10 time steps of reverse exploration depending on the number of vehicles in the scenario.

Figure 7 shows the wall-clock running time per time step and vehicle. Since there is a base overhead per simulation step, the running time per vehicle update decreases with the number of vehicles. With 32 vehicles, the absolute time required for one update in the forward simulation was between 0.53 and 0.68 µs. The backward simulation required between 0.31 and 0.48 µs per time step and is thus substantially faster than the forward simulation. We profiled the simulation using Google's gperftools[3] and found that the main cost of the forward simulation is incurred by the lane-changing model. In addition to collecting and evaluating the states of the ego vehicle's neighbors on the current and adjacent lanes, the MOBIL lane-changing model requires up to nine invocations of the car-following model per time step to determine the current and projected accelerations of the current and neighboring vehicles. In contrast, the backward simulation reverts the lane changes simply according to the values of the garbage bits. At the considered small numbers of vehicles, there is only a slight effect of the granularity on the running time.

5.3 Reverse state space exploration

We carried out reverse explorations of scenarios with vehicles placed randomly on the first 250m of a three-lane road without stationary obstacles. From these starting conditions, we simulated forward for 10 time steps before attempting a reverse exploration by 10 time steps. We repeated the experiment for vehicle counts in $\{1, 2, 4, 8\}$, each time starting from 10 scenarios generated with a different random number seed. The granularity was set to 0.0625.

We stress that the forward simulations are only carried out in this performance experiment to guarantee that the reverse exploration can proceed backwards by at least 10 time steps. In actual reachability studies such as the one described in Section 6, the reverse exploration starts directly from user-specified final simulation states without previous forward simulations.

Figure 8 shows the average number of visited and valid states during the reverse exploration for 8 vehicles. We can see that the number of states increases immensely with each additional time step. On average, about 1.5×10^{10} states were visited at time step 0. Of these, an average of about 1.3×10^6 were identified as valid states leading to the final state generated by the forward simulation.

In Figure 9, we plot the visited and valid states at time step 0 for the considered vehicle counts. The results demonstrate that in this densely populated scenario, a state space explosion occurs with even modest numbers of vehicles. The main reason is the large number of possible lane changes: in less crowded scenarios, only few opportunities for lane changes exist, strongly reducing the number of possible previous states. Further, as we will see in Section 6, even small-scale scenarios as the one considered here can be sufficient to answer questions on the reachability of certain simulation states.

With 8 vehicles in the scenario, our current sequential implementation of the reverse exploration explored around 3.7×10^6 states per second wall-clock time. In future work, a parallelization across the independent paths in the state space could substantially accelerate the exploration. The memory consumption of the reverse exploration is similar to the reversible simulation, with most memory being consumed by the transition tables. Since we are employing a depth-first search of the state space, it is not necessary to keep all valid states at each time step in memory. Instead, each valid state is immediately explored backward in time and discarded once all previous states have been visited. Opportunities for increasing the tractable scenario scale are discussed in Section 7.

6 CASE STUDY: DETERMINING ACCIDENT CAUSES

In the following, we demonstrate the use of the reverse exploration to study the reachability of specified road traffic situations. Reverse state space explorations are started directly from specified final simulation states without relying on previous forward simulation runs. Further, we compare the cost of the reverse exploration to a state space exploration using forward simulations.

As a first case study we determine the closest vehicle distance that can occur on a two-lane road after a lane change to avoid a stationary obstacle. The final states from which reverse explorations are started are illustrated in Figure 10a: two vehicles are positioned on the right lane at all possible degrees of overlap, reflecting accident situations. On the left lane, an obstacle is placed at 150m from the start of the lane. We carried out a parameter sweep across the vehicle velocities v_1, v_2, the position p of the first vehicle, and the distance Δp to the second vehicle, with −4.5m representing a full overlapping of the vehicles. The parameters were chosen from the following intervals using a granularity of 0.125: $v_1, v_2 \in [0, 20]\,\frac{\mathrm{m}}{\mathrm{s}}$,

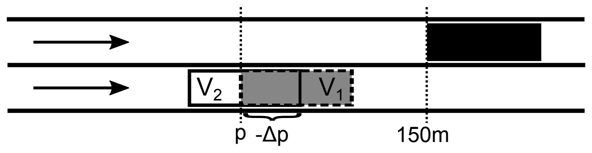
[3]https://github.com/gperftools

$p \in [100, 150]$m, and $\Delta p \in [-4.5, 0]$m. For each of the 3.8×10^8 parameter combinations, a reverse exploration by 10 steps was attempted.

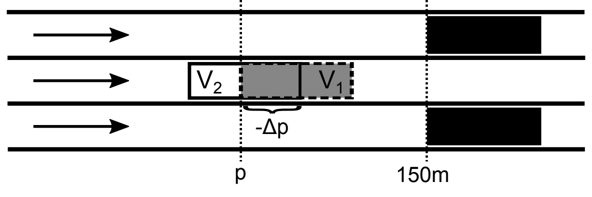
In total, the experiment required about 30 minutes of wall-clock time. No initial states could be identified that lead to any of the generated accident situations. Repeating the exploration starting with the reversal of a car-following step instead of a lane-changing step also did not yield any valid initial states, which is in accordance with the known collision-freeness of the Intelligent Driver Model at small time step sizes. From this observation, we can conclude that it is impossible for the models to generate vehicle overlaps in the considered configuration and scenario.

To identify the number of initial states leading to the minimum possible gap between the vehicles, we repeated the experiment with a fixed $\Delta p = 0.125$m. Now, 131,493 initial states leading to the generated final state were identified after about 80 seconds of wall-clock time. Due to the limited precision of the number representation and the reachability of most velocities both through acceleration and deceleration, final states may be reachable from a number of initial states. For instance, 388 unique initial states lead to the same final state with $v_1 = 15.125\frac{\text{m}}{\text{s}}$, $v_2 = 4\frac{\text{m}}{\text{s}}$, and $p = 143.125$m. For vehicle 1, the initial states cover velocities of $12.75\frac{\text{m}}{\text{s}}$ to $12.875\frac{\text{m}}{\text{s}}$ with a constant starting position of 129.875m. The starting velocities of vehicles 2 range from $6.25\frac{\text{m}}{\text{s}}$ to $16.625\frac{\text{m}}{\text{s}}$, with starting positions from 132.5m to 133.5m.

In our second case study, we study a known issue of the MOBIL model: since lane changes are based purely on the states of vehicles on adjacent lanes, vehicles traveling two lanes apart may change into the same position on the middle lane. The scenario is shown in Figure 10b: we configured a three-lane road with two obstacles at 150m on the outer lanes. Two vehicles traveling in the direction of the obstacles are placed on the middle lane so that they overlap by a configurable amount, reflecting an accident. We carried out a parameter sweep across the same values as in case study 1, attempting a reverse exploration by 10 steps at each parameter combination.



(a) Case study 1: vehicles V_1 and V_2 overlap by $-\Delta p$ m on a two-lane road with an obstacle at 150m.



(b) Case study 2: vehicles V_1 and V_2 overlap by $-\Delta p$ m on the middle lane of a three-lane road with obstacles at 150m.

Figure 10: The final simulation states in the case studies exploring the reachability of accident situations.

Overall, the exploration required about 78 minutes of wall-clock time to identify 3,089,142 initial states leading to the specified final states. As expected, the considered class of accidents only occurs through lane changes from the outer lanes. The exploration correctly identified the symmetry in the initial states, i.e., there are two sets of 1,544,571 initial states each, with the first vehicle starting on the left or right lane, respectively. The configured sensing range of 40m defines the minimum distance from the obstacles at which a lane change, and thus a potential accident, may occur. Figure 11 shows the frequencies of final distances from the obstacles for all the identified initial states. We can see that accidents may occur at a variety of positions, with a maximum distance of 40m from the obstacles, and a minimum distance of 11.625m. However, in most cases, the concurrent lane changes that lead to the accident occur at distances close to 40m. As in the first case study, we repeated the exploration starting with a reverse car-following update instead of a lane-changing update. Again, no initial states leading to any of the generated final states were identified. We conclude that in the considered combination of scenario and simulation models, concurrent lane changes to the middle lane are the **only** possible cause for accidents.

We now determine the cost of arriving at the same conclusion through forward simulations. The objective of the second case study as described above is to exhaustively enumerate the set of initial states that lead to an accident on the middle lane at a distance up to 50m from the obstacles, after completing 10 simulation steps. To conclusively determine all relevant initial states, forward simulations are carried out starting from all initial states that may plausibly lead to an accident. Here, plausibility is contingent on knowledge of the model behavior: at one extreme, a forward state space exploration without any model knowledge could be performed in a black-box fashion by executing a simulation of up to 10 time steps from each initial state permitted by the model. At the other extreme, perfect model knowledge would allow us to directly enumerate the initial states leading to accidents, rendering the study unnecessary. In practice, partial model knowledge allows us to eliminate a subset of initial states from consideration, and to terminate simulation runs after fewer than 10 time steps if it is guaranteed that no accident can occur.

Given that we are considering simulations spanning 10 time steps of 0.1s each and are interested in accidents up to 50m from the obstacles, we can rely on the maximum velocity of $20\frac{\text{m}}{\text{s}}$ to only

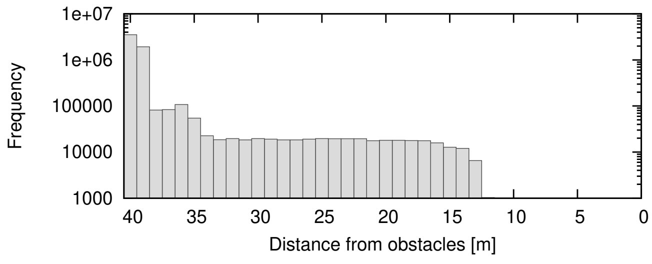


Figure 11: Frequencies of final distances from the obstacles in case study 2. Most accident situations occur immediately once the obstacles enter the sensing range of 40m.

Table 4: Absolute and relative number of parameter combinations, number of visited states, and running time required for forward and reverse exploration in case study 2.

Granularity [m], [m/s]	#Parameter combinations			#Visited states			Running time		
	Reverse	Forward	*Factor*	Reverse	Forward	*Factor*	Reverse	Forward	*Factor*
0.5	1.70×10^6	6.73×10^7	*39.65*	3.04×10^7	4.62×10^8	*15.21*	21s	448.7s	*21.37*
0.25	2.51×10^7	1.04×10^9	*41.50*	4.33×10^8	7.23×10^9	*16.68*	305.7s	6 899.3s	*22.57*
0.125	3.85×10^8	1.63×10^{10}	*42.50*	6.55×10^9	1.15×10^{11}	*17.48*	4 697.3s	113 307.3s	*24.12*

consider initial distances from the obstacles up to 70m. Since the two vehicles behave identically, we also avoid initial states that are symmetrical to previously considered initial states. Using further model knowledge and the specification of our case study, we can formulate the following conditions at which a running forward simulation can be terminated:

- The front vehicle is outside the sensing range of the vehicle behind, while driving at least at the same velocity. In this case, both vehicles accelerate according to the free road term of the Intelligent Driver Model, which does not allow the vehicle behind to catch up.
- An accident occurs and the vehicles are further than 50m from the obstacles, or fewer than 10 time steps have passed.
- Since the case study targets accidents in front of the obstacles, a simulation run is terminated once either of the vehicles passes the obstacle.

Further reductions of the considered states could be achieved by a more detailed analysis of the model-specific acceleration and lane-changing behavior by the user. In contrast, the reverse exploration does not rely on such user-specified conditions and instead eliminates unreachable states from consideration automatically. The parameters for the initial states were chosen from the following intervals: $v_1, v_2 \in [0, 20]\frac{\text{m}}{\text{s}}$, $p_1, p_2 \in [80, 150]$m. Symmetries are exploited by skipping any initial state with $p_2 < p_1$ and by placing vehicle 1 only on the left or middle lane, and vehicle 2 only on the right or middle lane.

Table 4 compares the number of parameter combinations, the number of explored states, and the running time of the forward and reverse exploration for granularities of 0.5, 0.25, and 0.125. The running times are averages of 3 runs. Disregarding symmetrical states, the initial states leading to accidents identified by the forward and reverse exploration were identical. We can see that even when discarding state trajectories based on model knowledge and by exploiting symmetries, the forward exploration considers a substantially larger number of parameter combinations. Compared to the reverse exploration, 39.65 to 42.50 times more combinations were considered. A factor of 15.21 to 17.48 more states were visited by the forward exploration. As a consequence, the forward exploration required a factor of 21.37 to 24.12 more time than the reverse exploration. From these results, we conclude that in our case study, even if model knowledge is applied to reduce the space of initial and intermediate simulation states to consider, forward exploration requires more time than reverse exploration to exhaustively determine the initial states that lead to an accident. In Section 8.2, we discuss techniques for simulation state space exploration that could further reduce the running time of both the forward and reverse exploration.

7 DISCUSSION

The limitations of our approach include the limited spatial granularity of the representations of vehicle velocities and positions, and the vast number of states that must be visited during the reverse exploration. In the following, we discuss opportunities to improve the scalability and performance of the approach. Further, our evaluation and case studies relied on scenarios comprised of a single multi-lane road. We sketch future work to extend the approach towards more complex scenarios and potential uses of the approach in practical safety studies.

7.1 Scalability and performance

7.1.1 Table size. The size of the forward and backward transition tables depends on the chosen granularity of velocities and positions. At a granularity of 0.0625, the table size is within the memory capacities of common workstations. Extrapolating from our measurement results, a granularity of 0.03125 would require about 36 GiB of main memory, an amount beyond the capacities of most current workstations but commonly available in the compute nodes of supercomputing environments.

If the velocity function of the considered car-following model can efficiently be reversed analytically or numerically, the reliance on transition tables could be avoided entirely. Since there may be multiple valid previous vehicle states (cf. Section 3.2), garbage bits are still be required for disambiguation. While we consider the memory consumption of the tables the main limitation of our work, efficient numerical and analytical methods are likely to be model-specific.

In Section 5.2, we saw that the number of required of garbage bits per time step is lower at finer granularities. Thus, the dynamic memory consumption of the reversible simulation can be minimized by choosing the finest granularity that still allows the tables to be stored in memory.

7.1.2 Concurrency. During a reverse simulation, the vehicle transitions are performed front-to-back starting from the first vehicle on each lane. This ordering is necessary because the backward transition of each vehicle depends on the previous state of the vehicle ahead. Compared to a forward simulation, the concurrency of the state updates is reduced: while reverse simulation serializes the updates on each lane, the forward simulation can perform updates for all vehicles in parallel. The need for serialization of the backward transitions is inherent to the fact that the car-following model specifies the relationship between the states of two consecutive vehicles V_{follower} and V_{leader} at time step n and the state of V_{follower} at time step $n+1$. Since this relationship does not include the state of

V_{leader} at $n+1$, the state of V_{leader} at time step n must be determined prior to the backward transition of V_{follower}.

The scalability of the reverse exploration towards larger scenarios and time spans is mostly limited by the vast number of possible previous lane positions of vehicles. Multiple options present themselves to counter the problem of state explosion: firstly, probabilities could be assigned to previous states depending on the question to be answered using the exploration. Then, a heuristic exploration could be carried out by only visiting previous states above a minimum probability. Secondly, the reverse exploration is highly amenable to parallelization: while each path of the depth-first search backwards in simulation time must be traversed sequentially, all valid states at a given step in simulation time can be explored in parallel.

7.2 Generality

7.2.1 Stochasticity. In practice, simulation-based traffic studies may rely on simulation models employing stochastic processes to represent the variability in aspects such as the driver behavior. The reversal of pseudo-random number generation is well-studied and has been applied to a number of generators [20, 33, 45]. Some car-following models include stochasticity by adding random numbers to a vehicle's acceleration or velocity [16, 35, 41]. The reversible simulation could support this type of stochastic model by subtracting the previously used random number at each time step before consulting the backward transition table. However, if the random process depends on previous states, as is commonly the case, a more elaborate reversal scheme is required.

The uses of the reverse exploration in stochastic traffic simulations could be two-fold: firstly, by exploring across all possible previous states given the defined bounds on the stochasticity in the models, the reachability of certain traffic situations can be proven or disproven. Secondly, based on the underlying probability distributions used by the stochastic models, previous states may be assigned probabilities, enabling statements on the probability of different state sequences towards a given final state.

7.2.2 Application to road networks. When considering an extension of the approach to entire road networks, we must consider changes in speed limits. As the velocity function is parametrized with a desired velocity, transition tables are required for each possible speed limit in the network. Depending on the chosen granularity of velocities and positions, the memory requirements limit the scale of road network that can be considered.

Further challenges are posed by the vehicle behavior at road boundaries: when a vehicle is close to the start of a road segment, it may have resided at the current or the previous road at the previous time step. While it may often be sufficient to resolve such situations by consulting the tables corresponding to the speed limit of both the previous and the current road, ambiguous situations may make it necessary to store additional garbage bits.

Reverse transitions are executed front-to-back starting with the leading vehicle on each lane. In a road network, vehicles may be affected by other vehicles within their sensing range on the road segment ahead. Thus, for each chain of vehicles within sensing range, the leader vehicle from which the reversal can be started must be identified first.

7.2.3 Use cases in safety analysis. In Section 6, we applied the reverse state exploration to explore side-impact collisions caused by a simultaneous lane changes. We also showed that collisions cannot result purely from car-following behavior. However, this statement is valid only for the chosen time step size. In future work, our approach could be applied to determine the maximum time step size that preserves the property of collision-freeness with respect to the car-following and lane-changing models.

To enable more practical safety analyses, a car-following model could be chosen that does not guarantee collision avoidance, enabling studies on rear-end collisions involving two or more vehicles. Accidents involving pedestrians could be analyzed by representing the pedestrians as obstacles that suddenly appear on the road.

Focusing on lane-changing behavior, our approach could be used to study head on collisions on two-lane bidirectional roads. Furthermore, side collisions could be studied to evaluate virtual intersection control mechanisms.

Finally, another class of use cases is introduced by autonomous vehicles (AVs). By providing reversible models of features such as cooperative adaptive cruise control, the reverse exploration could support studies on accidents among AVs and human drivers, among AVs and pedestrians, and within platoons.

8 RELATED WORK

In the following, we relate our reversible simulation and reverse exploration schemes to existing work from the literature on parallel simulation, state space exploration, and model checking.

8.1 Rollbacks in parallel simulation

In optimistic parallel and distributed simulations, each logical process (LP) that executes a segment of the simulation carries out computations speculatively. If a computation is later invalidated by a message from a remote LP, the receiving LP is rolled back to a previous point in simulated time. There are two general approaches to enable rollbacks: state saving and reversible simulation.

When using *state saving*, the information required to roll back to a previous simulation state is stored in memory. We can further differentiate checkpointing, which periodically stores the entire simulation state, from incremental state saving, which stores only the changes from the previous simulation state. A number of existing works focus on the automation of the state saving process [6, 29, 38, 40]. In our reversible simulation scheme, the storing of additional garbage bits at each time step bears some similarity to incremental state saving and shares with the delta encoding employed by LaPre et al. [17] the aim of storing only the minimum amount of data to enable a perfect reversal. However, instead of storing the state differences, we only store disambiguation information required to select the correct forward transition out of a transition set that can be determined based on fixed-sized lookup tables.

In *reversible simulation*, techniques from reversible computing are applied to enable rollbacks without the need for state saving. Instead, rollbacks are carried out by executing code that reverses the operations of the forward simulation program. Generic code transformation tools have been proposed that enable the creation of reverse model code without user intervention. In contrast to these generic approaches, our reversible execution scheme relies on model knowledge to store only the number of garbage bits required to disambiguate among the possible previous states.

Reversible simulation techniques have been proposed for model domains such as wireless communication [30], physics [25, 33], epidemics [26], and electronic design automation [14, 23]. To our knowledge, Perumalla and Yoginath [43, 44] are the only authors who proposed reversible execution schemes tailored to microscopic traffic simulation. In their works, substantial performance gains are achieved through the optimistic execution of a reversible discrete-event traffic model. In contrast to their proposal of models that permit a reversible execution, we enable the reversible execution of established models for car-following and lane-changing behavior widely analyzed and applied in the traffic engineering literature.

8.2 Simulation state space exploration

Simulation studies often involve the exploration of vast numbers of state trajectories. While traditional parameter studies vary the simulation input parameters, computational steering [37] alters the simulation state at runtime to explore branches of the state space.

Simulation cloning is a method to exploit overlaps in the simulation state across multiple state trajectories: duplication of computations and data is avoided by representing identical parts of the simulation state only once. Hybinette and Fujimoto investigated simulation cloning in the context of parallel discrete-event simulations [13]. An LP is cloned, i.e., a copy is created in memory, either actively to generate a new branching point, or when it interacts with another LP from a different simulation branch. This process generates a cloning tree where each node represents the simulation run that the child nodes have been cloned from. More recently, methods for simulation cloning have been studied targeting agent-based simulations [19] and on large-scale GPU platforms [46]. Our reverse exploration approach is currently implemented sequentially and does not attempt to avoid redundant computations across the reachable reverse simulation state trajectories. When exploring multiple branches of the state space in parallel, simulation cloning could reduce the running time and memory consumption.

As noted by Hybinette and Fujimoto [13] and also investigated in simulation-based optimization [2, 28], state space explorations may be accelerated further by early termination of runs if it is guaranteed that the simulation will not exhibit the desired behavior. In forward exploration, the decision for early termination relies on user-specified criteria. In contrast, since the reverse exploration starts directly from targeted final simulation states, unreachable previous state trajectories are automatically discarded based purely on the reverse state transitions permitted by the model. Nevertheless, model knowledge is still required to specify relevant final simulation states. In Section 6, we showed results from a case study comparing the cost of forward exploration using early termination of runs to the reverse exploration approach.

The techniques of updatable simulation [8] and exact-differential simulation [12] rely on rollbacks to support forward state space explorations: an initial simulation run generates an event log that permits the creation of branching points at arbitrary points in simulated time, thus allowing for a state exploration without frequent recomputations. The main difference to our reverse exploration scheme lies in the reliance on forward state transitions: updatable simulation and exact-differential simulation only roll back along state trajectories that have previously been executed in a forward simulation. In contrast, the reverse exploration starts directly from user-specified final simulation states.

Finally, memoization [1] is a generic approach to avoid repeated computations that has also been applied to forward state space explorations: at runtime, computation results are stored in memory and retrieved whenever the same computation occurs again. While early methods required modifications of user code [39], research moved towards methods applicable without user intervention, e.g., in the form of a generic middleware for discrete-event simulators [5]. Recently, Stoffers et al. presented a memoization approach targeting parameter studies [32]. For user-annotated code blocks, memoization is applied through automatic code transformation. Our reversible simulation scheme relies on lookup tables to determine the set of forward transitions leading to the same simulation state. In contrast to memoization, the lookup tables are generated a priori and remain unchanged over the course of the simulation.

8.3 Model checking

Model checking is a family of techniques to automatically verify properties of system models [3]. Typically, model properties are verified by proving or disproving the reachability of certain model states through an exhaustive exploration of relevant segments of the state space. Model checking is commonly used to evaluate and verify models in hardware design and distributed systems.

Several works considered the use of model checking techniques to study multi-agent systems [4, 27, 36]. Frameworks such as MCMAS [21] are able to check properties of any multi-agent system expressed in the supported formalism, e.g., computation tree logic.

A number of works have applied model checking techniques to the road traffic domain, focusing on safety-critical applications based on wireless inter-vehicle communication [7, 9, 22]. In these works, exhaustive state space exploration was shown to be feasible for scenarios of up to five vehicles.

Due to the enormous number of possible starting conditions to be explored, stochastic model checking can be applied to derive statistical results from partial state space explorations [18].

The reverse exploration approach of our present work can be seen as reverse-in-time model checking. Instead of an exploration of the state space through forward simulations, we specify final simulation states and explore the state space backwards in time.

9 CONCLUSIONS

In this paper, we presented an approach for reversible simulation and reverse exploration of microscopic traffic models. Perfect reversal of a previous forward simulation is achieved by storing about 1 bit of data per time step and vehicle for disambiguation among possible previous states. We presented the use of the reverse exploration for exhaustively studying the possible causes for a class of accident situations. The performance evaluation of the reverse exploration showed that our current implementation can visit multiple million previous states per second of wall-clock time. The main directions for future work are two-fold: firstly, reducing the constant memory consumption of the approach will enable a finer granularity in the representation of vehicle velocities and positions. Secondly, to support more complex safety analyses, reversible stochastic models and manoeuver protocols should be developed.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] Harold Abelson and Gerald J. Sussman. 1996. *Structure and Interpretation of Computer Programs, Second Edition.* MIT Press.

[2] Philipp Andelfinger, Sajeev Udayakumar, Wentong Cai, David Eckhoff, and Alois Knoll. 2018. Model Preemption Based on Dynamic Analysis of Simulation Data to Accelerate Traffic Light Timing Optimisation. In *Proceedings of the Winter Simulation Conference (WSC).* IEEE, 652–663.

[3] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. 2013. *Systems and Software Verification: Model-Checking Techniques and Tools.* Springer Science & Business Media.

[4] Rafael H Bordini, Michael Fisher, Willem Visser, and Michael Wooldridge. 2006. Verifying Multi-Agent Programs by Model Checking. *Autonomous Agents and Multi-Agent Systems* 12, 2 (2006), 239–256.

[5] Abhishek Chugh and Maria Hybinette. 2004. Towards Adaptive Caching for Parallel and Discrete Event Simulation. In *Proceedings of the Winter Simulation Conference, 2004.*, Vol. 1. IEEE, 433–441.

[6] Davide Cingolani, Alessandro Pellegrini, and Francesco Quaglia. 2017. Transparently Mixing Undo Logs and Software Reversibility for State Recovery in Optimistic PDES. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 27, 2 (2017), 11.

[7] Madeleine El-Zaher, Jean-michel Contet, Pablo Gruer, and Franck Gechter. 2011. Towards a Compositional Verification Approach for Multi-Agent Systems: Application to Platoon System. In *International Workshop on Verification and Validation of Multi-Agent Models for Complex Systems.*

[8] 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 Workshop on Parallel and distributed simulation.* IEEE, 107–114.

[9] Bruno Ferreira, Fernando AF Braz, Antonio AF Loureiro, and Sergio VA Campos. 2015. A Probabilistic Model Checking Analysis of Vehicular Ad-Hoc Networks. In *Vehicular Technology Conference.* IEEE, 1–7.

[10] David Goldberg. 1991. What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computing Surveys (CSUR)* 23, 1 (1991), 5–48.

[11] Solomon Golomb. 1966. Run-Length Encodings. *IEEE Transactions on Information Theory* 12, 3 (1966), 399–401.

[12] Masatoshi Hanai, Toyotaro Suzumura, Georgios Theodoropoulos, and Kalyan S Perumalla. 2015. Exact-Differential Large-Scale Traffic Simulation. In *Conference on Principles of Advanced Discrete Simulation (PADS).* ACM, 271–280.

[13] Maria Hybinette and Richard M Fujimoto. 2001. Cloning Parallel Simulations. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 11, 4 (2001), 378–407.

[14] Glenn Jennings. 1991. Reversible Functional Simulation for Digital System Design. In *Custom Integrated Circuits Conference.* IEEE, 8.2/8–8.2/4.

[15] Arne Kesting, Martin Treiber, and Dirk Helbing. 2007. General Lane-Changing Model MOBIL for Car-Following Models. *Transportation Research Record* 1999, 1 (2007), 86–94.

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

[17] Justin M LaPre, Elsa J Gonsiorowski, Christopher D Carothers, John Jenkins, Philip Carns, and Robert Ross. 2015. Time Warp State Restoration via Delta Encoding. In *Proceedings of the Winter Simulation Conference.* IEEE, 3025–3036.

[18] Axel Legay, Benoît Delahaye, and Saddek Bensalem. 2010. Statistical Model Checking: an Overview. In *International Conference on Runtime Verification.* Springer, 122–135.

[19] Xiaosong Li, Wentong Cai, and Stephen J Turner. 2017. Cloning Agent-Based Simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 27, 2 (2017), 15.

[20] Xinhu Liu and Philipp Andelfinger. 2017. Time Warp on the GPU: Design and Assessment. In *Conference on Principles of Advanced Discrete Simulation (PADS).* ACM, 109–120.

[21] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. 2009. MCMAS: A Model Checker for the Verification of Multi-Agent Systems. In *International Conference on Computer Aided Verification.* Springer, 682–688.

[22] Till Neudecker, Natalya An, and Hannes Hartenstein. 2013. Verification and Evaluation of Fail-Safe Virtual Traffic Light Applications. In *Vehicular Networking Conference.* IEEE, 158–165.

[23] Kalyan Perumalla and Richard Fujimoto. 2003. Using Reverse Circuit Execution for Efficient Parallel Simulation of Logic Circuits. In *Mathematics of Data/Image Coding, Compression, and Encryption V, with Applications*, Vol. 4793. International Society for Optics and Photonics, 267–276.

[24] Kalyan S Perumalla. 2013. *Introduction to Reversible Computing.* Chapman and Hall/CRC.

[25] Kalyan S Perumalla and Vladimir A Protopopescu. 2013. Reversible Simulations of Elastic Collisions. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 23, 2 (2013), 12.

[26] Kalyan S Perumalla and Sudip K Seal. 2010. Reversible Parallel Discrete-Event Execution of Large-scale Epidemic Outbreak Models. In *Workshop on Principles of Advanced and Distributed Simulation (PADS).* IEEE, 106–113.

[27] Anand S Rao and Michael P Georgeff. 1993. A Model-Theoretic Approach to the Verification of Agent-Oriented Systems. In *International Joint Conference on Artificial Intelligence.* Citeseer, 318–324.

[28] Saman Razavi, Bryan A Tolson, L Shawn Matott, Neil R Thomson, Angela MacLean, and Frank R Seglenieks. 2010. Reducing the Computational Cost of Automatic Calibration through Model Preemption. *Water Resources Research* 46, 11 (2010).

[29] Markus Schordan, Tomas Oppelstrup, David Jefferson, Peter D Barnes Jr, and Dan Quinlan. 2016. Automatic Generation of Reversible C++ Code and its Performance in a Scalable Kinetic Monte-Carlo Application. In *Conference on Principles of Advanced Discrete Simulation (PADS).* ACM, 111–122.

[30] Sudip K Seal and Kalyan S Perumalla. 2011. Reversible Parallel Discrete Event Formulation of a TLM-Based Radio Signal Propagation Model. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 22, 1 (2011), 4.

[31] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. 2017. On a Formal Model of Safe and Scalable Self-Driving Cars. *arXiv preprint arXiv:1708.06374* (2017).

[32] Mirko Stoffers, Daniel Schemmel, Oscar Soria Dustmann, and Klaus Wehrle. 2016. Automated Memoization for Parameter Studies Implemented in Impure Languages. In *Conference on Principles of Advanced Discrete Simulation (PADS).* ACM, 221–232.

[33] Yarong Tang, Kalyan S Perumalla, Richard M Fujimoto, Homa Karimabadi, Jonathan Driscoll, and Yuri Omelchenko. 2005. Optimistic Parallel Discrete Event Simulations of Physical Systems using Reverse Computation. In *Workshop on Principles of Advanced and Distributed Simulation (PADS).* IEEE, 26–35.

[34] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. 2000. Congested Traffic States in Empirical Observations and Microscopic Simulations. *Physical Review E* 62, 2 (2000), 1805.

[35] Martin Treiber, Arne Kesting, and Dirk Helbing. 2006. Delays, Inaccuracies and Anticipation in Microscopic Traffic Models. *Physica A: Statistical Mechanics and its Applications* 360, 1 (2006), 71–88.

[36] Wiebe Van Der Hoek and Michael Wooldridge. 2002. Model Checking Knowledge and Time. In *International SPIN Workshop on Model Checking of Software.* Springer, 95–111.

[37] Jeffrey Vetter and Karsten Schwan. 1997. High Performance Computational Steering of Physical Simulations. In *Proceedings of the International Parallel Processing Symposium.* IEEE, 128–132.

[38] George Vulov, Cong Hou, Richard Vuduc, Richard Fujimoto, Daniel Quinlan, and David Jefferson. 2011. The Backstroke Framework for Source Level Reverse Computation Applied to Parallel Discrete Event Simulation. In *Proceedings of the Winter Simulation Conference (WSC).* IEEE, 2960–2974.

[39] Kevin Walsh and Emin Gün Sirer. 2004. Staged Simulation: A General Technique for Improving Simulation Scale and Performance. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 14, 2 (2004), 170–195.

[40] Darrin West and Kiran Panesar. 1996. Automatic Incremental State Saving. *ACM SIGSIM Simulation Digest* 26, 1 (1996), 78–85.

[41] Rainer Wiedemann. 1974. Simulation des Strassenverkehrsflusses. Habilitation Thesis. Institute for Traffic Engineering, University of Karlsruhe.

[42] Rosemary H Wild and Joseph J Pignatiello Jr. 1994. Finding Stable System Designs: a Reverse Simulation Technique. *Commun. ACM* 37, 10 (1994), 87–99.

[43] Srikanth B Yoginath and Kalyan S Perumalla. 2008. Parallel Vehicular Traffic Simulation using Reverse Computation-Based Optimistic Execution. In *Workshop on Principles of Advanced and Distributed Simulation (PADS).* IEEE, 33–42.

[44] Srikanth B Yoginath and Kalyan S Perumalla. 2009. Reversible Discrete Event Formulation and Optimistic Parallel Execution of Vehicular Traffic Models. *International Journal of Simulation and Process Modelling* 5, 2 (2009), 104–119.

[45] Srikanth B Yoginath and Kalyan S Perumalla. 2018. Efficient Reversible Uniform and Non-Uniform Random Number Generation in UNU.RAN. In *Annual Simulation Symposium.* Society for Computer Simulation International, 2.

[46] Srikanth B Yoginath and Kalyan S Perumalla. 2018. Scalable Cloning on Large-Scale GPU Platforms with Application to Time-Stepped Simulations on Grids. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 28, 1 (2018), 5.

[47] Alwin Zulehner and Robert Wille. 2017. Make it Reversible: Efficient Embedding of Non-Reversible Functions. In *Conference on Design, Automation & Test in Europe.* European Design and Automation Association, 458–463.