Towards Differentiable Agent-Based Simulation

PHILIPP ANDELFINGER, University of Rostock, Germany

Simulation-based optimization using agent-based models is typically carried out under the assumption that the gradient describing the sensitivity of the simulation output to the input cannot be evaluated directly. To still apply gradient-based optimization methods, which efficiently steer the optimization towards a local optimum, gradient estimation methods can be employed. However, many simulation runs are needed to obtain accurate estimates if the input dimension is large. Automatic differentiation (AD) is a family of techniques to compute gradients of general programs directly. Here, we explore the use of AD in the context of time-driven agent-based simulations. By substituting common discrete model elements such as conditional branching with smooth approximations, we obtain gradient information across discontinuities in the model logic. On the examples of a synthetic grid-based model, an epidemics model, and a microscopic traffic model, we study the fidelity and overhead of the differentiable simulations as well as the convergence speed and solution quality achieved by gradient-based optimization compared with gradient-free methods. In traffic signal timing optimization problems with high input dimension, the gradient-based methods exhibit substantially superior performance. A further increase in optimization progress is achieved by combining gradient-free and gradient-based methods. We demonstrate that the approach enables gradient-based training of neural network-controlled simulation entities embedded in the model logic. Finally, we show that the performance overhead of differentiable agent-based simulations can be reduced substantially by exploiting sparsity in the model logic.

CCS Concepts: • **Mathematics of computing → Automatic differentiation**; • **Theory of computation → Continuous optimization**; **Multi-agent reinforcement learning**; • **Computing methodologies → Modeling methodologies**; **Agent/discrete models;**

Additional Key Words and Phrases: Agent-based simulation, optimization, Backpropagation

ACM Reference format:
Philipp Andelfinger. 2022. Towards Differentiable Agent-Based Simulation. *ACM Trans. Model. Comput. Simul.* 32, 4, Article 27 (November 2022), 26 pages.
https://doi.org/10.1145/3565810

1 INTRODUCTION

Simulation-based optimization comprises methods to determine a simulation input parameter combination that minimizes or maximizes an output statistic [16, 43, 64]. It has applications in a vast array of domains, such as supply chain management [47], transportation [62], crowd modeling [83], and health care [81]. The problem can be viewed as a special case of mathematical optimization in which an evaluation of the objective function is reflected by the execution of one or more simulation runs. Many mathematical optimization methods evaluate not only the objective function

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant no. 497901036.
Author's address: P. Andelfinger, University of Rostock, Institute for Visual and Analytic Computing, Rostock, 18059, Germany; email: philipp.andelfinger@uni-rostock.de.

1049-3301/2022/11-ART27 $15.00
https://doi.org/10.1145/3565810

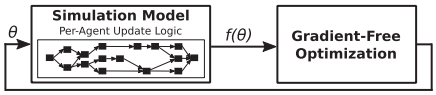
itself but also its partial derivatives to inform the choice of the next candidate solution. Given a suitable initial guess, gradient-based methods efficiently steer the optimization towards a local optimum [68].

In contrast, simulation-based optimization using agent-based models usually relies either on surrogate models [6, 11], which typically abandon the individual-based level of detail of the original simulation model [5], or on gradient-free methods such as genetic algorithms [15]. While gradient-free simulation-based optimization is a time-tested approach, the hypothesis underlying this article is that the targeted local search carried out by gradient-based methods may achieve faster convergence or higher-quality solutions for certain agent-based models. An existing method to obtain gradients targeted towards discrete-event simulations is Infinitesimal Perturbation Analysis (IPA) [40], which the literature applies by determining derivative expressions by a manual model analysis (e.g., [19, 31, 44]), limiting its applicability to relatively simple models. Alternatively, gradients can be estimated based on finite differences. However, in the classical finite differences approaches, the number of required simulation runs grows linearly with the input dimension. Approaches to tackling the computational costs for computing gradients include methods for dimensionality reduction as used in general sensitivity analysis of agent-based simulations [25, 29] and the parallel execution of ensembles of simulation runs on supercomputing clusters.

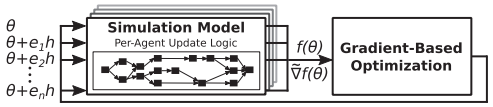
In the field of deep learning, the ability to optimize neural networks with millions of parameters within tolerable time frames rests on gradient information determined using the backpropagation algorithm [69]. Backpropagation is a special case of automatic differentiation, a family of methods to compute derivatives of computer programs written in general-purpose programming languages [58]. In this article, we explore the use of automatic differentiation for gradient-based optimization of agent-based simulations. The main obstacle to this goal is given by discontinuous model elements, which are considered to be among the constituent features of agent-based models [13]. To enable differentiation across such elements, we propose substitutes constructed from known smooth approximations of basic operations. In contrast to classical surrogate modeling approaches, our approach retains the per-agent logic of the original model. The resulting agent-based models are differentiable regarding some or all model aspects, depending on the degree to which discontinuous elements are substituted. Figure 1 contrasts the existing methods with our proposed approach, which we refer to as *differentiable agent-based simulation*.

As our focus is on exploring the merits of this novel approach, we present extensive empirical results based on differentiable implementations of a synthetic cellular model as well as models from the transportation and epidemics domains. We study (1) the fidelity of the results as compared with purely discrete reference models, (2) the fidelity of the gradients in the presence of stochasticity as compared with gradients computed using finite differences, (3) the overhead introduced by relying on smooth model building blocks, and (4) the relative convergence behavior and solution quality in simulation-based optimization problems as compared with gradient-free methods. To further showcase the potential of the approach, we extend the traffic simulation model by embedding neural network-controlled traffic signals in the differentiable model logic. This enables their training based on gradients reflecting the interactions among the static and neural model elements and serves as an example towards a unification of manually constructed representations of domain knowledge in the form of agent-based models and purely data-driven models based on neural networks.

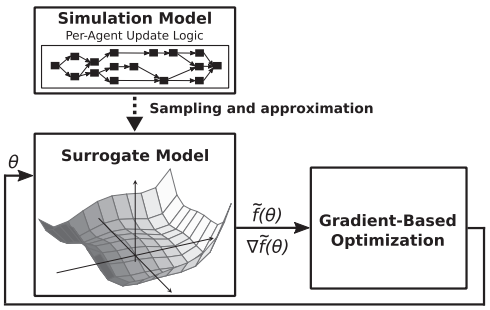
The novel contributions in this extended version of our conference paper [2] include the addition of a synthetic model to assess the fidelity of the output of differentiable simulations and the computed gradients, the combination of gradient-based and gradient-free optimization methods, and the exploitation of sparsity in the agent logic to reduce memory consumption and execution times. Further, the discussion of related work has been extended and updated.



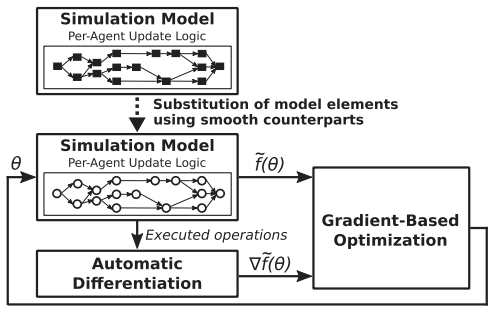
(a) **Gradient-free optimization** relies purely on the observed simulation output.



(b) **Finite differences** require several runs to estimate the gradient at a parameter combination.



(c) **Metamodeling** approaches typically abandon the consideration of individual agents.



(d) **Proposed approach:** Gradient-based optimization on an individual-based level of detail.

Fig. 1. Overview of existing simulation-based optimization methods (a) to (c) and our proposed approach (d). f: simulation, θ: simulation parameters, e_i: unit vector, h: step size for finite differences.

2 AUTOMATIC DIFFERENTIATION

Automatic Differentiation (AD) comprises techniques to computationally determine gradients of computer programs [35, 58]. In contrast with finite differences, AD determines exact partial derivatives for an arbitrary number of either input or output variables from a single program execution. In comparison to symbolic differentiation as implemented in computer algebra systems, AD avoids representing the frequently long derivative expressions explicitly [7].

AD computes derivatives based on the chain rule from differential calculus. In the *forward* mode, intermediate results of differentiating the computational operations with regard to one of the inputs are carried along during the program's execution. At termination, the partial derivatives of all output variables with regard to one input variable have been computed. Thus, given n input variables, n passes are required. Conversely, *reverse-mode* AD computes the derivatives of one output variable with regard to arbitrarily many inputs in a single pass. During the execution of the program, the computational operations and intermediate results are recorded in a graph. At termination, the graph is traversed in reverse, starting from the output. The chain rule is applied at each operation to update the intermediate derivative calculation. When arriving at an input variable, the computed value is the partial derivative of the simulation output with regard to the respective input. Given our use case of simulation-based optimization, for which we expect the input dimension to be larger than the output dimension, the remainder of the article will rely on reverse-mode AD.

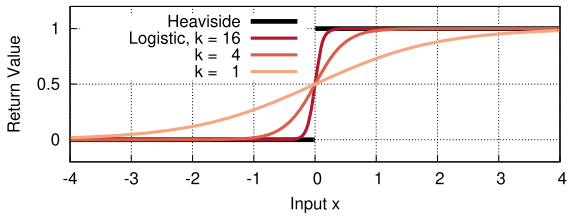
Mature implementations of AD are available for programming languages such as C and C++ [34, 42], Java [73], and Julia [46]. Modern AD tools rely on expression templates [42] or source-to-source transformation [46] to generate efficient differentiation code.

3 DIFFERENTIABLE AGENT-BASED SIMULATION

From a computational perspective, a simulation model implementation applies a sequence of logical and arithmetic operations to the input variables to produce a set of output variables. Our

```
if (x >= x0)
    return 1;
else
    return 0;
```

(a) Conditional branch.



(b) Discrete and smooth representation as functions on real numbers.

Fig. 2. A conditional branch (a) can be represented by the Heaviside function (b) with derivative 0 in any point $\neq 0$. The logistic function $1/(1 + e^{-k(x-x_0)})$ is a well-known smooth approximation that allows the derivatives to capture the effects of taking an alternative branch.

goal is to extract derivatives from model executions that can guide gradient-based optimization methods, which requires the operations to be differentiable and to yield finite and non-zero derivatives. However, typical agent-based models must be expected to contain discontinuous elements: viewed as functions on the reals, model elements such as conditional branching and discrete state transitions may be non-differentiable in certain points or regions of their domain and may carry unhelpful zero-valued derivatives in others. The approach of differentiable agent-based simulation involves the construction of model implementations from differentiable building blocks that act as substitutes for non-differentiable model elements.

Differentiable agent-based simulation shares its approximative nature with methods for metamodeling [6, 11]. However, this family of approaches typically generates representations of the model behavior on an aggregate level, for example, by regression over samples of the input-output relationship defined by the model. In contrast, the bottom-up approach of differentiable agent-based simulation aims to retain the per-agent logic of an discontinuous reference model. Once a differentiable counterpart to a reference model has been constructed, individual agents' state trajectories can be compared directly between the two model variants.

As a basic example, consider the C code fragment of Figure 2(a) containing a conditional statement. For $x_0 = 0$, the function behaves like the Heaviside step function (Figure 2(b)), the derivative of which is zero-valued at $x \neq 0$, and infinite at $x = 0$. A well-known smooth approximation [56] is given by the logistic function: $\mathrm{l}_k(x) = (1 + e^{-k(x-x_0)})^{-1}$, where k determines the steepness of the curve and x_0 shifts the curve along the x axis. In Figure 2(b), we also show the logistic function with $x_0 = 0$, varying k. When increasing k, the logistic function becomes an increasingly closer approximation of the Heaviside step function. Thus, to make our example C function amenable to automatic differentiation, we can simply substitute its body by `return logistic(x, x0);`.

By varying x_0, we can now approximate expressions of the form $x \geq x_0$. We refer to this basic building block as *smooth threshold*. In the remainder of this section, we describe a number of slightly more complex building blocks for agent-based simulations. This bottom-up approach is similar in spirit to the construction of reversible programs [65, 77], which has also found applications in the simulation realm (e.g., [3, 80]). We emphasize that the building blocks described in the following rely on well-known continuous approximations and that the list is far from complete. Our intention is to gather a basic list of constructs as a starting point for model development.

3.1 Conditional Execution and Branching

Arithmetic operations carried out conditionally depending on program input pose further challenges for automatic differentiation. Consider the code fragment `if (x >= x0) y = c; else y = d;`

where x is the input, x0, c, and d are constants, and y is the output. During each execution, the control flow covers exactly one of the branches. Thus, y is always assigned a constant, and $\frac{dy}{dx}$, that is, the sensitivity of y to changes in x, evaluates to 0. By expressing the control flow using a smooth approximation (e.g., as shown in [39]), we can extract a derivative across both branches: `double z = logistic(x, x0); y = z * c + (1 - z) * d;`. We refer to this simple pattern as *smooth branching*.

3.2 Iteration

The core of a typical time-driven agent-based simulation is a nested loop composed of an outer loop that iterates across the timesteps and an inner loop that iterates across the agents. Assuming that the number of timesteps and agents are both constants, the loop represents static — that is, input-independent — control flow, which requires no further preparation for automatic differentiation.

Input-dependent loop conditions can be transformed into branches if an upper bound for the number of iterations is known. For example, variations in the number of agents can be implemented by masking within each loop iteration the behavior of inactive agents using *smooth threshold*.

3.3 Selection of Interaction Partners

The selection of interaction partners based on their attributes is one of the fundamental primitives in agent-based simulations. A straightforward differentiable representation iterates across all agents, masking interactions with those for which a certain condition does not hold. An example is given by the traffic simulation model of Section 4: each vehicle chooses its acceleration based on attributes of the closest vehicle ahead. The selection of the minimum distance can be achieved by iteratively applying a smooth approximation of the minimum function: $log \sum_i e^{x_i}$ [21]. Once the distance to the closest vehicle has been determined, additional attributes are required to carry out the interaction. To support the retrieval of additional attributes, we construct a *select by attribute* building block, which selects an agent's specified "target" attribute based on the known value of a "reference" attribute. This is achieved by iterating across all agents' reference attributes, adding the target attribute if the reference attribute is sufficiently close to the known value.

A downside of this approach is its overhead, as each agent traverses all other agents. In Sections 4 and 5, we present opportunities for performance improvements and evaluate the overhead of different model variants.

3.4 Time-Dependent Behavior

Agents may dynamically alter their behavior throughout the simulation. If the relation between time and behavior is not input dependent, it can be viewed as a static part of the model logic and does not require further consideration with respect to differentiability. An example are actions triggered every n timesteps.

Input-dependent periodic behavior can be expressed using the smooth periodic step function $\mathrm{l}_k(\sin(\frac{\pi t}{p}))$, where t is the current time and p is the period. The action itself is then triggered through *smooth branching*, employing the smoothed periodic step function to mask the action if it is currently inactive.

Timed actions can be scheduled using *smooth timers*: a timer variable v_t is initialized to the desired delta in time and decremented at each timestep, where $\mathrm{l}_k(-v_t)$ serves as a condition for *smooth branching*. When the action associated with the timer shall not occur, v_t is set to a delta beyond the simulation end time.

3.5 Stochasticity

Stochastic elements in agent-based models represent uncertainty on the original system's behavior either due to insufficient knowledge (epistemic uncertainty) or due to the fundamental stochastic nature of certain system behaviors (aleatoric uncertainty) [26]. Here, we consider the case in which a stochastic model has already been constructed, that is, the uncertainty in the output of simulation runs is inherent to the model and, thus, aleatoric in nature.

In model implementations, the stochastic elements are typically represented by pseudo-random number (PRN) generators. For the purpose of differentiation, PRNs drawn based on a given seed throughout a simulation run can be regarded as constants, even if the numbers are drawn at runtime. For instance, in a discrete model, a Bernoulli trial with success probability p based on a PRN r takes the following form: `bool success = r < p;`. Using smooth branching, this is represented as `double success = logistic(p, r);`.

To estimate the gradient of a stochastic model at a given parameter combination, the gradients obtained from runs with different seeds can be aggregated. While simple averaging can yield biased gradient estimates, recent results by Eckman and Henderson indicate that such estimates still steer the optimization near local optima [28].

Importantly, even if PRNs are treated as constants, the effects of subsequent operations on PRNs are still captured by the computed gradients. An example is inverse transform sampling, which transforms uniformly distributed PRNs to a target distribution. In the epidemics model of Section 4, the rate of the target exponential distribution is affected by the simulation input. Here, the sensitivity of the exponential variate to changes in the input is captured by the gradient. A more sophisticated treatment of stochasticity by operating directly on distributions [18] is part of our future work.

4 MODEL IMPLEMENTATION

We prepared five models for automatic differentiation. First, we consider an abstract cellular model in which the cells' states are propagated stochastically across a grid. The second model is an agent-based formulation of the classical susceptible-infected-recovered model extended by movement on a graph. The remaining three models represent road traffic controlled by traffic lights. One model variant allows gradient information to propagate through all model elements but is limited in its scalability. By restricting differentiability to aspects relevant to our use case of simulation-based optimization, two further variants are able to scale to road networks populated by thousands of vehicles.

It is important to note that the benefits of our approach must be expected to depend strongly on the smoothness of the function defined by the simulation model. If the relationship between the simulation inputs and output in the original discrete formulation is non-smooth, gradient-based optimization approaches are likely to become caught on plateaus or low-quality local optima of the simulation output function. The selection of models described in the following aims to cover reference models of varying degrees of smoothness. In the discrete reference models of the cellular and epidemics model, agent states are Boolean or integer values that evolve throughout a sequence of Bernoulli trials, leading to discontinuous jumps in the simulation output. The microscopic traffic model combines a timestepped formulation of a car-following model with traffic lights alternating discretely between the "red" and "green" states. Here, when altering the traffic light phases as offsets in simulated time, jumps in the simulation response may be observed when a change in a traffic light's offset allows a vehicle to pass the light rather than having to brake sharply or vice versa. By the use of the differentiable building blocks, such jumps are smoothed out, allowing us to compute non-zero partial derivatives in a given point of the input space. Nevertheless, depending

on the chosen simulation output function and the degree of smoothing, the remaining "rugged" structure of the simulation response surface may still lead a gradient-based optimization to become caught in low-quality local optima.

4.1 State Propagation on a Cellular Grid

In order to the study the fidelity of the simulation results and gradients computed from a differentiable agent-based simulation, we implemented a synthetic model of state propagation on a grid. Similar to cellular agent-based models in biology [1], medicine [66], and epidemiology [71], Boolean agent states are propagated stochastically throughout a cellular grid. In such models, an agent's state may represent its infection status or its exposure to a piece of information. At each timestep, Bernoulli trials are conducted at each agent whose state is *true* to determine the neighbors to which the state is propagated. The success probability of a trial is a model parameter specific to the receiving agent, that is, the total number of model parameters is equal to the number of cells in the grid.

The differentiable model formulation represents the Boolean agent states as real numbers on $[0, 1]$. The Bernoulli trials take the form described in Section 3.5. *Smooth branching* serves to select cells that propagate their states, that is, cells with states close to or equal to 1. As smooth branching traverses both possible control flow paths, this implies that at each timestep, Bernoulli trials are carried out for the neighbors of all cells, even if their state is close to or equal to 0.

4.2 Epidemics Model on a Graph

This model follows Macal's agent-based formulation [57] of the well-known Susceptible-Infected-Recovered model [49], which imitates the spread of an infection. We extend Macal's model by random movement on a social graph. The model serves to illustrate the viability of automatic differentiation for simulation-based optimization given purely discrete agent states and under strong dependence on stochastic model elements.

An arbitrary number of agents is situated on each node of a static graph. Initially, each agent is either in the "susceptible" or "infected" state, the probability being a model input. At each timestep, each agent a acts as follows: if a is susceptible, each infected agent $a' \neq a$ at the current node infects a with a per-location probability given as an input. If a is newly infected, the delay to the transition to the "recovered" state is drawn from an exponential distribution, the rate being another input. Finally, the agent moves to a neighboring graph node chosen uniformly at random.

For a given seed value, the agents change their locations according to predetermined trajectories. Hence, the overhead of differentiable neighbor search can be avoided by gathering each agent's neighbors from an array updated at each timestep. The key remaining model aspects are constructed from differentiable building blocks. Infections are handled by supplying uniformly distributed random variates and the infection probabilities as input to *smooth branching*. Recovery is an instance of the *smooth timer* building block. Using as input a uniform random variate and the simulation input specifying the recovery rate, we determine the concrete delay until the agent recovers using inverse transform sampling. As the gradient information propagates through the uniform-to-exponential transformation, the computed gradients capture the sensitivity of the simulation output to the configured recovery rate. At each timestep, the *smooth branching* building block is applied to carry out the transition to the "recovered" state once the recovery delay has expired.

4.3 Microscopic Traffic Simulation

The traffic simulations employ model classes encountered in common academic and commercial traffic simulators, such as SUMO [8] and VISSIM [30]. The agents' longitudinal movement is

governed by the Intelligent Driver Model (IDM) [78]. The IDM defines a vehicle's acceleration by an ordinary differential equation, which relates the acceleration of the current vehicle to its own velocity and its distance and velocity to the vehicle in front. In time-driven microscopic traffic simulations, each vehicle determines its acceleration in the next timestep from t to $t + \tau$ based on the vehicle states at t. From the acceleration, the new velocity and position are determined using a suitable integration scheme [4].

For lane changing, we rely on a simplified model similar to MOBIL [50]: every n timesteps, the vehicles determine the projected increase in clearance observed after a hypothetical lane change to the left or right lane, if any. If the clearance increase is beyond a configurable threshold, an instantaneous lane change is carried out.

The traffic is controlled by traffic lights with static or dynamic timing as described later. Overall, we obtain hybrid models composed of an originally continuous car-following behavior, which is discretized through numerical integration and the purely discrete transitions in the lane-changing behavior and traffic light control.

4.3.1 Single Multi-Lane Road. The purpose of this initial road traffic model is to explore the viability of implementing a fully differentiable model, that is, one in which the computed gradients capture the behavior of all model elements, and to study the computed gradients when varying the input parameters. While we hope for the model description to be instructive, we will see that, for practical applications, it is preferable to limit the incurred overhead by restricting the differentiability to selected model elements.

We first consider the car-following model IDM. In a time-driven formulation, it directly relates the new acceleration of a vehicle to its leader's current state, making automatic differentiation of the acceleration update itself straightforward. A challenge lies in determining the leader: in a typical implementation, the vehicles located on a lane are stored in a sorted fashion, rendering the leader selection trivial. In the differentiable variant, we instead determine a vehicle's leader by iterating across all vehicles and selecting the vehicle with the minimum positive position delta. Since we require both the position and velocity of the leader, we arrive at a two-step process. First, we determine the leader's position by repeatedly applying *smooth minimum* as described in Section 3.3, masking negative position deltas using *smooth threshold*. Then, *select by attribute* determines the leader's velocity based on its position and lane.

Lane-changing decisions are made periodically by determining the lane with the largest forward clearance using *smooth maximum*, after which *smooth threshold* is applied to determine whether the clearance justifies a lane change.

A traffic light is positioned on the road, alternating between green and red phases of equal duration using *periodic step function*. The vehicles brake at red lights according to the IDM. This is achieved using *smooth branching* on three conditions: the light is red, the light is still ahead, and the light is closer than the leading vehicle, if any.

In contrast to this fully differentiable model, the variants described next follow a more pragmatic approach that restricts the differentiability to specific model aspects.

4.3.2 Grid Network with Static Signal Timings. In this model variant, the vehicles traverse a grid-shaped network of multi-lane roads connected at the boundaries to form a torus. At each intersection, traffic lights are placed at the incoming roads. The timings of the light phases at the intersections are given as offsets in simulation time, which form the simulation input. When encountering an intersection, the vehicles turn left or right or advance to the road ahead based on configurable probabilities. While the vehicles' behavior with regard to their acceleration, lane changing, and the traffic lights is identical to the previous model variant, the implementation follows a different approach. Since our objective is to maximize the overall vehicle progress by adjusting the traffic

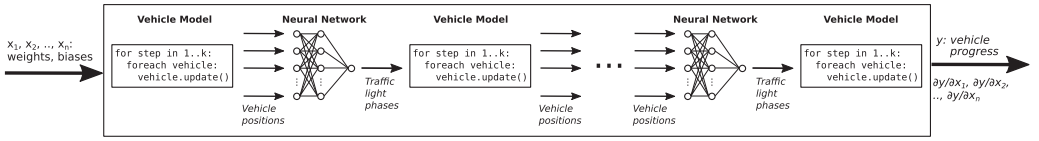


Fig. 3. Neural network-controlled traffic lights embedded in a traffic simulation. As the neural network is part of the differentiable model logic, training can rely on the partial derivatives with regard to the network coefficients returned by the simulation.

light timings, the key model elements for which we aim to extract gradient information are the light control and the longitudinal movement of the vehicles. Thus, the identification of a vehicle's leader, lane changing, and the advancement to adjacent roads follows their natural implementations based on storing and updating the vehicle states in per-lane arrays sorted by position. Hence, they are not captured in the computed gradients. As an example, suppose that a slight change in the simulation input would cause a new lane change, which, in turn, would affect the simulation output. As the model does not offer differentiability across lane changes, the gradient would not reflect this possibility.

Aside from the performance benefits of this approach (see Section 5), its lowered implementation effort suggests that integrating automatic differentiation capabilities in an existing traffic simulator could be possible without excessive development efforts.

In Section 5.3.2, this model variant serves as an example for simulation-based optimization. One input parameter per intersection represents the light phase offset and will be adjusted to maximize the vehicles' progress. To limit the input dimension, existing work considers only small numbers of intersections (e.g., [24]) or reduces the model detail from an individual-based view as used in our work to the mesoscopic or macroscopic level (e.g., [82]).

4.3.3 Grid Network with Neural Network-Controlled Signals. In this model variant, we substitute the traffic light control based on time offsets with a dynamic control using a neural network. The setup is illustrated in Figure 3: the neural network is invoked periodically as part of the model logic. At each decision point, the current positions of all vehicles in the simulation are provided as input to the neural network, its output being the new traffic light phases (red or green) for the horizontal roads at each intersection, the vertical roads being assigned the opposite phase. Since the neural network is implemented as part of the model logic, the gradients extracted through automatic differentiation directly reflect the sensitivity of the vehicles' movement to the neural network's coefficients, enabling gradient-based training in order to optimize the traffic flow. This is in contrast to reinforcement learning approaches, which typically operate under the assumption that the system model is non-differentiable and that the effect of the trained entity's actions must be explored purely by observing the resulting states and "rewards" throughout repeated simulation runs [48]. In our case, each simulation run returns not only an overall reward in the form of the vehicles' progress, but also derivatives that guide the optimization towards a locally optimal traffic light response.

The neural network follows a fully connected feed-forward architecture: there are 5 input neurons for each lane in the road network, the input being the sorted positions of the 5 vehicles closest to the intersection. There is a single hidden layer composed of h neurons. The output layer is composed of one neuron per intersection, yielding the new traffic light states. Given i intersections, 4 incoming roads per intersection, and 3 lanes per road, the architecture results in $(60i+1)h+(h+1)i$ coefficients to be adjusted. All neurons employ the hyperbolic tangent function for activation. The traffic light states returned by the neural network are floating point numbers translated to green

or red phases using *smooth threshold*, a positive value representing a green light on a horizontal road.

5 EXPERIMENTS

Our experiments are intended to answer the overarching research question: *"Can gradient-based simulation-based optimization using the presented differentiable models outperform gradient-free methods?"*

To achieve a benefit during optimization, the fidelity of the differentiable model must be sufficient so that the quality of identified solutions carries over to the non-differentiable reference model. Further, the execution time overhead of the simulation must be small enough not to outweigh potential improvements in convergence speed. Thus, in the remainder of this section, we evaluate the fidelity and overhead of the differentiable model variants. The overall benefit of gradient-based over gradient-free optimization is evaluated in a number of simulation-based optimization experiments. In these experiments, we compare our approach to a number of methods for simulation-based optimization that can operate on the original discrete implementations of our models, avoiding the overheads of the differentiable variants. We also study the benefits of combining gradient-based and gradient-free optimization methods. Finally, we conduct experiments in which sparsity in the agent actions is exploited to reduce the execution times and memory consumption of the differentiable simulations.

The following gradient-free optimization methods are employed: Differential Evolution (DE) [75], Conventional Neural Evolution (CNE) [61], and Simulated Annealing (SA) [54]. As a representative of methods based on finite differences, we also show results using Stochastic Perturbation Stochastic Approximation (SPSA) [74], which has previously been studied in the context of simulation-based optimization [10]. The optimization using differentiable simulation relies on the following gradient-based methods: Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (Adam) [52], and Nadam [27].

In all of the optimization methods, an evaluation at a parameter combination executes a fixed number of simulation runs specified below for each model. The return value is the average across the simulation runs' outputs. An evaluation of the differentiable simulation also returns the averaged partial derivatives of the simulation output with respect to all parameters. Each overall optimization process runs sequentially on a single processor core and is terminated once a time budget specified with respect to wall-clock time is exceeded.

The experiments were conducted on two machines, each equipped with two 16-core Intel Xeon CPU E5-2683v4 and 256 GB RAM, running CentOS Linux 7.9.2009. The automatic differentiation relied on *Adept* [42]. For optimization, we employed the *ensmallen* library [22].

5.1 State Propagation on a Cellular Grid

We first focus on the fidelity of our approach when executing the differentiable implementation of the abstract state propagation model. To this end, we compare the simulation output between the differentiable model and the discrete reference model as well as gradients computed using automatic differentiation and finite differences.

Each simulation is initialized using a single cell with state *true* (or 1.0 in the differentiable simulation) at the center of a 15×15 grid. During initialization, the per-cell success probabilities for the Bernoulli trials are drawn from a uniform distribution on $[0, p_{\max}]$. In total, there are 225 simulation parameters. Each simulation is terminated after 15 steps. As the simulation is highly stochastic, we consider batches of simulation runs and calculate the simulation output and gradients across an entire batch at a time.

Table 1. Mean Absolute Error when Comparing the Final Cell States from the Discrete Cellular Automaton Model and Its Smoothed Counterpart

Slope	$p_{\max} = 0.125$	$p_{\max} = 0.25$	$p_{\max} = 0.5$	$p_{\max} = 1$
2	0.47 ± 0.15	0.30 ± 0.16	0.37 ± 0.08	0.38 ± 0.04
8	0.03 ± 0.04	0.037 ± 0.06	0.044 ± 0.09	0.031 ± 0.07
32	0.007 ± 0.009	0.013 ± 0.01	0.005 ± 0.008	0.001 ± 0.004
128	0.007 ± 0.009	0.012 ± 0.01	0.004 ± 0.007	0.0009 ± 0.004
∞	$\mathbf{0.007 \pm 0.009}$	$\mathbf{0.012 \pm 0.01}$	$\mathbf{0.004 \pm 0.007}$	$\mathbf{0.0009 \pm 0.004}$

For reference, setting the slope of the logistic function to ∞ represents the comparison of two batches of discrete runs. Beyond a slope of 32, the smoothed model closely follows the discrete reference.

Table 1 compares the simulation output from the differentiable and discrete simulations for batches of 1 000 simulation runs. We list the mean absolute error and its standard deviation across the cell states. The slope of the logistic function is varied to determine how closely the differentiable model approximates the discrete reference model. We observe that the error diminishes when increasing the slope of the logistic function. At slopes of 32 and beyond, the mean absolute error is as low as in a comparison of two discrete runs, which is shown on the bottom row of the table.

We also assessed the fidelity of the computed gradient in a synthetic calibration problem compared with two methods based on finite differences. The results, in which we observed that the gradients computed using automatic differentiation most closely reflected the model behavior in most cases, are shown in detail in the supplementary materials.

5.2 Epidemics Model on a Graph

To assess the fidelity of the differentiable Susceptible-Infected-Recovered model, we executed 100 simulations each for 1 000 parametrizations of a scenario populated by 1 000 agents moving across a random geometric graph with 500 nodes and an average degree of 5, each run spanning 10 timesteps. The per-location infection rate coefficients and the initial infection probability were drawn from $U(0, 0.1)$. The recovery rate was drawn from $U(0, 0.01)$. The slope of the logistic function was set to 128. Figure 4 shows a histogram of the percentage of agents attributed to a different state than that in the non-differentiable reference runs. The median deviation amounts to 0.54% of the agents, and the 95% and 99% quantiles are 1.17% and 1.48%, indicating that the differentiable model closely represents the reference model.

The differentiable simulation is associated with substantial overhead: a simulation of 10 000 agents on a graph of 5 000 nodes was slowed down by a factor of 20.1. The memory consumption increased from about 7 MiB to 252 MiB, a factor of 36. In Section 5.5, we will exploit sparsity in the differentiable model to reduce the overhead.

We also conducted an optimization experiment in which we calibrated a simulation of 10 000 agents on a graph of 5 000 nodes to a state trajectory across 10 steps of a randomized reference run. A recent similar calibration effort for an epidemics model operated on a neural surrogate instead of a full agent-based model [5].

Figure 5(a) shows the optimization progress across simulation batches. When considering the progress across simulation batches, we observe that the gradient-based methods achieve similar initial converge speeds compared with the best-performing gradient-free methods CNE and DE. Since practical applications are concerned with the solution quality achieved within a given time or compute budget, Figure 5(b) shows the progress across wall-clock time, which, due to the overhead of the differentiable simulations, is slower with the gradient-based methods. Apart from SPSA and

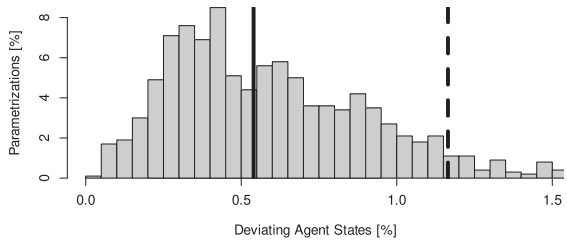
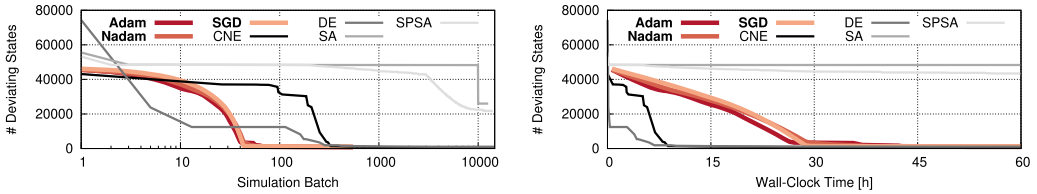


Fig. 4. Comparison of results from the differentiable and non-differentiable epidemics models. The vertical and dashed lines indicate the median and 95% quantile.



(a) Best observed solution across batches of 100 runs each.

(b) Best observed solution across wall-clock time.

Fig. 5. Progress of the epidemics model calibration.

SA, all methods achieved a good fit to the reference trajectory within the time budget of 6 h. CNE and DE achieved the best results with about 0.8% and 1.1% misattributed agent states compared with 2.3% to 2.5% using the gradient-based methods. For comparison, the solutions identified by SPSA and SA misattributed 22% and 26% of the states.

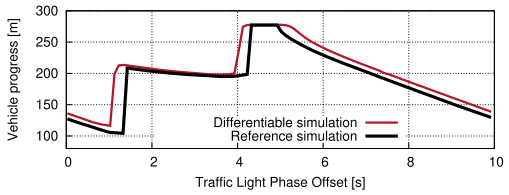
5.3 Microscopic Traffic Simulation

5.3.1 Single Multi-Lane Road. We first study the deviation of the results generated by the fully differentiable model as compared with a non-differentiable reference implementation. The road has three lanes, 250 m in length. A traffic light is positioned at 100 m, with an overall period of 10 s, divided into green and red phases of 5 s each. The speed limit is set to 50 km/h. Lane changes may occur every 2.5 s, requiring a minimum clearance gain of 10 m. The IDM parameters defining the maximum acceleration and deceleration are both set to 2 m/s. Where not otherwise noted, the same parameters are used in the microscopic traffic simulation experiments presented later. The timestep size τ is set to 0.1 s. Initially, we position the vehicle at different lanes in non-zero increments of 40 m from the beginning of the road.

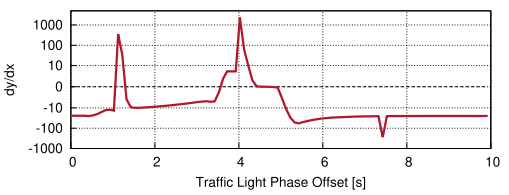
Figure 6(a) compares the vehicle progress in meters throughout a simulation involving two vehicles spanning 10 s of simulation time between the differentiable simulation and the non-differentiable reference. The x axis shows the simulation input, which is the time at which the traffic light first changes to red. We show results with the slope parameter of the logistic function set to 32.

The sharp increases in vehicle progress around 1 s and 4 s reflect situations in which a vehicle passes the light just before it changes to red. The curve for the differentiable simulation slightly deviates from the reference due to the smoothing of the traffic light control, which delays the braking and acceleration when the light changes.

Figure 6(b) shows the derivative of the simulation output with regard to the input parameter determined by automatic differentiation. The derivative expresses the sensitivity of the vehicle progress to changes in the traffic light offset. We can see that, overall, the derivative follows the slope of the simulation output curve, sharp increases in the simulation output being reflected by

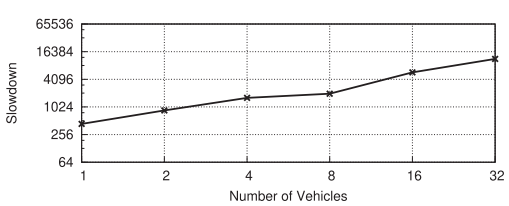


(a) Vehicle progress.

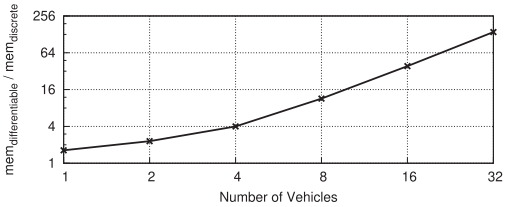


(b) Derivative of the vehicle progress with regard to the phase offset.

Fig. 6. Overall vehicle progress and its derivative in the single-road scenario with two vehicles.



(a) Slowdown.



(b) Relative memory usage.

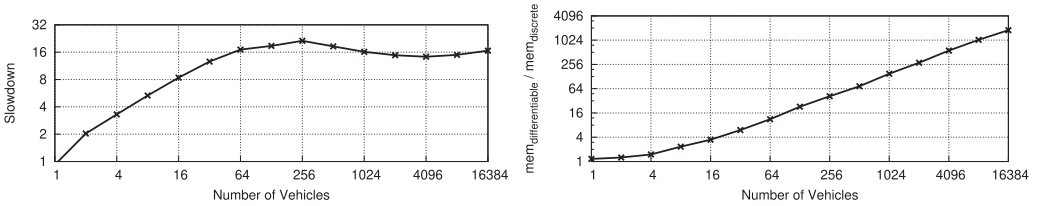
Fig. 7. Performance of the fully differentiable traffic simulation relative to the discrete reference simulation. The overhead increases strongly with the vehicle count.

spikes in the derivative. However, an additional negative spike at an offset of around 7.5 s illustrates that the derivative represents the slope only in a given point of the simulation input and, thus, is sensitive to implementation artifacts. Such artifacts occur when a real value that represents a Boolean or integer deviates too far from its reference value. A resulting minor deviation from the reference simulation can translate to a large derivative.

We measured the performance of the differentiable simulation compared with the basic reference implementation. The benefits of our approach depend critically on its overhead of the differentiable model implementation as compared with the discrete reference. An execution of the differentiable simulation returns the simulation output and its gradient at the current parameter combination. Sampling-based methods, such as traditional finite differences, can employ the substantially faster original discrete model. If the required series of simulation runs, which may be parallelized on supercomputing clusters, can be completed faster than a single run of the differentiable model, the sampling-based method is to be preferred.

Figure 7 shows the relative wall-clock time per run and the relative memory usage. The main overhead is induced by the simulation itself, during which the operations are recorded in preparation for the subsequent differentiation step. The contribution of the differentiation at 32 vehicles was about 170 s of a total of 789 s, or about 22%. The results demonstrate the enormous execution time overhead of this fully differentiable model implementation, which we address in the following by a selective substitution of model elements.

5.3.2 Grid Network with Static Signal Timings. We now evaluate the traffic model in which we restricted the differentiable aspects to the traffic light control and the vehicles' longitudinal movement. The grid network used in the experiments is composed of three-lane roads 100 m in length, with a speed limit of 35 km/h. At an intersection, a vehicle advances to the road on the left, right, or straight ahead with probabilities 0.05, 0.05, and 0.9.



(a) Slowdown.

(b) Relative memory usage.

Fig. 8. Performance of the traffic simulation on a grid network relative to a non-differentiable reference implementation. The overhead is substantially smaller compared with the fully differentiable model (see Figure 7).

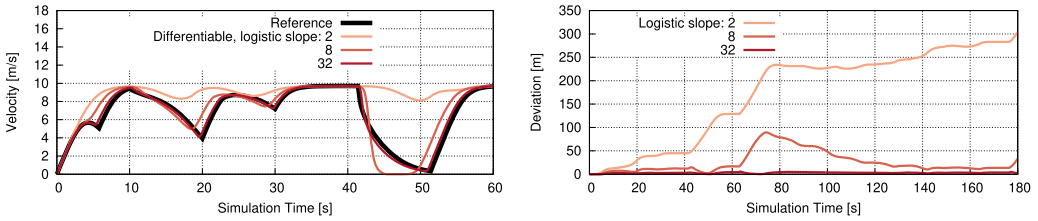
Figure 8 shows the performance comparison with the non-differentiable reference simulation on a grid composed of 50×50 intersections spanning 180 s of simulation time, using a slope parameter for the logistic function of 32. The traffic lights alternate between red and green phases of 10 s each. While there is still a substantial overhead in memory usage and execution time, the overhead is significantly lower than with the fully differentiable model. In particular, the execution time factor is now only weakly affected by the number of vehicles. To give an indication of the potential benefits of our approach, we can compare the time required to compute the gradient in one point to traditional forward finite differences. Forward finite differences estimates the gradient based on varying each of the 2 500 model parameters of the discrete model. Given that the differentiable simulation is slower than the discrete model by a factor of about 16, one gradient computation using our approach is roughly $\frac{2\,500}{16} = 156.25$ times faster than forward finite differences. We note that schemes for dimensionality reduction [29] and the parallel execution of simulation runs using additional hardware resources may reduce or eliminate this gap in performance. Our evaluation on overall optimization includes SPSA as an approximate scheme based on finite differences.

In contrast to the execution time, the relative memory consumption still increases with the number of vehicles as the results of increasing numbers of intermediate computational operations have to be stored to permit the subsequent gradient computation. Still, with an execution time of 2.7 s and a memory usage of 1.6 GiB with 1 024 vehicles, and 47.2 s using 23.2 GiB with 16 384 vehicles, we consider this model to be sufficiently scalable to cover scenarios of practical relevance.

We evaluate the fidelity of the differentiable simulation compared with the non-differentiable reference simulation using a scenario in which a single vehicle traverses the road network. In Figure 9(a), we plot the vehicle's velocity over time, which reflects the acceleration and deceleration behavior depending on the states of the traffic lights encountered on its route. As expected, the adherence to the velocities from the reference simulation improves when increasing the slope parameter of the logistic function. With a slope of 32, the median absolute deviation in velocity throughout the simulation run spanning 180 s of simulation time was 0.12 m/s, with 95% and 99% quantiles of 0.88 m/s and 1.39 m/s, respectively.

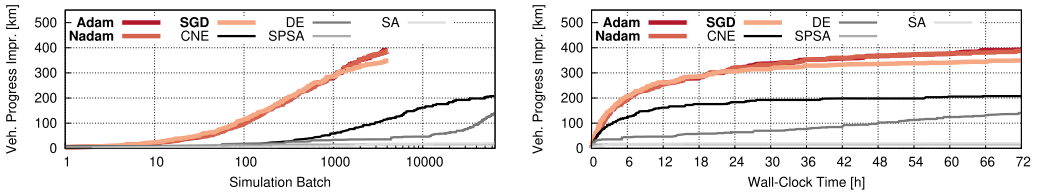
Figure 9(b) shows the absolute deviation of the distance driven over time from the reference simulation. With the slope parameter set to 2 and 8, the deviation accumulates to substantial distances, whereas a slope of 32 leads to a maximum deviation of only 5.7 m over the course of 180 s of simulation time.

We now turn to the comparison of the gradient-based and gradient-free simulation-based optimization. Our goal is to maximize vehicle progress by adjusting traffic light timings. To limit the number of optimization runs for evaluation, we adjust only two of the optimizers' own parameters and use the default values configured in the ensmallen library for the remaining parameters: (1) The step size (or its equivalent) is set to the values $\{10^{-3}, 10^{-2}, \ldots, 10^{0}\}$ and (2) for



(a) Velocity. (b) Distance driven.

Fig. 9. Fidelity of the differentiable traffic simulation on a grid compared to the discrete reference implementation when varying the slope parameter of the logistic function. The deviation diminishes when increasing the slope.



(a) Best observed solution across batches of 10 runs each. (b) Best observed solution across wall-clock time.

Fig. 10. Optimization progress for the 50 × 50 grid scenario using static traffic light control with a period of 20 s.

DE and CNE, which combine the current best solutions only after completing a so-called generation of runs, we reduce the generation size from its default of 500 to 50. Each optimization process starts from the same randomly initialized parameter combination. For each optimizer, we show the improvement over the initial parameter combination for the step size that achieved the maximum value within the time budget of 72 hours.

Figure 10(a) shows the optimization progress as the improvement over the initial random parametrization for a 50 × 50 grid populated with 2 500 vehicles, with an overall traffic light control period of 20 s. The total number of parameters to be adjusted is 2 500. Each point in the parameter space is evaluated by executing a batch of 10 simulation runs, averaging the returned output values and, for the differentiable simulation, the gradients. To avoid an excessive impact of large individual derivatives, we employ gradient clipping [63], restricting the derivatives to the arbitrarily chosen interval $[-10, 10]$. The same initial vehicle positions are used in all runs, randomizing only the turns at intersections, to introduce sufficient regularity in the traffic patterns to permit an optimization of the traffic light timings. Due to the overhead of the differentiable simulation, the gradient-based methods executed only about 4 000 simulation batches within the time budget, compared with about 64 000 with the gradient-free methods. However, the optimization progress per batch is immensely faster than with the gradient-free methods. For instance, after 100 batches, all gradient-based methods achieve an improvement of about 100 km, whereas the improvement achieved by any of the gradient-free methods was still below 20 km.

Again, we also consider the progress across wall-clock time as plotted in Figure 10(b). Due to the faster execution of the non-differentiable reference simulation, the initial benefit of the gradient-based optimization is somewhat reduced. Still, the best solution quality up to any given point in time is still vastly superior with the gradient-based methods.

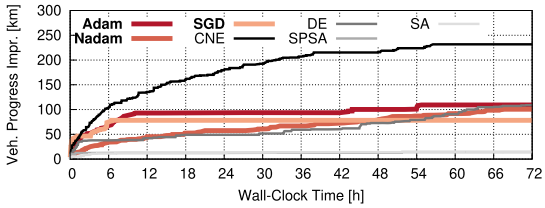
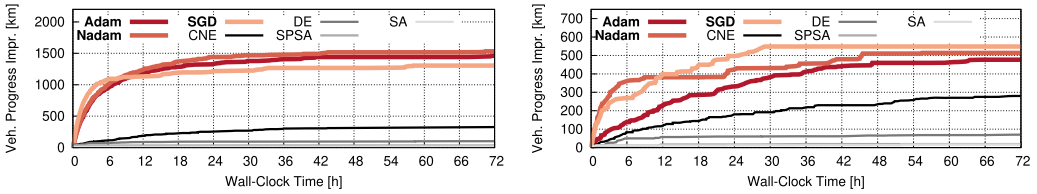


Fig. 11. Optimization progress with 50 × 50 intersections and a traffic light control period of 40 s.



(a) Traffic light control period of 20 s.

(b) Traffic light control period of 40 s.

Fig. 12. Optimization progress with 100 × 100 intersections.

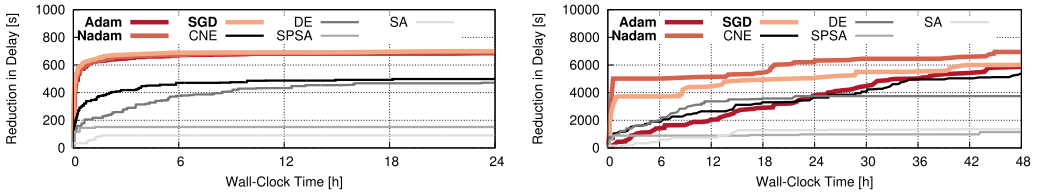
To show the validity of the comparison between the gradient-based and gradient-free optimization results, we used the highest-quality solution returned by Adam as input to the non-differentiable reference simulation, executing 100 runs. The mean overall vehicle progress includes the progress in the initial solution, with 95% confidence intervals of 3 467.2 ± 0.8 km in the differentiable simulation and 3 375.1 ± 0.9 km in the non-differentiable simulation. For comparison, the best solution found using CNE translated to only 3 175.2 ± 0.9 km of vehicle progress.

We repeated the experiment after increasing the traffic light period from 20 s to 40 s. In Figure 11, we see that in this configuration, the gradient-based methods are outperformed by CNE, achieving a similar solution quality to that of DE. A likely reason is given by the *periodic step function*: with a longer light period, there is a larger probability of generating very small gradients, which pose a challenge to the gradient-based methods [41] (see Section 6).

The experiment was repeated with 100 × 100 intersections and 10 000 vehicles, which increases the number of parameters to 10 000 at the same vehicle density. Figures 12(a) and 12(b) show that the advantage of the gradient-based methods is more pronounced, with consistently better solution quality compared with the gradient-free methods.

Finally, we carried out an optimization with an alternative simulation output. Now, instead of the overall vehicle progress, we aim to minimize the cumulative delay at intersections summed across the vehicles. We define the delay as the time spent at velocities below $0.1\frac{m}{s}$. Figure 13 shows the reduction in delay compared with the initial random parametrization over wall-clock time. Overall, the results are similar to the previous experiments, with the gradient-based methods exhibiting a faster increase in solution quality than the gradient-free methods.

5.3.3 Grid Network with Neural Network–Controlled Signals. An optimization experiment similar to that described earlier was carried out on a 5 × 5 grid populated by 200 vehicles, introducing dynamic traffic light control by a neural network. Given that each decision of the traffic light control remains in effect for 20 s, an optimal policy would consider not only the vehicle positions at each intersection but would also include the positions and traffic light phase at the neighboring intersections. The number of neurons in the hidden layer was set to 60, resulting in 91 585 neural network coefficients forming the simulation input. All optimization methods started from the same



(a) 500 vehicles. (b) 2,500 vehicles.

Fig. 13. Optimization progress for the 50 × 50 grid scenario using static traffic light control with a period of 20 s, using the cumulative delay as the simulation output to be minimized.

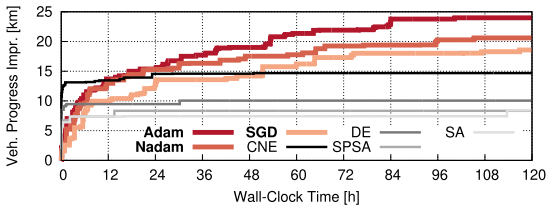


Fig. 14. Optimization progress for the grid traffic scenario with neural network–controlled traffic lights.

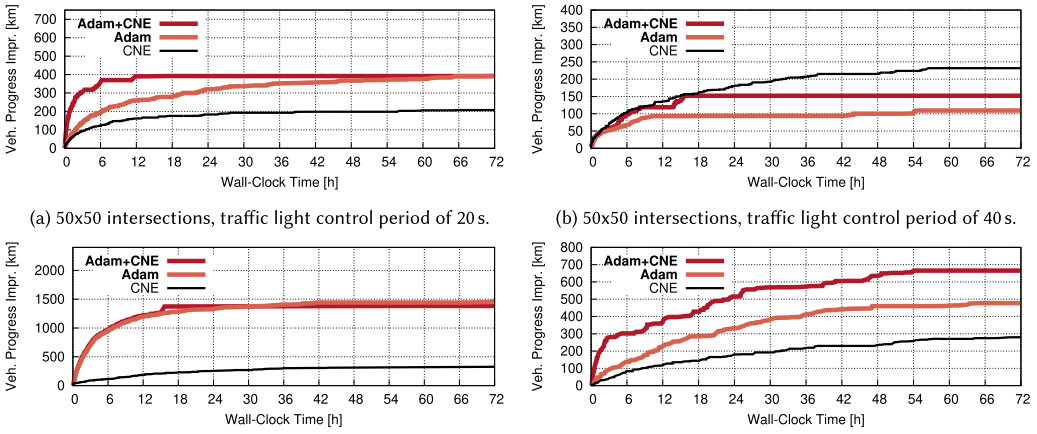
initial parameter combination drawn from a standard normal distribution. After preliminary experiments, we accelerated the optimization progress by randomizing the initial vehicle positions for each run and by modifying the simulation output to be the minimum progress among the vehicles instead of the sum progress.

Figure 14 shows that for this problem, the gradient-free methods achieve somewhat faster initial progress. However, beyond about 12 h, Adam and Nadam outperform all gradient-free methods. At the end of the time budget of 120 h, the highest overall vehicle progress achieved by Adam, as measured in the non-differentiable simulation, was 47.8 ± 0.7 km. The best result among the gradient-free methods achieved by CNE was 31.3 ± 0.9 km. Given 91 585 inputs, the cost of estimating gradients using finite differences is again prohibitive.

5.4 Combining Local and Global Search

In the previous experiments, we have seen that gradient-based optimization outperforms gradient-free methods in many cases. However, gradient-based methods carry out a *local* search, that is, a search for a local optimum starting from an initial parameter combination. Although the stochasticity of the considered model, a sufficiently large learning rate, or heuristics such as momentum make it possible to escape local minima in some cases, the optimization process may still explore areas of the simulation response surface far from the global optimum and become stuck near low-quality local optima. This issue is particularly pressing if the simulation response surface is non-smooth, which depends on properties of the simulation model itself as well as the chosen simulation output function. A natural approach is to combine gradient-based local search with gradient-free global search methods, for example, genetic algorithms, which yielded promising results in other problem domains [37, 67]. By alternating local and global search, the local search is supplied with multiple starting points in the parameter space from which a local optimum can be identified.

We repeated the experiments from Section 5.3.2 using a combination of gradient-based and gradient-free optimization. To this end, we combined the gradient-based local search using Adam with a global search using CNE, which in the previous experiments achieved by far the best



(a) 50x50 intersections, traffic light control period of 20 s.

(b) 50x50 intersections, traffic light control period of 40 s.

(c) 100x100 intersections, traffic light control period of 20 s.

(d) 100x100 intersections, traffic light control period of 40 s.

Fig. 15. Progress of traffic light timing optimization when combining gradient-based local search and gradient-free global search.

solution quality out of the gradient-free methods. For CNE, we used a fixed step size of 1, which performed best in all of the previous experiments. As previously, we set the step size for Adam to each value in $\{10^{-3}, 10^{-2}, \ldots, 10^{0}\}$ and show the results for the best-performing step size. During the optimization, we alternate between gradient-based and gradient-free search, switching methods whenever the current method fails to improve on the best solution for 30 minutes. The search then resumes using the current best solution.

Figure 15 shows the optimization progress for the different parameterizations of the traffic simulation using static signal timings on a grid, comparing the combined search using Adam and CNE to a search using only Adam or CNE in isolation. A slight reduction in solution quality compared with Adam alone is observed only for 100×100 intersections and a traffic light control period of 20 s. In all other cases, the combined search achieved higher performance than Adam on its own. The largest relative increase in solution quality was observed for 100×100 intersections with a traffic light control period of 40 s. When running the discrete reference simulation using the best solutions identified using the combined search and Adam alone, the overall vehicle progress was 11.03×10^6 m and 10.80×10^6m, respectively. For 50×50 intersections with a traffic control period of 40 s, CNE outperformed the combined search as in the previous experiments, which was not able to identify higher-quality solutions beyond the first 14 h.

We conclude that the combined search can further improve the quality of the identified solutions. However, it is still possible for the combined search to get caught near local optima.

5.5 Exploiting Sparsity

In Section 5.3, we studied the execution time and memory consumption of differentiable simulations. By restricting the differentiable model elements, memory consumption was reduced sufficiently to support large-scale simulations. However, increased execution times and gradual growth of the computational graph stored for subsequent reverse-mode differentiation still limits the maximum feasible simulation duration and scale. Here, we attempt to further reduce memory consumption based on the observation that actions in agent-based simulations are frequently sparse, that is, not all actions are carried out by all agents at each timestep. We recall from Section 3.1 that a conditional branch such as `if (x >= 0) y = 1;` is represented in the differentiable simulation as

`double z = logistic(x, 0); y = z * 1 + (1 - z) * y;`. In the differentiable simulation, these arithmetic operations are executed and stored in the computational graph even if $x \ll 0$, although this implies that z is very close to 0 and the effects of the operations both on the simulation output and its derivatives are likely to be miniscule.

To avoid such branches entirely even in the differentiable simulation, we introduce a *branch threshold* t_b. The differentiable operations are guarded using non-differentiable branches to skip them entirely if the value returned by the logistic function is below t_b. We implemented this approach for the traffic simulation and the epidemics simulation in order to answer two questions:

(1) *How is the simulation execution time and memory consumption affected by different branch thresholds?*
(2) *Do the resulting gradients still hold sufficient information to support simulation-based optimization?*

For the epidemics model, both the execution time and memory consumption decrease severely when increasing the branch threshold. With a branch threshold of 10^{-1}, the execution time and memory consumption are now higher by factors of only 4.3 and 8.7 compared with those of the non-differentiable reference model. This corresponds to reductions in execution time and memory consumption by factors of 4.7 and 3.6 compared with the original differentiable model.

We now consider the traffic simulation on a grid of 50×50 intersections populated by 2 500 agents. In this model, we previously severely restricted the differentiable model elements (see Section 4.3.2), which reduced the sparsity in the model logic. Thus, the additional effect of introducing a branch threshold is minor: we observe a speedup by about 17% and a reduction in memory consumption by only 8%.

We also studied the effects of introducing the branch threshold on the optimization progress. We observed that the optimization progress per simulation batch remains largely unchanged. This indicates that the introduction of the branch threshold has not substantially affected the utility of the computed derivatives for the gradient-based optimization. Notably, the observation holds even for a branch threshold as high as 0.1. Due to the decreased simulation execution times, Adam was able to converge faster with larger branch thresholds, keeping pace with CNE at a branch threshold of 10^{-1}. As in Section 5.2, CNE converges to solutions of slightly higher quality than Adam, with about 0.7% deviation from the reference run for CNE, and 2.3% to 2.4% for Adam.

In the traffic light timing optimization, we do not observe an improvement over the results from Section 5.3. The total number of executed batches within the time budget varied from 3 939 without a branch threshold to 4 565 with a branch threshold of 0.5. However, we observe that the resulting solution quality remains virtually identical independently of the branch threshold, that is, the computed derivatives were again able to support the gradient-based optimization even if large branch thresholds were configured.

From these results, we conclude that the exploitation of sparsity in the agent actions can be beneficial both in terms of sheer execution times and memory usage and in terms of the optimization progress. However, the effects are highly dependent on the amount of sparsity present in the differentiable model.

6 LIMITATIONS AND RESEARCH DIRECTIONS

Our experiments show the benefit of differentiable agent-based simulation in several optimization problems. Still, a number of avenues remain to be explored, the main focal points being the fidelity and performance of the differentiable models and the applicability of the approach by model domain experts and in the context of machine learning.

6.1 Fidelity

Most of the presented building blocks rely on approximations using the logistic function, the error being adjusted by a slope parameter. The configuration of the slope involves a trade-off: with steep slopes, the logistic function closely approximates a step function. However, negative or positive arguments with sufficiently large magnitude quickly approach 0 or 1, respectively. Similar to the vanishing gradient problem in machine learning [41], the resulting small gradients may lead to a sluggish optimization. On the other hand, with shallow slopes, arguments close to zero yield large deviations from the original step function. An important direction for future work lies in determining model-specific error bounds based on known bounds for the building blocks (e.g., [56]), and on the detection of artifacts.

6.2 Performance

We have seen that the overhead of the differentiable model variants is substantial, limiting their applicability for large scenarios. One of the causes is the implementation of branching: in effect, the operations of all possible branches are executed. While we showed that the selective use of differentiable model elements can limit the overhead, a challenge lies in identifying the model aspects for which gradient information is required. For instance, in the scalable traffic model variants, the sensitivity of lane-changing decisions to the model input is not captured in the computed gradients.

If an optimization targets the steady-state behavior of a model, some overhead could be avoided by first executing a fast non-differentiable implementation. Once a steady state has been reached, the simulation state is used to initialize a differentiable implementation that computes the output and gradient.

Memory consumption may potentially be reduced by forming so-called super-nodes [58]: first, groups of operations are identified that are repeatedly executed. Then, by manually defining an operation that represents the contribution of an entire group to the partial derivatives, the gradient computation is simplified. In agent-based simulations, sequences of operations executed for every agent and at every timestep may be candidates for super-nodes.

6.3 Applicability

The models presented in Section 4 were implemented manually, which, despite the relative simplicity of the considered models, proved to be somewhat cumbersome and error prone. Automatic translations could support more convenient development processes. Recent efforts aim to define differentiable general-purpose programming languages (e.g., [72]). Domain-specific languages defined in a similar vein could cater to agent-based modeling experts and facilitate the generation of optimized model code, for example, by exploiting sparsity as shown in Section 5.5.

Some recent research aims at integrating differentiable programming facilities into machine learning frameworks such as PyTorch [51]. Implementing differentiable agent-based models within such frameworks would enable a natural and efficient unification of simulation-based optimization and neural network training, making use of the frameworks' optimized GPU-based implementations of neural networks and automatic differentiation while accelerating the model execution through fine-grained manycore parallelism [79]. We discuss recent work towards this goal in Section 7.5.

7 RELATED WORK

Existing work in a several fields has considered the approximate execution of programs for a number of different purposes. In the following, we contrast these works with ours and differentiate our approach from existing methods to compute gradients of simulations.

7.1 Approximate Computing

Approximate computing techniques carry out computations at reduced fidelity, for example, by scaling the numerical precision or by relying on neural network–based function approximation [60]. Often, the intention is to reduce computational cost, to increase resilience to errors, or to solve problems for which an exact solution is not known. In contrast to these aims, the goal of our approximations is to allow automatic differentiation to extract gradient information. In the context of machine learning, there is currently intensive research towards approximate differentiable algorithms for problems such as path finding [76] and sorting [12, 23, 36]. In future work, we plan to build on such approximate algorithms to express increasingly complex agent behavior in a differentiable manner.

7.2 Sensitivity Analysis and Uncertainty Quantification

Sensitivity analysis is "the study of how uncertainty in the output of a model (numerical or otherwise) can be apportioned to different sources of uncertainty in the model input" [70]. In the context of optimization, sensitivity analysis is used to determine the effects of changes in the model parameters on the quality of the solution represented by the model output [17]. Automatic differentiation (see Section 2) enables a local first-order sensitivity analysis by calculating the partial derivatives at a given combination of model parameters. This is in contrast to global sensitivity analysis methods, which explore the model response at several points in the parameter space. Our approach employs automatic differentiation to determine local sensitivities to a given combination of model parameters but employs smoothing to account for effects of control flow branches not visited at the current parameter combination.

Closely related to sensitivity analysis is the field of uncertainty quantification. Methods for forward propagation of uncertainty are concerned with quantifying the uncertainty in a system's output given uncertainty in the parameters [45]. A simple form of forward propagation of uncertainty is given by differential error analysis, which calculates the variance of the model output based on its derivative with respect to all inputs and the inputs' own variances [9]. When targeting computer programs, the partial derivatives used in differential error analysis can be calculated via automatic differentiation. The literature distinguishes uncertainty quantification methods based on their intrusiveness: non-intrusive methods evaluate the original model at various points in the parameter space, whereas intrusive methods require modifications to the model, for example, the substitution of model components by surrogates. Our approach shares with intrusive uncertainty quantification methods its reliance on model adaptations. As described in the next section, our future work will explore the use of smooth interpretation, which is a form of approximative uncertainty quantification with the goal of facilitating optimization.

7.3 Smooth Interpretation

Smooth interpretation [18] is a method that aims to make general programs more amenable to numerical optimization by smoothing across discontinuities. The program input is supplied in the form of Gaussian random variables and propagated through a symbolic execution of the program, approximating the resulting complex distributions by combinations of Gaussian distributions based on rules defined in a smoothed semantics. Hence, smooth interpretation is a form of uncertainty quantification. To limit the overhead of the approach, all intermediate distributions are approximated using Gaussian mixture distributions. For instance, after modifying a variable v in the body of a conditional branch, the mixture distribution of v reflects the probabilities of taking or bypassing the branch together with v's respective expectation and variance. The

approach constitutes a potential alternative to our bottom-up construction of differentiable model implementations based on a set of smooth building blocks. Smooth interpretation can also be regarded as a special case of probabilistic programming, in which the probability distribution defined by a probabilistic program is computed automatically [33]. However, in contrast to most probabilistic programming approaches, the Gaussian approximations used by smooth interpretation avoid the need to sample from the input or intermediate distributions. We plan to explore the overhead of smooth interpretation; its ability to accurately capture the logic of agent-based models is part of our future work. By propagating variances assigned to the input variables throughout the simulation, smooth interpretation would provide a justification for the degree of smoothing applied at each discontinuity of the model and would permit a clear interpretation of the smoothed simulation outputs as approximate expectations.

7.4 Infinitesimal Perturbation Analysis

In Infinitesimal Perturbation Analysis (IPA) [40], gradient expressions are derived through an analysis of the given model. In contrast to our approach, IPA focuses on discrete-event simulations. While IPA can yield computations similar to those carried out by automatic differentiation, the IPA literature derives model-specific gradient estimators manually (e.g., [14, 19, 31, 44]), which is limited to relatively simple models. In contrast, automatic differentiation allows gradients to be computed directly from model implementations in general-purpose programming languages. A key concern in the literature on IPA is the bias introduced when aggregating derivatives from multiple runs of a stochastic simulation in the presence of discontinuities. Smoothed IPA [32] is a method to produce unbiased gradient estimations in the presence of discontinuities. By expressing the derivatives using conditional expectations, discontinuities can be eliminated for some simulations. However, Smoothed IPA requires a custom and manual analysis of each given simulation model. Our use of automatic differentiation allows gradients to be computed without manually determining gradient expressions. We eliminate discontinuities by the use of smooth approximations. Although we studied the fidelity of the computed gradients empirically in Section 5.1, we defer a closer analysis of the relationship between the smoothing and bias in the aggregated derivatives to future work.

7.5 Computing Gradients of Agent-Based Simulations

Considering existing research related to agent-based simulation, a continuous approximation of cellular automata (CAs) was proposed to enable gradient-based search for CAs with desired properties [59]. As in IPA, expressions for the partial derivatives are determined manually.

A recent preprint proposes the use of automatic differentiation to calibrate a variant of the Social Force model for pedestrian dynamics [38] against real-world data [55]. Their model variant represents the forces among the simulated agents using multi-layer perceptrons, which act as universal function approximators. Since the Social Force model is specified with respect to continuous time and space, it is a natural candidate for automatic differentiation.

In another recent preprint, a differentiable epidemics model implemented in a machine learning framework is presented [20]. The propagation of infections among the agents' interaction graph is carried out using a convolutional graph neural network [53]. In contrast, our own epidemics model represents the interaction graph as a non-differentiable model element, that is, the graph topology is reflected in the computed gradient only through the resulting agent interactions. However, their work does not study the computed gradients or report results from optimization experiments. As part of our future work, we are planning to explore the benefits of accounting for the graph topology as part of the gradient computation in simulation-based optimization problems.

These related works share our goal of enabling gradient-based optimization. However, they are specific to the considered models and do not provide more general sets of building blocks to construct differentiable agent-based simulations.

8 CONCLUSIONS

Simulation-based optimization of agent-based models with large numbers of inputs is usually carried out either on surrogate models, which typically abandon the individual-based level of detail of an original model, or using gradient-free methods such as genetic algorithms. To enable direct gradient-based optimization of agent-based models, we proposed the construction of differentiable implementations using smooth building blocks, enabling an automatic computation of the partial derivatives reflecting the sensitivity of the simulation output to the inputs.

To explore the merit of this novel approach to extracting gradients from agent-based simulations, our focus was on the question of whether gradient-based optimization using the differentiable models can outperform gradient-free methods. By constructing models from combinations of differentiable and non-differentiable model elements, we achieved sufficient performance to tackle scenarios populated by thousands of agents. Comparing the relative solution quality and convergence speed of gradient-based and gradient-free methods in simulation-based optimization experiments, we observed that the gradient-based methods in fact achieved better results in several cases. Particularly vast margins were observed in problems with large input dimension, which indicates that the approach could extend the reach of simulation-based optimization using agent-based models to problems that could previously only be tackled via surrogate modeling at a reduced level of detail. As an additional benefit of the approach, we demonstrated that neural network–controlled simulation entities embedded into the differentiable model logic can efficiently be trained using gradient-based methods, with substantially superior results over gradient-free methods.

We studied two approaches to further improve the optimization progress. First, by alternating gradient-free and gradient-based simulation, the solution quality within a given time budget was increased. Second, we exploited sparsity in the model logic by avoiding control flow paths with only a minor contribution to the simulation output and gradients. The execution time and memory usage of the epidemics simulation were substantially decreased without affecting the utility of the computed gradients.

Promising research directions lie in automated methods to reduce the overhead of differentiable simulations, in providing language support for expressing differentiable models in a natural way, and in model implementations targeting machine learning frameworks.

REFERENCES

[1] Mark S. Alber, Maria A. Kiskowski, James A. Glazier, and Yi Jiang. 2003. On cellular automaton approaches to modeling biological cells. In *Mathematical Systems Theory in Biology, Communications, Computation, and Finance*. Springer, 1–39.

[2] Philipp Andelfinger. 2021. Differentiable agent-based simulation for gradient-guided simulation-based optimization. In *Proceedings of the 2021 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ACM, 27–38.

[3] 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 *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ACM, Chicago, IL, USA, 173–184.

[4] Philipp Andelfinger, Yadong Xu, David Eckhoff, Wentong Cai, and Alois Knoll. 2020. Fidelity and performance of state fast-forwarding in microscopic traffic simulations. *ACM Transactions on Modeling and Computer Simulation* 30, 2, Article 10 (April 2020), 26 pages.

[5] Rushil Anirudh, Jayaraman J. Thiagarajan, Peer-Timo Bremer, Timothy C. Germann, Sara Y. Del Valle, and Frederick H. Streitz. 2020. Accurate calibration of agent-based epidemiological models with neural network surrogates. *arXiv preprint arXiv:2010.06558* (2020).

[6] Russell R. Barton. 2020. Tutorial: Metamodeling for simulation. In *2020 Winter Simulation Conference (WSC'20)*. IEEE, 1102–1116.
[7] Atılım Günes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. 2017. Automatic differentiation in machine learning: A survey. *The Journal of Machine Learning Research* 18, 1 (2017), 5595–5637.
[8] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. 2011. SUMO–simulation of urban mobility: An overview. In *Proceedings of SIMUL 2011, The 3rd International Conference on Advances in System Simulation*. ThinkMind, Barcelona, Spain, 55–60.
[9] K. K. Benke, S. Norng, N. J. Robinson, L. R. Benke, and T. J. Peterson. 2018. Error propagation in computer models: Analytic approaches, advantages, disadvantages and constraints. *Stochastic Environmental Research and Risk Assessment* 32, 10 (2018), 2971–2985.
[10] Shalabh Bhatnagar, Michael C. Fu, Steven I. Marcus, and I.-Jeng Wang. 2003. Two-timescale simultaneous perturbation stochastic approximation using deterministic perturbation sequences. *ACM Transactions on Modeling and Computer Simulation* 13, 2 (2003), 180–209.
[11] Atharv Bhosekar and Marianthi Ierapetritou. 2018. Advances in surrogate based modeling, feasibility analysis, and optimization: A review. *Computers & Chemical Engineering* 108 (2018), 250–267.
[12] Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. 2020. Fast differentiable sorting and ranking. In *International Conference on Machine Learning*. PMLR, 950–959.
[13] Eric Bonabeau. 2002. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences* 99, Suppl 3 (2002), 7280–7287.
[14] Pierre Brémaud and W.-B. Gong. 1993. Derivatives of likelihood ratios and smoothed perturbation analysis for the routing problem. *ACM Transactions on Modeling and Computer Simulation* 3, 2 (1993), 134–161.
[15] Benoît Calvez and Guillaume Hutzler. 2005. Automatic tuning of agent-based models using genetic algorithms. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*. Springer, Utrecht, The Netherlands, 41–57.
[16] Yolanda Carson and Anu Maria. 1997. Simulation optimization: Methods and applications. In *Proceedings of the 29th Winter Simulation Conference*. IEEE, Atlanta, GA, USA, 118–126.
[17] Enrique Castillo, Roberto Mínguez, and Carmen Castillo. 2008. Sensitivity analysis in optimization and reliability problems. *Reliability Engineering & System Safety* 93, 12 (2008), 1788–1800.
[18] Swarat Chaudhuri and Armando Solar-Lezama. 2010. Smooth interpretation. *ACM SIGPLAN Notices* 45, 6 (2010), 279–291.
[19] Min Chen, Jian-Qiang Hu, and Michael C. Fu. 2010. Perturbation analysis of a dynamic priority call center. *IEEE Transactions on Automatic Control* 55, 5 (2010), 1191–1196.
[20] Ayush Chopra, Ramesh Raskar, Jayakumar Subramanian, Balaji Krishnamurthy, Esma S. Gel, Santiago Romero-Brufau, Kalyan S. Pasupathy, and Thomas C. Kingsley. 2021. DeepABM: Scalable and efficient agent-based simulations via geometric learning frameworks—a case study for COVID-19 spread and interventions. In *2021 Winter Simulation Conference (WSC'21)*. IEEE, Phoenix, AZ, USA, 1–12.
[21] John D. Cook. 2011. Basic properties of the soft maximum. Working Paper Series 70, UT MD Anderson Cancer Center Dept. Biostatistics [Online]. Available http://biostats.bepress.com/mdandersonbiostat/paper70.
[22] Ryan R. Curtin, Marcus Edel, Rahul Ganesh Prabhu, Suryoday Basak, Zhihao Lou, and Conrad Sanderson. 2021. The ensmallen library for flexible numerical optimization. *Journal of Machine Learning Research* 22 (2021), 166–1.
[23] Marco Cuturi, Olivier Teboul, and Jean-Philippe Vert. 2019. Differentiable ranking and sorting using optimal transport. In *Advances in Neural Information Processing Systems*. PMLR, Vancouver, Canada, 6861–6871.
[24] Sina Dabiri and Montasir Abbas. 2016. Arterial traffic signal optimization using particle swarm optimization in an integrated VISSIM-MATLAB simulation environment. In *IEEE 19th International Conference on Intelligent Transportation Systems (ITSC'16)*. IEEE, Rio de Janeiro, Brazil, 766–771.
[25] Andreas Deckert and Robert Klein. 2014. Simulation-based optimization of an agent-based simulation. *NETNOMICS: Economic Research and Electronic Networking* 15, 1 (2014), 33–56.
[26] Armen Der Kiureghian and Ove Ditlevsen. 2009. Aleatory or epistemic? Does it matter? *Structural Safety* 31, 2 (2009), 105–112.
[27] Timothy Dozat. 2016. Incorporating Nesterov momentum into Adam. In *Proceedings of 4th International Conference on Learning Representations, Workshop Track*. OpenReview, San Juan, Puerto Rico, 1–4.
[28] J. David Eckman and Shane G. Henderson. 2020. Biased gradient estimators in simulation optimization. In *Proceedings of the Winter Simulation Conference*. IEEE, 2935–2946.
[29] Wouter Edeling, Hamid Arabnejad, Robbie Sinclair, Diana Suleimenova, Krishnakumar Gopalakrishnan, Bartosz Bosak, Derek Groen, Imran Mahmood, Daan Crommelin, and Peter V. Coveney. 2021. The impact of uncertainty on predictions of the CovidSim epidemiological code. *Nature Computational Science* 1, 2 (2021), 128–135.

[30] Martin Fellendorf and Peter Vortisch. 2010. Microscopic traffic flow simulator VISSIM. In *Fundamentals of Traffic Simulation*, Jaume Barceló (Ed.). Springer, 63–93.
[31] Yanfeng Geng and Christos G. Cassandras. 2012. Multi-intersection traffic light control using infinitesimal perturbation analysis. *11th IFAC Workshop on Discrete Event Systems*. IFAC, Guadalajara, Jalisco, Mexico, 104–109.
[32] Wei-Bo Gong and Yu-Chi Ho. 1987. Smoothed (conditional) perturbation analysis of discrete event dynamical systems. *IEEE Transactions on Automatic Control* 32, 10 (1987), 858–866.
[33] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. 2014. Probabilistic programming. In *Future of Software Engineering Proceedings*, Matthew B. Dwyer and James Herbsleb (Eds.). ACM, Hyderabad India, 167–181.
[34] Andreas Griewank, David Juedes, and Jean Utke. 1996. Algorithm 755: ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software* 22, 2 (1996), 131–167.
[35] Andreas Griewank and Andrea Walther. 2008. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM.
[36] Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. 2019. Stochastic optimization of sorting networks via continuous relaxations. In *International Conference on Learning Representations*. OpenReview, New Orleans, LA, USA, 1–23. https://openreview.net/forum?id=H1eSS3CcKX.
[37] Clemens Heitzinger and Siegfried Selberherr. 2002. An extensible TCAD optimization framework combining gradient based and genetic optimizers. *Microelectronics Journal* 33, 1-2 (2002), 61–68.
[38] Dirk Helbing and Peter Molnar. 1995. Social force model for pedestrian dynamics. *Physical Review E* 51, 5 (1995), 4282.
[39] Adrián Hernández and José M. Amigó. 2019. Differentiable programming and its applications to dynamical systems. *arXiv preprint arXiv:1912.08168* (2019).
[40] Yu-Chi Ho and Christos Cassandras. 1983. A new approach to the analysis of discrete event dynamic systems. *Automatica* 19, 2 (1983), 149–167.
[41] Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6, 02 (1998), 107–116.
[42] Robin J. Hogan. 2014. Fast reverse-mode automatic differentiation using expression templates in C++. *ACM Transactions on Mathematical Software* 40, 4 (2014), 1–16.
[43] L. Jeff Hong and Barry L. Nelson. 2009. A brief introduction to optimization via simulation. In *Proceedings of the 2009 Winter Simulation Conference (WSC'09)*. IEEE, Austin, TX, USA, 75–85.
[44] William Casey Howell. 2006. *Simulation Optimization of Traffic Light Signal Timings via Perturbation Analysis*. Ph.D. Dissertation. University of Maryland.
[45] Ronald L. Iman and Jon C. Helton. 1988. An investigation of uncertainty and sensitivity analysis techniques for computer models. *Risk Analysis* 8, 1 (1988), 71–90.
[46] Mike Innes, Alan Edelman, Keno Fischer, Chris Rackauckus, Elliot Saba, Viral B. Shah, and Will Tebbutt. 2019. Zygote: A differentiable programming system to bridge machine learning and scientific computing. *arXiv preprint arXiv:1907.07587* (2019), 140.
[47] June Young Jung, Gary Blau, Joseph F. Pekny, Gintaras V. Reklaitis, and David Eversdyk. 2004. A simulation based optimization approach to supply chain management under demand uncertainty. *Computers & Chemical Engineering* 28, 10 (2004), 2087–2106.
[48] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4 (1996), 237–285.
[49] William Ogilvy Kermack and Anderson G. McKendrick. 1927. A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A* 115, 772 (1927), 700–721.
[50] 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.
[51] Nikhil Ketkar. 2017. Introduction to PyTorch. In *Deep Learning with Python*. Springer, 195–208.
[52] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[53] Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations (ICLR'17)*. OpenReview, Toulon, France, 1–14.
[54] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220, 4598 (1983), 671–680.
[55] Sven Kreiss. 2021. Deep social force. *arXiv preprint arXiv:2109.12081* (2021).
[56] Nikolay Kyurkchiev and Svetoslav Markov. 2015. Sigmoid functions: Some approximation and modelling aspects. *LAP LAMBERT Academic Publishing, Saarbrücken* (2015).
[57] Charles M. Macal. 2010. To agent-based simulation from system dynamics. In *Proceedings of the 2010 Winter Simulation Conference*. IEEE, Baltimore, Maryland, USA, 371–382.

[58] Charles C. Margossian. 2019. A review of automatic differentiation and its efficient implementation. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 9, 4 (2019), e1305.
[59] Carlos Martin. 2017. Differentiable cellular automata. *arXiv preprint arXiv:1708.09546* (2017).
[60] Sparsh Mittal. 2016. A survey of techniques for approximate computing. *ACM Computing Surveys* 48, 4 (2016), 1–33.
[61] David J. Montana and Lawrence Davis. 1989. Training feedforward neural networks using genetic algorithms. In *International Joint Conference on Artificial Intelligence*, Vol. 89, Morgan Kaufmann Publishers Inc., Detroit, Michigan, USA, 762–767.
[62] Carolina Osorio and Linsen Chong. 2015. A computationally efficient simulation-based optimization algorithm for large-scale urban transportation problems. *Transportation Science* 49, 3 (2015), 623–636.
[63] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*. PMLR, Atlanta, Georgia, USA, 1310–1318.
[64] Raghu Pasupathy and Soumyadip Ghosh. 2013. Simulation optimization: A concise overview and implementation guide. *Theory Driven by Influential Applications* (2013), 122–150.
[65] Kalyan S. Perumalla and Richard M. Fujimoto. 1999. *Source-code Transformations for Efficient Reversibility*. Technical Report. Georgia Institute of Technology.
[66] An-Shen Qi, Xiang Zheng, Chan-Ying Du, and Bao-Sheng An. 1993. A cellular automaton model of cancerous growth. *Journal of Theoretical Biology* 161, 1 (1993), 1–12.
[67] M. E. Requena-Perez, A. Albero-Ortiz, J. Monzo-Cabrera, and A. Diaz-Morcillo. 2006. Combined use of genetic algorithms and gradient descent optimization methods for accurate inverse permittivity measurement. *IEEE Transactions on Microwave Theory and Techniques* 54, 2 (2006), 615–624.
[68] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).
[69] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (1986), 533–536.
[70] Andrea Saltelli. 2002. Sensitivity analysis for importance assessment. *Risk Analysis* 22, 3 (2002), 579–590.
[71] Junkichi Satsuma, R. Willox, A. Ramani, B. Grammaticos, and A. S. Carstea. 2004. Extending the SIR epidemic model. *Physica A: Statistical Mechanics and its Applications* 336, 3-4 (2004), 369–375.
[72] Amir Shaikhha, Andrew Fitzgibbon, Dimitrios Vytiniotis, and Simon Peyton Jones. 2019. Efficient differentiable programming in a functional array-processing language. *Proceedings of the ACM on Programming Languages* 3, ICFP, Article 97 (Jul 2019), 30 pages. https://doi.org/10.1145/3341701
[73] Emil I. Sluşanschi and Vlad Dumitrel. 2016. ADiJaC–automatic differentiation of Java classfiles. *ACM Transactions on Mathematical Software* 43, 2 (2016), 1–33.
[74] James C. Spall. 1992. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control* 37, 3 (1992), 332–341.
[75] Rainer Storn and Kenneth Price. 1995. *DE-A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Space*. Technical Report 6. 95–102.
[76] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. 2016. Value iteration networks. *Advances in Neural Information Processing Systems* 29 (2016), 2146–2154.
[77] Tommaso Toffoli. 1980. Reversible computing. In *International Colloquium on Automata, Languages, and Programming*, Jaco Bakker and Jan Leeuwen (Eds.). Springer, Noordwijkerhout, The Netherlands, 632–644.
[78] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. 2000. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E* 62, 2 (February 2000), 1805–1824.
[79] Jiajian Xiao, Philipp Andelfinger, David Eckhoff, Wentong Cai, and Alois Knoll. 2019. A survey on agent-based simulation using hardware accelerators. *ACM Computing Surveys* 51, 6 (2019), 1–35.
[80] 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.
[81] Milad Yousefi, Moslem Yousefi, and Flavio S. Fogliatto. 2020. Simulation-based optimization methods applied in hospital emergency departments: A systematic review. *Simulation* 96, 10 (2020), 791–806.
[82] Lihui Zhang, Yafeng Yin, and Shigang Chen. 2013. Robust signal timing optimization with environmental concerns. *Transportation Research Part C: Emerging Technologies* 29 (2013), 55–71.
[83] Jinghui Zhong, Nan Hu, Wentong Cai, Michael Lees, and Linbo Luo. 2015. Density-based evolutionary framework for crowd model calibration. *Journal of Computational Science* 6 (2015), 11–22.

Received 10 December 2021; revised 14 July 2022; accepted 27 September 2022