TraceVis: Towards Visualization for Deep Statistical Model Checking*

Timo P. Gros[2], David Groß[1], Stefan Gumhold[1], Jörg Hoffmann[2], Michaela Klauck[2], and Marcel Steinmetz[2]

[1] Technical University Dresden, Dresden, Germany
{david.gross1,stefan.gumhold}@tu-dresden.de
[2] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
{timopgros,hoffmann,klauck,steinmetz}@cs.uni-saarland.de

Abstract. With the proliferation of neural networks (NN), the need to analyze, and ideally verify, their behavior becomes more and more pressing. Significant progress has been made in the analysis of individual NN decision episodes, but the verification of NNs as part of larger systems remains a grand challenge. Deep statistical model checking (DSMC) is a recent approach addressing that challenge in the context of Markov decision processes (MDP) where a NN represents a policy taking action decisions. The NN determinizes the MDP, resulting in a Markov chain which is analyzed by statistical model checking. Initial results in a Racetrack case study (a simple abstract encoding of driving control) suggest that such a DSMC analysis can be useful for quality assurance in system approval or certification.
Here we explore the use of visualization to support DSMC users (human analysts, domain engineers). We implement an interactive visualization tool, TraceVis, for the Racetrack case study. The tool allows to explore crash probabilities into particular wall segments as a function of start position and velocity. It furthermore supports the in-depth examination of the policy traces generated by DSMC, in aggregate form as well as individually. This demonstrates how visualization can foster the effective analysis of DSMC results, and it forms a first step in combining model checking and visualization in the analysis of NN behavior.

Keywords: Statistical Model Checking · Neural Networks · Visualization.

1 Introduction

Neural networks (NN), in particular deep neural networks, have led to astounding advances in many areas of computer science [25,20,35]. NNs are more and more at the core of *intelligent systems*, taking decisions traditionally taken by humans.

For such systems, the need to analyze, and ideally verify, NN behavior becomes more and more pressing. This constitutes a grand challenge as it combines (1) the complexity of analyzing NN function representations with (2) the state space explosion problem (analyzing large system behavior state spaces). Remarkable progress is being made on (1), through SAT modulo theories [24,22,8], abstract interpretation

* Authors are listed alphabetically.

[12,28], and quantitative analysis [41,6]. This pertains to the verification of individual NN decision episodes, i.e., the behavior of a single input/output function call. Yet the verification of decision-taking NNs in intelligent systems requires the analysis of all possible situations that may result from sequences of NN decisions.

Many intelligent systems using NN, e.g., the control of various forms of cyber-physical systems, can be cast as discrete decision making in the presence of random phenomena. Hence a natural framework within which to start addressing the problem are Markov decision processes [34] (MDP), and specifically the model families considered in probabilistic model checking [27]. Assume a decision-making problem for which a NN has been trained, and assume that the problem can be formally cast as a MDP. Then we may use this MDP as a context to study properties of the NN. The NN is perceived as an action *policy* in the MDP, determinizing the non-deterministic choices. This yields a Markov chain which can be analyzed by probabilistic model checking techniques.

Recent work [14], henceforth referred to as *DSMC20*, proposed so-called *deep statistical model checking (DSMC)* as a scalable approach of this kind. The idea is to apply statistical model checking [42,19] to the Markov chain resulting from the use of a NN policy in an MDP. DSMC20 realize this idea in the context of MDPs represented in Jani [5], a language interfacing with leading probabilistic model checking tools. They implement a generic connection between NNs and the state-of-the-art statistical model checker modes [2,4], part of The Modest Toolset [18].

DSMC20 perform practical experiments in a Racetrack case study (adopted from benchmarks in AI autonomous decision making [1,33]), where a vehicle needs to choose accelerate/decelerate actions on a discrete map so as to reach a goal line without bumping into a wall. We adopt this case study here. While the problem is simple, it is suited as a starting point in the grand challenge of intelligent system verification. It can be readily extended to include traffic, sensing, fuel consumption, etc, ultimately up to models reflecting important challenges in autonomous driving.

DSMC20 propose DSMC as a tool for quality assurance by human analysts or domain engineers in system approval or certification. Clearly, given the complexity of problem parameter spaces and the need to understand what is going wrong and how, visualization methods are potentially very useful for that purpose. DSMC20 illustrate this with simple heat maps localizing safety issues. In the present paper, we begin to address the full scope of this visualization problem.

We design a new highly immersive visual exploration tool, that we baptize *TraceVis*, for the data space in DSMC on Racetrack. TraceVis exploits 3D visualization for mapping probabilities to height, stacking of trajectories and mapping of time. We develop a hierarchical navigation concept to avoid multiple views, such that all visualizations are integrated into a single 3D scene (which will ease a future extension to a virtual reality setup). Besides TraceVis itself, our contributions are:

- An interactive overview and context visualization of the goal and crash probabilities computed by DSMC, where we can inspect for all start positions p of the track either a single start velocity v or all start velocities at the same time.
- An aggregate view of all trajectories for a given start configuration (p, v) that visualizes the velocity distribution over the whole track in a concise and comprehensible way. Optionally, we allow disaggregation of time providing more details into the data space.

- An efficient hierarchical navigation approach, from an overview over the whole track to a trajectory ensemble for a selected start configuration, and over two levels of trajectory clusters down to individual trajectories.
- A replay mode that animates policy traces, which can be used in the stacked as well as in the aggregate trajectory visualization mode.
- A case study illustrating the analysis power of the new visual exploration tool in Racetrack.

Our endeavor differs from previous work on visualization techniques for NNs (e.g. [21,43,39]) in its focus on DSMC and Racetrack. We draw on established techniques in visualization, in particular in ensemble visualization (e.g. [37,30,9]).

Overall, we initiate a connection between DSMC and visualization research, laying the groundwork for long-term synergy between model checking and visualization in this context. In particular, we believe that some of the key ideas in TraceVis will carry over to more faithful representations of autonomous driving, and to other domains involving cyber-physical systems where position and velocity in physical space are key dimensions.

The paper is organized as follows. Section 2 discusses related work. Section 3 briefly summarizes DSMC20 as relevant to understand our work and contribution. Section 4 outlines our data space and visualization concept, followed by Sections 5 and 6 which describe our visualization techniques for crash probabilities and policy traces respectively. Section 7 exemplifies the use of TraceVis for DSMC result analysis in Racetrack. Section 8 closes the paper with an outlook on future challenges.

A video demonstrating TraceVis as well as its source code is available at DOI 10.5281/zenodo.3961196 [13]. [3]

2 Related Work

In the context of explainable AI research, a lot of recent research has been devoted to interactive visualization of NN [21]. Goals for such techniques include interpretability, explainability, NN debugging, as well as model comparison and selection. Most of the work has been dedicated to NNs for image analysis tasks. Only few recent works address the debugging and interpretation of deep networks used in reinforcement learning [43,39]. These are dedicated to Deep Q-Learning of agents playing Atari Retro Games, where high state-space dimensionality is the core problem addressed. In [43] the authors embed the state space based on the last layer of the NN with t-SNE into two dimensions and colorize the 2D points with handcrafted features. Their main contribution is an analysis of the MDP through spacetime clustering of the state space. The resulting hierarchical decomposition into skills allows for a better interpretation of the strategy of the learned agents. Wang et al. [39] developed a visual analysis tool with multiple coordinated views supporting a hierarchical navigation from an overview of the learning process down to individual traces of gameplays. Their main contribution is a scalable visualization of these traces that visualizes the position of the paddle together with the actions taken.

[3] For convenience, the video can also be played directly here https://cloudstore.zih.tu-dresden.de/index.php/s/Y7JmrZLfYEpkXPD.

One of our contributions pertains to the visual analysis of large collections of traces. So our work relates to ensemble visualization, which is an active research area. Wang et al. [40] survey sixty recent ensemble visualization papers and found that all visualization techniques are based on aggregation over ensemble members before the visualization, and on composition of ensemble members after the visualization. For the case of trace or trajectory ensembles, prominent aggregation techniques generalize 1D boxplots [38]. Mirzargar et al. [30] use the concept of data depth to define a band which encloses a given percentile of the curves in a curve ensemble. Due to the high computational complexity of curve boxplots, Etienne et al. [9] propose trajectory box plots, which are based on per frame oriented boxes that are fast to compute but introduce more visual clutter. With respect to composition of 2D trajectory ensembles, the stacking of the trajectories in 3D has shown to be a versatile solution [37]. Here, we develop a new aggregation technique specifically designed for Racetrack, and also support the stacking of trajectory ensembles.

3 Background: DSMC20 and Racetrack

This paper is a direct follow-up on DSMC20 [14], so we give the background in terms of a summary of that work, as relevant to understand our study and contribution.

Deep Statistical Model Checking (DSMC). The models considered in DSMC20, and here, are discrete-state MDPs. For any nonempty set S let $\mathcal{D}(S)$ denote the set of probability distributions over S.

Definition 1 (Markov Decision Process). *A Markov Decision Process (MDP) is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, s_0 \rangle$ consisting of a finite set of* states *$\mathcal{S}$, a finite set of* actions *$\mathcal{A}$, a partial* transition probability function *$\mathcal{T} \colon \mathcal{S} \times \mathcal{A} \nrightarrow \mathcal{D}(\mathcal{S})$, and an* initial state *$s_0 \in \mathcal{S}$. We say that action $a \in \mathcal{A}$ is* applicable *in state $s \in \mathcal{S}$ if $\mathcal{T}(s, a)$ is defined. We denote by $\mathcal{A}(s) \subseteq \mathcal{A}$ the set of actions applicable in s. We assume that $\mathcal{A}(s)$ is nonempty for each s (which is no restriction).*

An action policy resolves the non-deterministic choices in a state, determining which applicable action to apply as a function of the state history so far. We represent histories as finite sequences of states, hence elements of $\mathcal{S}^+$. We use $last(w)$ to denote the last state in $w \in \mathcal{S}^+$.

Definition 2 (Action Policy). *A (deterministic, history-dependent)* action policy *is a function $\sigma \colon \mathcal{S}^+ \to \mathcal{A}$ such that $\forall w \in \mathcal{S}^+ \colon \sigma(w) \in \mathcal{A}(last(w))$.*

An MDP together with an action policy defines a Markov chain:

Definition 3 (Markov Chain). *A Markov Chain is a tuple $\mathcal{C} = \langle \mathcal{S}, \mathcal{T}, s_0 \rangle$ consisting of a set of states $\mathcal{S}$, a transition probability function $\mathcal{T} \colon \mathcal{S} \to \mathcal{D}(\mathcal{S})$ and an initial state $s_0 \in \mathcal{S}$.*

Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, s_0 \rangle$, an action policy $\sigma \colon \mathcal{S}^+ \to \mathcal{A}$ induces a countable-state Markov chain $\langle \mathcal{S}^+, \mathcal{T}', s_0 \rangle$ over state histories in the obvious way: For any $w \in S^+$ with $\mathcal{T}(last(w), \sigma(w)) = \mu$, set $\mathcal{T}'(w) = \rho$ where $\rho(ws) = \mu(s)$ for all $s \in \mathcal{S}$.

The idea in DSMC is to analyze this Markov chain for an action policy represented as a neural network (NN). The NN is assumed to be trained externally prior to the DSMC analysis, but is assumed to operate on the same state space as a given MDP $\mathcal{M}$ (i.e., the NN's inputs are states and its outputs are actions). The NN policy σ together with $\mathcal{M}$ then induces a Markov chain $\mathcal{C}$ as described. Statistical model checking is a promising approach to analyze $\mathcal{C}$, as it merely requires to evaluate the NN on input states, otherwise treating it like a blackbox. DSMC20 implemented this approach for MODES [4] in THE MODEST TOOLSET [18].

Observe that, for DSMC to work in this form, the MDP and the NN need to operate on the same level of system abstraction. This is a simplification (relative to, e.g., NNs whose input are camera images) that renders the model checking problem crisp. Another subtlety is that the NN may return inapplicable actions (giving guarantees on NN outputs is notoriously hard), and in that sense may not actually fit the definition of an action policy. DSMC20 handle this through a more permissive definition of *action oracle*, transitioning to a new stalled state in the induced Markov chain $\mathcal{C}$ if the NN oracle's chosen action is inapplicable.

Racetrack Benchmark and JANI *Model.* Racetrack is originally a pen and paper game [10]. It was adopted as a benchmark for MDP algorithms in the AI community [1,3,29,32,33]. The track is a two-dimensional grid, where each cell of the grid can be a starting position, a goal position, a free position, or a wall. The vehicle starts with velocity 0 at any of the starting positions, and the objective is to reach the goal as fast as possible without crashing into a wall. The actions modify the velocity vector by one unit in the eight discrete directions; one can also choose to keep the current velocity. We consider noise emulating slippery road conditions: actions may *fail* with a given probability, in which case the velocity remains unchanged. Here we use two Racetrack benchmarks, i.e., track shapes, originally introduced by Barto et al. [1]. They are illustrated in Figure 1.

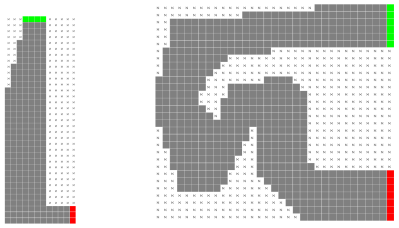

Fig. 1. The maps of our Racetrack benchmarks: Barto-small (left) and Barto-big (right). Starting positions green, goal positions red.

DSMC20 encode these Racetrack benchmarks in JANI [5,23]. Many tools offer direct support for JANI, including ePMC, Storm and THE MODEST TOOLSET [17,18,7]; an automatic translation from JANI to PRISM [26] is available too.

DSMC20's JANI model represents the grid as a two-dimensional array. Vehicle movements and collision checks are represented by separate automata that synchronize using shared actions. This is straightforward except for the collision checks, i.e., checking whether the vehicle's move – represented through horizontal and vertical speed (dx, dy) – hits a wall. This is done by generating a (discrete approximation

of) a straight line from the vehicle position (x, y) to $(x + dx, y + dy)$, and checking whether any position on that line contains a wall segment.

Neural Networks. NNs consist of neurons that apply a non-linear function to a weighted sum of their inputs. DSMC20 use feed-forward NNs, where neurons are arranged in a sequence from an input layer via several hidden layers to an output layer. So-called "deep" neural networks consist of many layers. Feed-forward NNs are comparatively simple, yet are wide-spread [11] (and anyway our visualization techniques are independent of the NN architecture).

To learn NN action policies in Racetrack, DSMC20 employ *deep Q-learning* [31], where the NN is trained by iterative execution and refinement steps. Each step executes the current policy until a terminal state is reached (goal or crash), and updates the NN weights using gradient descent. NNs are learnt for a specific map (cf. Figure 1). The NNs have two hidden layers each of size 64.

Case Study and Heat Maps. DSMC20 use Racetrack as a case study to highlight the use of DSMC for quality assurance. They use simple heat maps for a limited visualization of the DSMC outcome. Here we advance way beyond this, to interactive visualization of a much richer data space. To give the comparison to DSMC20, in what follows we briefly show a representative result from their case study.

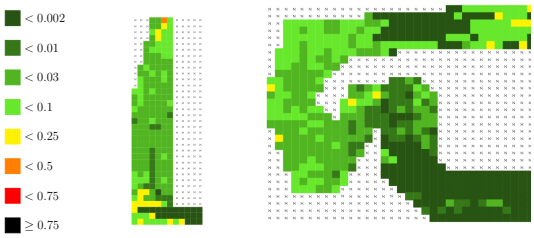

Fig. 2. DSMC20 heat maps, showing aggregated crash probability as a function of start position when fixing start velocity to 0.

Figure 2 shows aggregated crash probability – the probability of crashing into any wall – as a function of start position when fixing start velocity to 0. The heat maps use a simple color scheme as indicated in the figure. From this simple visualization, quality assurance analysts can conclude that the NN policies are fairly safe, to different degrees depending on the map region. What the heat maps don't show is, for example, how the shown probabilities depend on initial velocity, *where* unsuccessful policy runs tend to crash into the wall, and to what degree such crashes are due to noise or bad policy decisions. We show in what follows how to make all these details accessible through interactive visualization methods.

4 Visualization Concept

Before we go into the details of our visualization techniques, let us outline our concept in terms of the data space we visualize, and the principles behind our visualization approach.

Data Collection. We collect extensive information about the to-be-analyzed action policy from MODES, allowing to analyze policy behavior as a function of start position p *and* start velocity v, and showing not only whether the policy succeeded or crashed but also *where*. To this end, we run separate model checks with MODES for every pair (v, p), with properties representing every possible terminal (goal/crash) position. The number of runs is thus quadratic in map size, with a constant factor of 25 for the start velocities (in $\{-2, -1, 0, 1, 2\}^2$). We ignore start velocities that directly lead to a crash in the first step.

We furthermore collect all policy traces generated by MODES during DSMC, with detailed per-step information: position, velocity, action taken by policy, and a Boolean indicating whether the action succeeded or failed (i.e., whether noise occurred). In Barto-big, to keep computation times reasonable, we generated this data only for 7 of the 25 possible start velocities.

We want to highlight that the information about the policies we extracted from MODES are not specific to DSMC. We only used state and action information of the MDP under investigation. These trace information can be obtained with every statistical model checker independent of the mechanism used to resolve nondeterminism which in our case was DSMC.

Computation and export of this data for Barto-small/Barto-big took 17/20 hours on 25 virtual machines having an AMD EPYC Processor at approximately 2.5 GHz using Ubuntu 18.04 with 8vCPUs and 16GB RAM. The data comprises 5473/3826 start configurations consuming 1.25MB/1.18MB for probabilities and 15.3GB/13.4GB for traces/reduced traces on disk, in a concise text file format organized in two folders for probabilities and traces with one file per start configuration. The largest trace file has 13MB on disk and contains 18270 traces of average/max length 44/65. The data is publicly available at DOI 10.5281/zenodo.3961196 [13].

Visualization principles and rationales. Neither the probabilities nor the traces can be visualized in their entirety in a single visualization. We therefore opted for the development of a highly interactive visual analysis tool, TraceVis. As Racetrack is 2-dimensional, we chose a 3-dimensional visualization space to be able to exploit the 3rd dimension to map additional features. We implemented TraceVis as a plugin to the CGV-Framework [15], which allows rapid prototyping of interactive 3D tools in C++ with OpenGL. The CGV-Framework supports efficient high-quality rendering of large amounts of primitive shapes like boxes, spheres and rounded cones based on the concept of GPU based raycasting [16,36]. All primitives allow color mapping.

Figure 3 illustrates the design of TraceVis. For each track position we render a box whose type is color coded: start/goal locations in green/blue, walls in red, other track locations in light gray or color-mapped, and an additional row of dark grey boundary cells added around the track.

To keep the tool as clear as possible, we completely abstained from multiple views, incorporating all visualizations and interactions within a single 3D scene built on top of the Racetrack map. The view onto the 3D scene can be adjusted with the mouse based on an adjustable focus point with the typical navigation commands for translate, zoom and orbiting around the focus. Mode switches are used to navigate through different visualizations, and mouse pointer and wheel are used for direct selection and ergonomic configuration. All selections and configurations can also be adjusted through a classical user interface, which shows the current status of

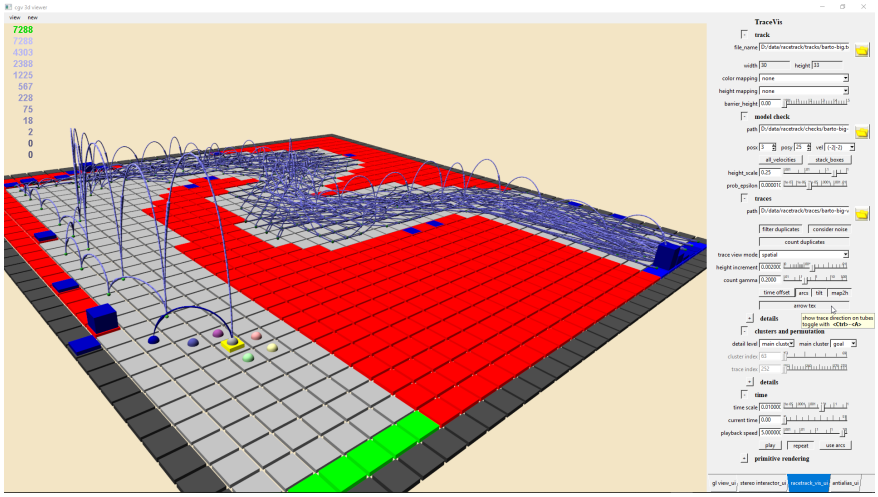

Fig. 3. Screenshot to illustrate the design of TraceVis. Highly interactive 3D view, accompanied by classical UI showing current tool state and providing tooltip based help.

TraceVis and serves as manual by providing help on mouse interaction and hotkeys through tooltips. For fast navigation and to foster comprehension, an important design goal was the support for high frame rates even when visualizing a large number of traces at the same time.

In accordance to Schneiderman's mantra – "Overview first, zoom and filter, then details-on-demand" – we designed a hierarchical navigation scheme. A heat-map visualization similar to the one in DSMC20 serves as overview over all start configurations. The user can select individual start configurations to view crash and goal probabilities (as described in Section 5). The user can dive into more detail by switching to trace visualization mode where the corresponding trace file is read on the fly; the traces can further be navigated from main clusters down to single traces (as described in Section 6). To navigate to a different start configuration, the user first needs to navigate back up the hierarchy to the probability visualization mode. This allows for the reuse of the same hotkeys on different hierarchy levels, reducing the number of hotkeys to be learned for fast interaction.

5 Visualizing Probabilities

We next describe our techniques for visualizing crash/goal probabilities as a function of start position p and start velocity v.

Start Configuration Selector. DSMC20 provides for each start configuration (p, v) and each wall, boundary and goal location q the probability that traces from (p, v) end in q. Visualizing the entire probability mapping $P(p, v, q)$ in a single image or 3D-scene seems futile. Our approach is to instead leverage interactive visualization, based on selection and aggregation of arguments to $P(p, v, q)$. TraceVis supports selection of a single p and/or a single v at a time. We indicate these user-fixed parameters notationally as $\hat{p}$ and $\hat{v}$.

The user can interactively select $\hat{p}$ by hovering with the mouse over the track while pressing Shift. $\hat{v}$ can be selected by additionally holding the Ctrl modifier key

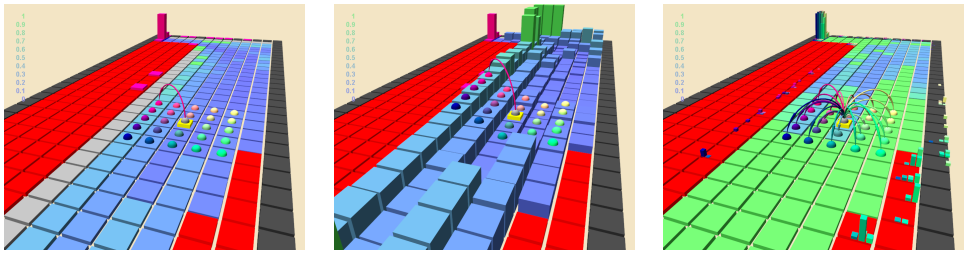

Fig. 4. Start configuration selection and different probability visualization approaches. Left: selected start configuration $(\hat{p}, \hat{v})$ shown as yellow box and pink arrow. Summed crash probabilities $\sum_{\tilde{q}} P(p, \hat{v}, \tilde{q})$ mapped to color of valid track locations p. Pink bar charts show crash and goal probabilities for start configuration $(\hat{p}, \hat{v})$. Middle: Same as left, with additional mapping of summed crash probabilities to height of track boxes. Right: *All velocity* mode shows summed probabilities aggregated over all start velocities – here the maximum of the summed crash probabilities. Colored charts show crash and goal probabilities for all start configurations $(\hat{p}, v)$.

and hovering to neighboring locations of $\hat{p}$. $\hat{p}$ is visualized by a yellow box and $\hat{v}$ by a bent arrow with a direction dependent color scale as shown in Figure 4 (left). The user can either focus on $\hat{v}$, or on all possible velocities v for which probabilities have been precomputed. The latter *all velocity* mode is auto-selected by hovering over the track with the Ctrl modifier pressed as illustrated in Figure 4 (right).

Heat Map Overview. We extend the DSMC20 heat map overview by the option to adjust the height of the track boxes, as shown in Figure 4 (middle). The user can select the probability type with hotkeys. In *all velocity mode* the summed probabilities are aggregated per start location over all start velocities with one of the user-selectable aggregation operators min, max or range = min − max. In this way we can visualize $\sum_{\tilde{q}} P(p, \hat{v}, \tilde{q})$ and $\text{agg}_{\tilde{v}} \sum_{\tilde{q}} P(p, \tilde{v}, \tilde{q})$, where agg denotes the selected aggregation operator. These visualizations can be used as a guidance to finding start configurations of interest and continuing further investigation from there.

Bar Chart Details. While the heat maps allow to efficiently determine start positions with a high rate of crashing, they do not show the crash positions q. TraceVis supports the latter through track boxes in the form of bar charts, visible e.g. in Figure 4 (left) in the back on the left-hand side (pink bar). The bar heights indicate the probability of crashing/reaching the goal, thus visualizing $P(\hat{p}, \hat{v}, q)$. In *all velocity* mode, an individual bar is included for each possible start velocity, i.e. we visualize $P(\hat{p}, v, q)$. To this end, we use the visual metaphor of spatial and color coding: the thin bars have the same color and positional offset as the start velocity vectors, as can be seen in Figure 4 (right).

6 Visualizing Policy Traces

Once a start configuration $(\hat{p}, \hat{v})$ of interest is found, a natural means to investigate further is to inspect the actual policy traces generated by DSMC starting from $(\hat{p}, \hat{v})$. TraceVis supports this in depth, through the techniques we describe next.

Trace Visualization Modes. To initiate trace inspection, the user presses the Enter key. TraceVis reads the trace file, and by default filters out duplicates of the traces

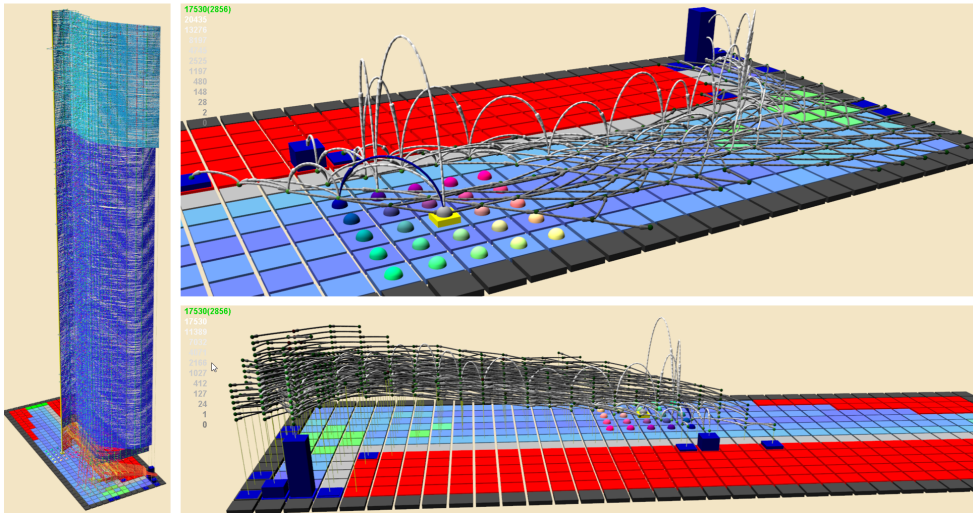

Fig. 5. Comparison of different trace rendering modes for a start configuration with 17530 traces of which 2856 remain after duplicate filtering. Left: *stacked* rendering of 2856 traces, sorted and color coded by end location. Top right: *spatial* aggregation showing segments as arcs with appearance counts mapped to height and luminance. Bottom right: *spacetime* mode disaggregates segments over time, mapping time to height.

while keeping track of the number of duplicates per trace. Figure 5 illustrates our three distinct modes to visualize traces: *Stacked*, *spatial* and *spacetime*.

Traces are visualized as colored 3D tubes or, in the case of an aggregated view, bent arrows. While the direction of traces is towards the goal positions in general, there are exceptions (e.g. when the agent needs to turn around first given a particular start velocity). Using tubes alone is not sufficient to show the direction of movement, hence we map an arrow texture onto the tubes.

Stacked Trace Visualization. In *stacked* mode, all traces that were calculated for a specific start configuration are shown stacked vertically above the track. Traces are sorted by their end location, and are arranged into two main clusters: one for traces that end at a goal position and one for those that crash. A sub-cluster is formed for each end position. As shown in Figure 5 (left), the goal (crash) clusters are colored with a blue to cyan (red to orange) color scale. Stacking in the order of the sub clusters and with cluster based coloring reduces visual cluttering significantly. The stacking offset in z direction can be adjusted with the mouse wheel.

Spatial Aggregation. Given the number of traces, simply visualizing the set of all traces is often not helpful. We design a more comprehensible visualization in terms of the velocity distribution over the track. To this end, we leverage the discrete nature of the underlying MDP, aggregating over discrete time and space. Specifically, we consider the possible move segments that action applications result in on the map. Each segment is defined by a start position p_s and end position p_e. Multiple segments of different traces share the same p_s and p_e. We can therefore compute a segment histogram by counting, for each segment (p_s, p_e), the number of appearances in the DSMC traces.

In the *spatial* mode shown in Figure 5 (top right) this histogram is visualized by mapping the appearance counts to bent arcs with height proportional to the appearance count. Additionally, the appearance count is mapped to the luminance of the bent arc with a gamma correction that can be adjusted with the mouse wheel. The absolute values of the appearance counts can be read from the legend. The mapping to bent arcs has the additional advantage that overlapping segments get visually separated (compare Figure 6). To maximize visual separation, we optionally allow mapping the arc height to half the segment length, resulting in circular arcs.

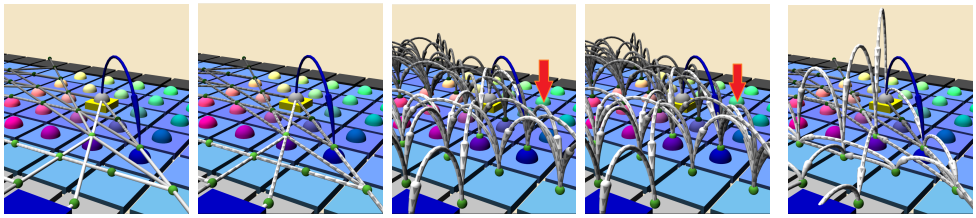

Fig. 6. Variants of segment rendering. From left to right: straight tube segments; added arrow texture; arcs with height equal to half step size; tilted arcs; arcs with height proportional to appearance counts.

For starting configurations where the agent needs to reverse its direction, overlapping inverse movements can be observed, see Figure 6 (middle). This prevents the visual separation of oppositely pointing arrows. We overcome this issue by slightly tilting the arcs sideways to visually separate them again as shown in Figure 6 (2nd from right).

Spacetime Visualization. The aggregation of segments characterized by start and end position (p_s, p_e) can be extended to also incorporate time information. Due to the noise influencing the successful application of a policy action, it is possible that no agent movement occurs in a given time step. Since we do not consider time in the other visualization modes, this is hard to notice. Furthermore, different traces might run along segments with equal start and end position but at different times. For exploration scenarios where this is important, we therefore implemented the *spacetime* mode as shown in Figure 5 (bottom right). Given the discrete-time nature of the MDP, the time t of a segment is simply defined by its position in the trace.

To calculate the aggregated segments for the *spacetime* mode we calculate the appearance count histogram over the triples (p_s, p_e, t). While rendering the segments, appearance counts are again mapped to luminance and optionally to arc height. We map time to an increasing height offset. This allows to efficiently identify faster and slower runs, as well as showing track points where the agent temporarily stops. A thin yellow stick is rendered to visually link trace vertices to their corresponding track locations.

To support the analysis of local behavior at a given position, we added another mouse hovering mode (activated by the Alt modifier), that restricts the view to the outgoing arcs at the current mouse pointer position. The height scale for arcs and the time offset can be adapted with the mouse wheel with different modifier keys.

Cluster Navigation. To reduce visual clutter in the aggregated trace visualization modes further, the user can navigate hierarchically through clusters and individual

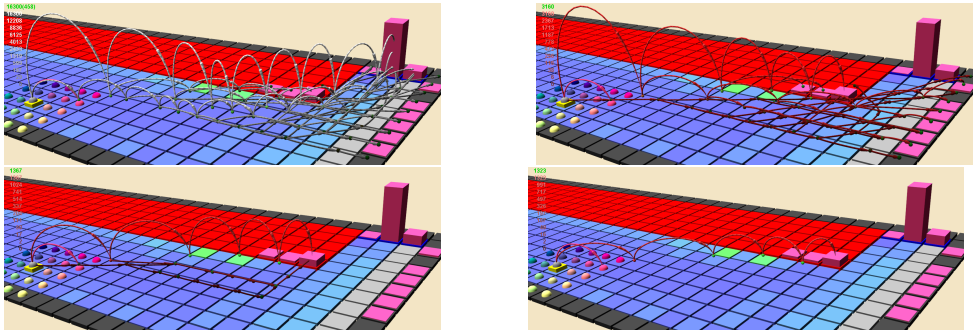

Fig. 7. Illustration of four cluster hierarchy levels in *spatial* mode: top level (top left), main cluster level (top right), sub-cluster level (bottom left), individual trace (bottom right).

traces as illustrated in Figure 7. The Enter key goes down the hierarchy from all traces to main clusters, then to sub-clusters and finally to individual traces. Sub-clusters can also be selected by hovering over a trace end position. At each level, the Up and Down arrow keys allow to navigate through the respective clusters/traces. Backspace is used to back up one hierarchy level. Pressing Backspace on the all-trace level terminates the trace mode, and brings the user back to probability mode.

Noise Visualization. All visualization modes make use of spheres placed at the track points where two segments are connected. The color coding of these spheres correlates to the amount of noise which influenced the agent during the DSMC runs. Red color indicates the appearance of noise, while green color states the successful movement according to the action chosen by the NN. This is especially useful for the in-depth examination of individual traces, visualizing the policy reacting to action failures at difficult track locations and configurations. For aggregated views, the color is interpolated between red and green according to the noise frequency.

Animation. To illustrate synchronicity in time across traces, we added an animation of spherical probe particles moving along the traces synchronously with adjustable speed. The animation is supported in all trace visualization modes. Space allows to toggle the animation, and with the Left and Right arrow keys one can step back and forth over individual time steps.

7 Case Study

To illustrate the use of TraceVis for policy behavior analysis, we now consider TraceVis from a user's (rather than a visualization researcher's) perspective. We highlight some interesting observations supported by TraceVis in analyzing the NN policies trained by DSMC20 in Racetrack.

Figure 8 shows our first observation, for a position just before the goal curve in Barto-small. We can see in Figure 8 (left) that, overall, the policy has a high chance of reaching the goal line as one would expect. However there are two start velocities not directed into the wall for which that is not so. Such problematic cases can very conveniently be located simply by dragging this overview presentation over the map. Selecting the most problematic start velocity in Figure 8 (right), it becomes evident

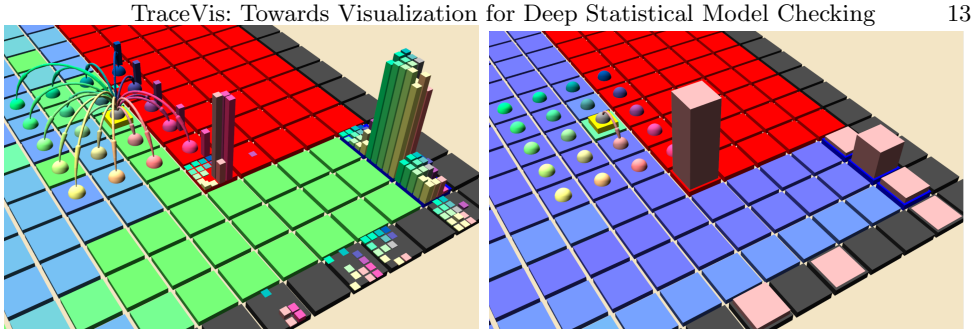

Fig. 8. Unsafe behavior near goal line. Overview of crash/goal probabilities across start velocities (left), and individual view for particularly problematic start velocity (right).

that policy behavior is highly, and unnecessarily, unsafe here. One can reach the goal with high probability simply by keeping the velocity and turning once the wall is cleared. Yet the policy tends to "cut the corner" and crash.

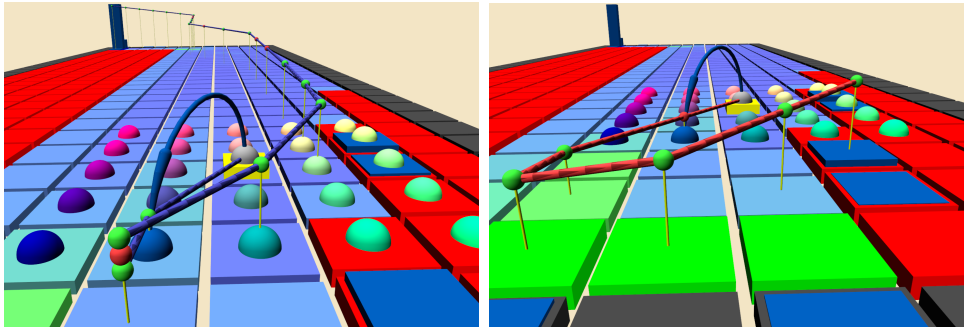

Fig. 9. Unsafe turning between walls: Successful (left) vs. crashed (right) trace.

Figure 9 shows another instance of curious policy behavior, also needlessly unsafe but less obviously so. Here the start position is in a tight spot between walls on the left and right, with a start velocity away from the goal and to the left. The safest decision would be to "turn around on the spot", i.e., decelerate, get left-right velocity down to 0, accelerate to the goal. Instead, as we see in the successful policy trace in Figure 9 (left), the policy over-accelerates to the right, going for a curve that only just avoids the right-hand side wall. Yet that curve relies on action success (green balls at move arcs in the visualization), and is brittle to action failure (red balls) as we see in the crashing trace in Figure 9 (right).

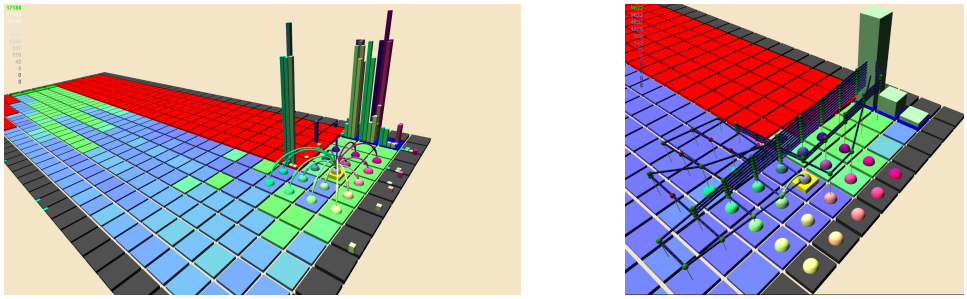

Fig. 10. Counterintuitive turning near goal: Overview of crash/goal probabilities across start velocities (left), and policy-trace overview for one particular start velocity (right).

Consider finally Figure 10. Here the agent is placed in front of the goal near the corner, and the overview (left) shows that the start velocities going towards the corner have a tendency to crash. This is not surprising given the actual risk of crashing here when actions fail, plus our previous observations in Figure 8. A more surprising insight is obtained when choosing a harmless start velocity, away from the goal at speed 1. Here we again get a counterintuitive turning behavior. Like above, a human player would "turn around on the spot", simply accelerating towards the goal and reaching it on a straight path with probability almost 1 (the only possibility to crash being 7 action failures in a row). Yet the trained neural network policy does not do that. Instead, it travels along a potentially large de-tour towards the start line, curving back to reach the goal on a trajectory scraping along the wall. This does work out with a high probability here, but nevertheless points to a weakness in policy behavior. Together with the odd behavior observed in Figure 9, it seems the policy generally has issues in situations requiring a full turn-around – giving a strong hint for possible re-training.

Note that TraceVis is key to all these observations. We miss them if we aggregate over start velocities (or fix these to 0 as DSMC20 does), if we aggregate over crash positions, if we have no in-depth visualization of policy traces.

Interestingly, TraceVis enabled us to find bugs in our own technology stack. Apart from initial data discrepancies due to bugs in cross-tool communication, this pertained also to a bug in our Jani model introduced when modifying it for this paper. Examining crash probabilities as a function of start velocity as illustrated above, we observed unintuitive results where start velocities heading directly into a wall did not lead to a crash. This behavior prompted us to re-examine the Jani model, identifying a bug in the vehicle automaton (where an initialization value was set incorrectly). Such a faulty behavior would not have been visible in DSMC20's heat maps as these ignore start velocities. The bug would be exceedingly hard (if not impossible) to identify based solely on modes, given the overwhelming amount of log data. Hence TraceVis can be useful also for debugging the model itself, arguably a crucial part of model checking.

8 Conclusion

Deep statistical model checking (DSMC) is a natural approach to quality assurance of MDP action policies represented by neural networks. We have designed and implemented a new tool, TraceVis, for visualizing and navigating DSMC results, as well as for deeply understanding the underlying causes by examining the actual policy traces. Our case study and own debugging experience with TraceVis suggests that interactive visualization can be useful for the practical application of DSMC, and potentially more general statistical model checking contexts as well.

We believe that the combination of formal methods with visualization is a key instrument to address the problem of NN action policy analysis (by DSMC or other methods). First, in contrast to traditional software artefacts, NN defy direct human inspection. Second, in many cases, full verification will be prohibitively complex or bound to fail (an autonomous car will hardly guarantee to avoid all possible accidents). Therefore, third, to gain trust in an action policy, human quality assurance analysts will have to understand its behavior and inspect its reactions against a

large space of possible environment behaviors. The combination of formal analysis tools with human-accessible data and results presentation seems predestined for that purpose. We view our work as one initial piece of this big puzzle.

Our contribution at this point is, of course, limited to the simple Racetrack benchmark, and it remains to be seen which ideas will carry over to other and more complex domains. That said, we believe that the Racetrack case study was useful, and remains useful, to focus on key aspects of many cyber-physical systems: position and velocity in physical space. This is different from the focus on visualizing complex strategy patterns, naturally entailed by the study of policies for computer games as done by the aforementioned previous works [43,39].

Our envisioned research trajectory thus is to stick to Racetrack-like case studies, incrementally extending these to reflect more aspects of, and ultimately approach, autonomous driving. Fuel consumption for example seems easy to integrate, Lidar sensing can be integrated by additional views, similarly for simple camera images showing a grey-scale view of what's ahead. In 3-dimensional extensions like drone control, most of TraceVis's current features will be applicable.

In the longer term, a major challenge will be multi-dimensional state spaces, in particular multi-agent behavior like traffic in autonomous driving/drones, or collaborative agents in cyber-physical production. We envision to extend TraceVis by dimension reduction techniques, providing an abstract visualization of the state space, where we can morph back to 3D space in order to focus on the behavior of individual agents or the 3D relationship between the agents at certain instances of time. It may also be possible to leverage prior insights from computer games, not as much to elicit strategy patterns, but to elicit environment patterns, like typical traffic scenarios of special cases of particular interest. We expect to still stick to physical space as the main organization paradigm for the visualization.

Acknowledgements: This work was partially supported by the ERC Advanced Investigators Grant 695614 (POWVER), by DFG grant 389792660 as part of TRR 248 (see https://perspicuous-computing.science) and by DFG Cluster of Excellence CeTI (EXC2050/1 Project ID 390696704).

References

1. Barto, A.G., Bradtke, S.J., Singh, S.P.: Learning to act using real-time dynamic programming. Artif. Intell. **72**(1-2), 81–138 (1995)
2. Bogdoll, J., Fioriti, L.M.F., Hartmanns, A., Hermanns, H.: Partial order methods for statistical model checking and simulation. In: FMOODS-FORTE. pp. 59–74. LNCS 6722 (2011)
3. Bonet, B., Geffner, H.: Labeled RTDP: improving the convergence of real-time dynamic programming. In: ICAPS. pp. 12–21 (2003)
4. Budde, C.E., D'Argenio, P.R., Hartmanns, A., Sedwards, S.: A statistical model checker for nondeterminism and rare events. In: TACAS (2). pp. 340–358. LNCS 10806 (2018)
5. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: Quantitative model and tool interaction. In: TACAS (2). pp. 151–168. LNCS 10206 (2017)
6. Croce, F., Andriushchenko, M., Hein, M.: Provable robustness of relu networks via maximization of linear regions. In: AISTATS. pp. 2057–2066. PMLR 89 (2019)

7. Dehnert, C., Junges, S., Katoen, J., Volk, M.: A storm is coming: A modern probabilistic model checker. In: CAV. pp. 592–600. LNCS 10427 (2017)
8. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: ATVA. pp. 269–286. LNCS 10482 (2017)
9. Etienne, L., Devogele, T., Buchin, M., McArdle, G.: Trajectory Box Plot: a new pattern to summarize movements. International Journal of Geographical Information Science **30**(5), 835–853 (May 2016). https://doi.org/10.1080/13658816.2015.1081205, https://doi.org/10.1080/13658816.2015.1081205, publisher: Taylor & Francis _eprint: https://doi.org/10.1080/13658816.2015.1081205
10. Gardner, M.: Mathematical games. Scientific American **229**, 118–121 (1973)
11. Gardner, M., Dorling, S.: Artificial neural networks (the multilayer perceptron)a review of applications in the atmospheric sciences. Atmospheric Environment **32**(14), 2627 – 2636 (1998)
12. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.T.: AI2: Safety and robustness certification of neural networks with abstract interpretation. In: IEEE Symposium on Security and Privacy 2018. pp. 3–18 (2018)
13. Gros, T.P., Gro, D., Gumhold, S., Hoffmann, J., Klauck, M., Steinmetz, M.: Tool Demonstration, Source Code and Data for "TraceVis: Towards Visualization for Deep Statistical Model Checking" (2020), available at http://doi.org/10.5281/zenodo.3961196
14. Gros, T.P., Hermanns, H., Hoffmann, J., Klauck, M., Steinmetz, M.: Deep Statistical Model Checking. In: Proceedings of the 40th International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE'20) (2020)
15. Gumhold, S.: The computer graphics and visualization framework. https://github.com/sgumhold/cgv, accessed: 18-May-2020
16. Gumhold, S.: Splatting illuminated ellipsoids with depth correction. In: Ertl, T. (ed.) Proceedings of the Vision, Modeling, and Visualization Conference 2003 (VMV 2003), Mnchen, Germany, November 19-21, 2003. pp. 245–252. Aka GmbH
17. Hahn, E.M., Li, Y., Schewe, S., Turrini, A., Zhang, L.: iscasMc: A web-based probabilistic model checker. In: FM 2014. pp. 312–317. LNCS 8442 (2014)
18. Hartmanns, A., Hermanns, H.: The Modest toolset: An integrated environment for quantitative modelling and verification. In: TACAS. pp. 593–598. LNCS 8413 (2014)
19. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: VMCAI. pp. 73–84. LNCS 2937 (2004)
20. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., Kingsbury, B.: Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine **29**(6), 82–97 (2012)
21. Hohman, F., Kahng, M., Pienta, R., Chau, D.H.: Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers. arXiv:1801.06889 [cs, stat] (May 2018), http://arxiv.org/abs/1801.06889, arXiv: 1801.06889
22. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: CAV (1). pp. 3–29. LNCS 10426 (2017)
23. The JANI specification. http://www.jani-spec.org/, accessed on 28/02/2020
24. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: CAV (1). pp. 97–117. LNCS 10426 (2017)
25. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS. pp. 1097–1105 (2012)
26. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. pp. 585–591. LNCS 6806 (2011)

27. Kwiatkowska, M.Z., Norman, G., Parker, D.: Stochastic model checking. In: SFM 2007, Advanced Lectures. pp. 220–270. LNCS 4486 (2007)
28. Li, J., Liu, J., Yang, P., Chen, L., Huang, X., Zhang, L.: Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In: SAS. pp. 296–319. LNCS 11822 (2019)
29. McMahan, H.B., Gordon, G.J.: Fast exact planning in Markov decision processes. In: ICAPS. pp. 151–160 (2005)
30. Mirzargar, M., Whitaker, R.T., Kirby, R.M.: Curve Boxplot: Generalization of Boxplot for Ensembles of Curves. IEEE Transactions on Visualization and Computer Graphics **20**(12), 2654–2663 (Dec 2014). https://doi.org/10.1109/TVCG.2014.2346455, conference Name: IEEE Transactions on Visualization and Computer Graphics
31. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M.A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)
32. Pineda, L.E., Lu, Y., Zilberstein, S., Goldman, C.V.: Fault-tolerant planning under uncertainty. In: IJCAI. pp. 2350–2356 (2013)
33. Pineda, L.E., Zilberstein, S.: Planning under uncertainty using reduced models: Revisiting determinization. In: ICAPS (2014)
34. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley (1994)
35. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018)
36. Stoll, C., Gumhold, S., Seidel, H.p.: Incremental raycasting of piecewise quadratic surfaces on the GPU. In: 2006 IEEE Symposium on Interactive Ray Tracing. pp. 141–150. IEEE. https://doi.org/10.1109/RT.2006.280225, http://ieeexplore.ieee.org/document/4061556/
37. Tominski, C., Schumann, H., Andrienko, G., Andrienko, N.: Stacking-Based Visualization of Trajectory Attribute Data. IEEE Transactions on Visualization and Computer Graphics **18**(12), 2565–2574 (Dec 2012). https://doi.org/10.1109/TVCG.2012.265, conference Name: IEEE Transactions on Visualization and Computer Graphics
38. Tukey, J.W.: Mathematics and the picturing of data. In: Proceedings of the International Congress of Mathematicians, Vancouver, 1975. vol. 2, pp. 523–531 (1975)
39. Wang, J., Gou, L., Shen, H.W., Yang, H.: DQNViz: A Visual Analytics Approach to Understand Deep Q-Networks. IEEE Transactions on Visualization and Computer Graphics **25**(1), 288–298 (Jan 2019). https://doi.org/10.1109/TVCG.2018.2864504, https://ieeexplore.ieee.org/document/8454905/
40. Wang, J., Hazarika, S., Li, C., Shen, H.W.: Visualization and Visual Analysis of Ensemble Data: A Survey. IEEE Transactions on Visualization and Computer Graphics **25**(9), 2853–2872 (Sep 2019). https://doi.org/10.1109/TVCG.2018.2853721, conference Name: IEEE Transactions on Visualization and Computer Graphics
41. Wicker, M., Huang, X., Kwiatkowska, M.: Feature-guided black-box safety testing of deep neural networks. In: TACAS (1). pp. 408–426. LNCS 10805 (2018)
42. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: CAV. pp. 223–235. LNCS 2404 (2002)
43. Zahavy, T., Zrihem, N.B., Mannor, S.: Graying the black box: Understanding DQNs. arXiv:1602.02658 [cs] (Apr 2017), http://arxiv.org/abs/1602.02658, arXiv: 1602.02658