Interactive Visualization Meets Probabilistic Model Checking

Max Korn[1]⋆, Julián Méndez[2]⋆, Sascha Klüppelholz[1], Ricardo Langner[2], Christel Baier[1], and Raimund Dachselt[2]

[1] Institute of Theoretical Computer Science, TU Dresden, Germany
{max.korn, sascha.klueppelholz, christel.baier}@tu-dresden.de
[2] Interactive Media Lab Dresden, TU Dresden, Germany
{julian.mendez2, ricardo.langner, raimund.dachselt}@tu-dresden.de

Abstract. State-of-the-art Probabilistic Model Checking (PMC) offers multiple engines for the quantitative analysis of Markov Decision Processes (MDPs), including rewards modeling cost or utility values. Despite the huge amount of internally computed information, support for debugging and facilities that enhance the understandability of PMC models and results are very limited. As a first step to improve on that, we present the basic principles of PMC-Vis, a tool that supports the exploration of large MDPs together with the computed PMC-results per MDP-state through interactive visualization. By combining visualization techniques, such as node-link diagrams and parallel coordinates, with quantitative analysis capabilities, PMC-Vis supports users in gaining insights into the probabilistic behavior of MDPs and PMC-results and enables different ways to explore the behaviour of schedulers of multiple target properties. The usefulness of PMC-Vis is demonstrated through three different application scenarios.

1 Introduction

Probabilistic model checking (PMC) is a well established technique used in the field of formal verification to analyze and assess the behavior of probabilistic systems. Sources of probabilistic behavior include randomized algorithms as well as stochastic assumptions about the external use of the system (i.e., the system environment) and error probabilities. PMC combines concepts from probability theory and model checking to provide quantitative insights into the reliability and performance of such systems. PMC supports various types of stochastic models. It is applicable at all stages of the life cycle of a system (i.e., in the design phase, at runtime, and at inspection time) for the evaluation of system properties, such as reliability, safety, and various cost and performance metrics.

State-of-the-art model checkers, such as PRISM [23], Storm [15], and MRMC [21], have successfully been applied in various application fields, including computer science, engineering, robotics, and biology [3, 6]. Nevertheless, understanding, debugging and using computed results for a given general goal often involves a

⋆ Authors contributed equally

laborious manual process that is only supported by handwritten and tailored software scripts in combination with general purpose tools for handling large datasets. Existing model checkers provide rather limited support for *(re)configuration* tasks (i.e., calibrating systems before or at runtime to complete set objectives under possibly multiple criteria) that involve systematically exploring and understanding (**1**) complex system behavior and (**2**) metrics returned by PMC processes. For example, PRISMs simulator engine allows for exploring individual paths either on the command line or inside the graphical user interface. Tools like IBM's Promela [20] (and also PRISM itself) provide support for organizing experiments and visualizing model checking results gained in multiple model checking runs. The model checker Spin [16] provides a simple view for program graphs. Although model checkers usually provide basic functionality or interfaces to other tools that can indirectly contribute in addressing (re)configuration tasks, to the best of our knowledge, no model checker harnesses the full potential of advanced interactive visualization techniques.

From the visualization side, approaches for graph visualization [27, 22] are suited for tasks in PMC, where the models can be seen as *large*, *multivariate*, *multi-criteria decision* graphs. While the research branches for each of those characteristics, many of the challenges arising from graphs that simultaneously exhibit multiple such qualities remain open. From the application viewpoint, the missing visual tool support for using computed PMC-results has already been recognized in some domains, such as DNA sequencing (e.g., HMMEditor [6]) and automated driving (e.g., TraceVis [13]). These tools provide support for solving concrete problems in their respective domains, yet they do not transfer to more domain-independent general goals such as (re)configuration. To the best of our knowledge, there is no PMC-tool that integrates sophisticated visualization to support the exploration and facilitate the understanding of large models and their (functional or non-functional) properties. We therefore set to create a visual tool that explicitly supports the above mentioned processes (**1**) and (**2**) while remaining independent from a concrete application field.

The main challenges we aim our efforts at are: (**C-1**) Scalability: to handle models that range from hundreds to several million states, both in terms of supporting user understanding and efficiency of computations. (**C-2**) Practicability: to empower users to solve (re)configuration tasks through an interactive and intuitive process, thereby providing appropriate representations, interactions and configuration options. (**C-3**) Extendibility: the tool should be ready-to-use but flexible enough to be extended for alternative operational models, qualitative and quantitative properties, and goals beyond (re)configuration, raising additional challenges on the design of a future-proof and domain-agnostic tool.

In fact, our vision is to have an integrated development environment (IDE) available at design time, in which the interactive visualization is interconnected with the editor and other views while communicating directly with model checkers. We contribute our approach for the creation of such a platform, and present our initial step towards it as a prototype of a new tool called PMC-Vis, that connects the PRISM model checker on the backend side with a visualization frontend.

Our focus is on *Markov Decision Processes* (MDPs) as operational models, in which the initial states stand for design alternatives (configurations) and the nondeterministic choices stand for possible reconfiguration steps. Metrics in this setting are of a quantitative nature and include probabilities and expectations of random variables, standing for either costs or gained rewards. The backend of our tool PMC-Vis consists of a simple API shell around PRISM that allows for calls to the model checker at runtime and a database wrapper for efficient data-exchange to the frontend. The frontend consists of a web-based application that provides the following key functionalities:

- Interactive visualization for exploring MDPs including features for comparing metrics computed by PMC attached to MDP states, actions and schedulers.
- Visual support for finding suited configurations in families of MDPs.
- Visual support for reconfiguration in adaptive systems modeled as MDPs.

After delving deeper into related work (section 2), we present our proposed approach in section 3, detailing our implementation of PMC-Vis. Then, in section 4, we show the usefulness of PMC-Vis regarding (**1**) and (**2**) with the help of three scenarios. Lastly, section 5 illustrates performance measures of PMC-Vis. The current version of PMC-Vis, usage scenarios and performance experiments can be downloaded at `https://imld.de/pmc-vis`.

2 Related Work

First, we recall some general notions of probabilistic model checking (PMC) (for formal definitions, see [4]). *Markov Decision Processes* (MDP) are formal, stochastic models used to describe systems that exhibit both controllable and uncontrollable behaviour. MDPs are typically represented as directed graphs with *states* as vertices and *actions* (also referred to as *transitions*) as edges. Edges can additionally express uncertainty, or in other words, the same action may lead to different states according to a probability distribution.

We will use the term *PMC-results* to refer to the output of the PMC, including probabilities of temporal events and expectations of random variables. Apart from the sole purpose of evaluation by means of a quantitative analysis, the computed PMC-results can be used to decide on different design alternatives, determine appropriate values for system parameters (i.e., parameter synthesis for configuration) and to synthesize suited adaptation policies (i.e., strategy synthesis for reconfiguration in adaptive systems) by examining *scheduler*, the functions that resolved the non-determinism during computation.

Visual Tools for PMC. While most existing model checkers (e.g., PRISM [23], Spin [16]) provide graphical user-interfaces to inspect the model and its results, they do not incorporate advanced visualization techniques (**C-2**). As we previously stated, most of the existing work is constrained to specific use-cases (**C-3**, e.g., HMMEditor [6] and TraceVis [13]). On the other hand, tools like [20, 11] visualize model checking outcomes, but are not concerned with the model or its inherent behaviour. A prominent example of this situation is MDPVis [26], where the results of simulations for policy optimization are plotted as histograms, followed by fan charts that show the evolution of the decisions as time-series data. While this tool provides a succinct overview of MDPs of arbitrary size (**C-1**), and provides insights for debugging, it does not visualize individual decisions nor the model itself, making it quite abstract for understanding the model behavior. Visual support for understanding general MDPs has been partially addressed for learning scenarios [28] as well as for the understanding of model checking counterexamples [18], using multiple coordinated views. While these approaches are useful in understanding small MDPs, the scalability with respect to the model size remains an open challenge.

In information visualization, the general challenge of visual scalability (**C-1**) remains open for graphs, despite the numerous works towards this [8, 24]. In broad terms, our case involves 1) *large*, 2) *multi-variate* 3) *decision* graphs. Each of those qualities present their own challenges and have been independently in the focus of a lot of approaches, as described in the following paragraphs.

Large Graph Visualization. Aggregation and clustering methods [12] to simplify the graph structure onto multiple *levels of abstraction*, provide means to interact with large graphs, exemplified by ZAME [7] and ASK-GraphView [1]. On the other hand, focus+context techniques [5] such as structure-aware fish-eye views approach [31] distort the representation to focus on parts of the graph. These techniques handle visual clutter by increasing the required amount of interaction, which could be detrimental for sense-making in the long term.

Multivariate Graph Visualization. The surveys on multivariate network visualization [27, 22] present various approaches fitting to our challenge of joining per-state PMC-results to MDPs. Most prominently, multiple coordinated view setups featuring *parallel coordinates plots* (PCPs) [19] are used to effectively display and explore the attributes alongside or within the graph view. HybridVis [25] illustrates an approach for graphs that are both large and multi-variate, using (a) a "hybrid-scale" view showing multiple levels of abstraction at once, accompanied by a (b) tree view for the abstraction hierarchy and (c) a PCP attribute view. However, visual clutter and the cognitive burden of the managing arbitrarily complex abstraction hierarchies remain as challenges.

Visualization for Decision-Making. For decision graphs, an overview does not typically communicate relevant insights, which is why the degree-of-interest (DOI) graphs approach [14] proposed to extend the graph from starting positions, on demand. This is fitting for MDPs as we can expand by discreet time steps of the model. However, visual clutter and subsequently, cognitive load, start low but increase as the graph is expanded in this approach. More generally, selections made on distinct axes of a PCP can satisfy multiple criteria simultaneously, making PCP a prime candidate for interactive decision-making (e.g., Parasol [29]). Furthermore, it follows that comparison facilities for outcomes (i.e., what-if analysis) can also support user decisions. Thus, we also take inspiration from discussed concepts for iterative graph exploration [30, 17] and comparison interfaces [2, 10].

Summary. Combining a degree-of-interest (DOI) graph view of the model with parallel coordinates views plots (PCP) for additional data attributes allows for understanding and exploration of large, multi-variate, and multi-criteria decision graphs. Joint with methods to manage the growing complexity of the views and facilities for what-if analysis, a recipe for visual exploration and understanding of MDPs integrating PMC-results is at hand.

3 Visualising Probabilistic Model Checking

Although it is possible to base PMC computations only on a decision over the initial states of the related MDP (as seen in MDPVis [26]), the user may be interested in information only obtainable from states beyond the initial ones. The model checker Prism [23], which we use in our approach, provides features (e.g., simulator, filter expressions, strategy conversion) to generate data related to the MDP components. However, understanding and debugging using the output of these features can take several days of manual combing through data using self-made scripts or general purpose tools, because the data is typically delivered as massive log files or in tabular form. The following sections describe our approach to improve on this situation, as the current process is error prone and often leads to overlooking crucial details.

3.1 Design Goals

We conducted interdisciplinary sessions bridging the visualization and formal verification communities to design and develop a platform that integrates visualization techniques and efficient retrieval methods. Towards such a platform, we defined the following design goals from the identified challenges (C-1 to C-3).

DG-1: *Support understanding and exploration of models.* As these processes highly depend on user preferences and expertise, we aim to provide various configuration options to support them, an easy-to-learn visual encoding and intuitive interaction features (**C-1**, **C-2**).

DG-2: *Support decision-making based on PMC-results.* Since PMC-results can consist of a large number of attributes, we aim to provide methods for the users to discern and filter through the multiple options with ease (**C-2**).

DG-3: *Provide scalable methods to support our tasks.* Our solution should be easy to use for both small and large models, with varying amounts of attributes from PMC-results (**C-1**)

DG-4: *Compute, retrieve and serve data efficiently.* Our solution should remain competitive in terms of speed and memory, ensuring minimal wait times and a smooth user experience (**C-1**).

DG-5: *Provide a holistic, flexible environment with easy setup.* Model checking environments are infamous for involving a multitude of dependencies and scripts that may at times be at odds with each other. We aim to provide a ready-to-use package that bypasses these setup complications (**C-3**).

We will refer to these goals whenever a design choice was made towards fulfilling them. We created PMC-Vis as a first step towards our envisioned platform. It consists of two web servers, *Backend* and *Frontend*. Our architecture decouples the heavy probabilistic model checking computations done in the *Backend* from the visualization and interaction service provided by the *Frontend*. Frontend can request data for states or subgraphs of interest from Backend, and uses this data to populate interactive visualizations. Since both servers specialize on their respective discipline-specific tasks, our architecture provides an easy to understand interface between not only the two servers, but also the two disciplines. Furthermore, this architecture is a step towards **DG-5**, since it is possible to install the servers on separate machines if needed. We additionally provide the current version of PMC-Vis as an accompanying artifact via Zenodo[3].

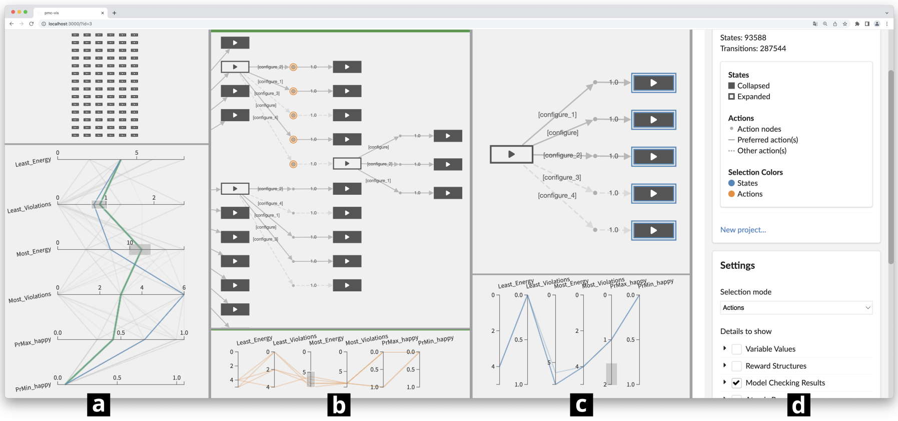
3.2 Backend

Computation times in model checking range from seconds to several days, depending on the size of the model in question. In order to satisfy **DG-3** and **DG-4** simultaneously, we need to find a way to retrieve necessary data from these processes without the need to start the process itself every time. In order to do this, the Backend server:

- triggers Model Checking processes to generate information whenever necessary (typically, only once at build time, or at introduction of a new project),
- stores and organizes the data efficiently for individual projects in a database,
- retrieves data on demand from the database, ensuring low access times and support for concurrent clients.

MC Process and PRISM. The Backend server manages instances of the model checker Prism [23], using its publicly available API[4]. As opposed to retrieving the data from resulting log files, we extract and store it in a database while it is generated during the stages of the PMC process. More concretely:
Before model checking, we extract structural information from the model input: existence of variables and their domains (i.e., possible values they can take), labels for the states, parameters for experiments, and reward functions.
After model construction, we create two tables, for (1) reachable states in the model, together with information about variables, labels and rewards, and (2) reachable actions, along with their labels and possible outcomes.
After model checking a property, the computed PMC-results are extracted for every reachable state of the MDP and stored in the above-mentioned tables. Additionally, the scheduler used for the computations is analyzed, and every action taken or possibly taken is noted in the table of actions. We do this instead of directly extracting the scheduler in order to see every action that maximizes/minimizes a property when there are actions with equal outcomes. Motivated by **DG-4**, we serve the data on demand, which fits the usage of degree-of-interest graphs [14] to visualize the MDP. By inserting data into a local

[3] The artifact DOI will be included for Camera Ready version
[4] `https://github.com/prismmodelchecker/prism-api`



States: 93588
Transitions: 287544

States
■ Collapsed
□ Expanded

Actions
● Action nodes
— Preferred action(s)
··· Other action(s)

Selection Colors
● States
● Actions

New project...

Settings

Selection mode
Actions

Details to show
☐ Variable Values
☐ Reward Structures
☑ Model Checking Results

Fig. 1: Screenshot of PMC-Vis, showing 3 *panes* (a, b, c). Pane (a) shows the initial states of the MDP (as a grid of unconnected nodes above, and as polylines in the PCP below). Pane (b) shows a set of selected actions (in orange). Pane (c) shows a selection over the possible outcomes of a chosen action (states, in blue). Finally, (d) shows the settings sidebar, currently linked to pane (b) as indicated by the green border on top of this pane.

database the moment we acquire it, the memory usage is kept similar to that of a normal Prism call.

Implementation. The Backend is a RESTful webserver implemented in Java with *dropwizard*[5] to wrap the Prism and manage the SQL-lite database. Using GET requests, subgraphs of the model are served in JSON format. The current exchange between Backend and Frontend mainly relies on two requests that deliver (1) the set of initial states of the graph, and (2) the set of states reachable from an input set of states, including the actions needed to reach them.

With the information from these resources, the Frontend can then populate the visualizations, as will be described in the following section. It is also possible to request information about single states and to request the entire model, but these additional resources are available for debugging purposes. Finally, project creation is triggered through a POST request that receives two files: (1) A model description written in the Prism specification language (.prism, .mdp, or .pm), that describes the MDP through variables with all possible values as our state space, and actions as guarded commands on those variables. (2) A property file describing the properties we want to check with PMC, written in Prism's property specification language (.props). In this file, properties can be given human-readable identifiers.

[5] `https://www.dropwizard.io/`

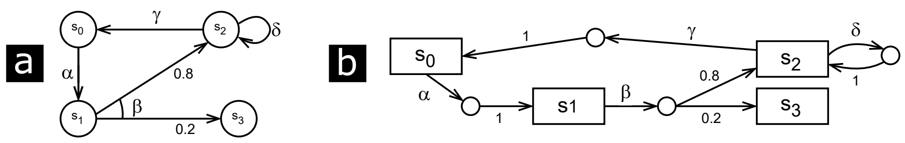


Fig. 2: Representation of MDPs. Version (a) is a more typical, compact version, with edges for actions and vertices for states. We propose version (b) for PMC-Vis, with a more stretched but explicit representation of actions versus outcomes.

3.3 Frontend

The Frontend of PMC-Vis is a web-based application that visualizes MDPs and their conjunct PMC-results through sequential *panes* (see Figure 1). Each pane has two sub-panes, for a *graph view* and an *attribute view*. Every graph view uses the degree-of-interest (DOI) [14] approach to reveal the MDP on demand (**DG-1**), and every attribute view uses configurable parallel coordinates plots (PCPs) [19] to navigate through the PMC-results related to selections within the graph view (**DG-2**). Both panes and sub-panes are re-sizable, and the content within them adapts to the available space. For example, Figure 1*a* shows a vertical PCP to better use the height of the sub-pane, whereas the PCPs in Figure 1*b* and *c* are horizontal. We discuss our approach and the current version of the interaction features in the following paragraphs, but the workflows will be discussed in more detail on section 4.

DOI MDPs: Towards **DG-1**, we designed a representation of MDPs, visible in Figure 2*b*. For the *states*, we use rectangular nodes. For the edges, instead of overloading their meaning to indicate both *actions* and probabilistic *outcomes* (as shown in edge β of Figure 2*a*), we explicitly separate these meanings by introducing handle nodes for the actions as label-less circles. This allows the user to more easily scan and parse the edges, since an outgoing edge from a state (e.g., α, β, γ) will always carry the action name, and any outgoing edge from a handle node shows the probability of reaching the state that it points to. For example, when only glancing over Figure 1*a* one could mistakenly get the impression that there are two outgoing actions from s_1 (as opposed to just β). This effect worsens when, (e.g., for each state), there are several outgoing actions, each with several probable outcomes. The handle nodes also simplify the selection of actions by area (e.g., using rectangle or lasso selections over the circle nodes instead of edges), which is particularly useful for reconfiguration tasks, where the user needs to decide between several actions. A set of selected actions can be seen in Figure 1*b*.

We indicate that a state has been expanded by removing its background color, alluding to the metaphor that the contents of the node are now outside of it. Furthermore, the labels of the states are provided by the Backend and can be converted to icons when certain properties are to be shown. In our case, we show the labels defined in the Prism specification for each state. Likewise, the scheduler choices are represented as solid edges, with the sub-optimal choices

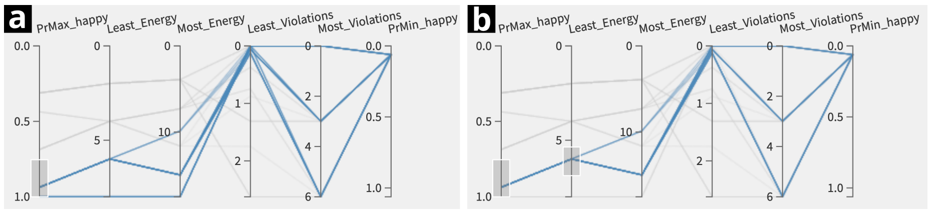


Fig. 3: Two parallel coordinate plots showing the state before (a) and after (b) brushing over the axis of the attribute $LeastEnergy$.

shown as dashed edges. This visual encoding can be seen in Figure 1b and c, with a legend present in the sidebar shown in Figure 1d. Besides allowing the users to scan the graph quickly, our proposed representation adapts well to the DOI approach. While the additional edges and nodes would impact response times if the entire graph (or large portions of it) had to be transferred at once, it barely has any effect on the small transfers we make (**DG-3**). Expanding states reveals their immediate next actions and states. This expanding can be done within the same pane (alt+click), or onto another pane (double click). Both of these options can also be accessed through the context menu (right click).

PMC-results via PCPs: Alongside the MDPs, a parallel coordinates plot (PCP) is shown in each pane to explore the additional PMC-results (**DG-2**). PCPs are well-known for their ability to accommodate a large amount of data both in terms of data points and data dimensions (**DG-3**). For interaction, PMC-Vis incorporates axes re-ordering, axes brushing for selections, hovering to preview selections, support for ordinal and nominal data, and other miscellaneous options. On a PCP, states (blue) and actions (orange) are presented as poly-lines over the axes of the plot at the locations where the values of each property lie. Figure 3 illustrates effect of brushing on PCP (i.e., click and drag along an axis). In Figure 3a, the user brushed over $PrMaxHappy$ from ≈0.7 to 1, selecting the states that are within that range. Brushing on other axes further refines the selection to include only states with values within all selected ranges. This effect is shown in Figure 3b, where the user additionally brushed over $LeastEnergy$. Using the context menu, one can directly select the highest or lowest value(s) for each axis, which is a natural adaptation of PCP for PMC. It is worth noting that, although more recent variants of PCP exist, they are specific to certain use cases that we don't immediately benefit from. The data that is shown on the PCP of a pane is linked to the selection of nodes on its respective graph view, meaning that states (blue) and actions (orange) can be loaded onto the PCP for filtering based on the shown PMC-results, which in turn can refine the selection made on the graph view through e.g., axes brushing. Loading content onto PCPs and synchronizing selections are currently supported via the context menu on the graph or attribute view.

Settings: The sidebar to the right of the panes (Figure 1d) provides several options to modify the shown attributes of the PMC-results, as well as several graph layout options (e.g., force-directed, hierarchical) that can be applied to each *pane* individually. Understanding the part of the MDP shown in each pane may be easier using different graph layout options for particular cases, and this flexibility in configuration supports varying user preferences (**DG-1**). Likewise, the shown attributes of the PMC-results can be adjusted per-pane, ensuring that only relevant information is shown for each MDP decision (**DG-2**).

Multi-pane possibilities: Each *pane* can show an arbitrary part of the MDP. Theoretically, a user could expand the entire MDP on a single pane, but the size of the MDP seldom allows for so much content to be digested at once. Our multi-pane approach allows the users to expand as much as desired and to create structural "check-points" by expanding a selection of the MDP onto new panes. Doing so creates multiple work spaces within the same MDP, which supports users in exploring without overloading panes or fearing loss of progress (**DG-1**). This approach also supports backtracking to previous states and work spaces, to re-evaluate decisions and explore branching paths of the MDP. Furthermore, through the context menu of any pane, the user can trigger the duplication of the source pane content onto a neighbor pane. This again allows users to try various configurations without fear that their previous work will be compromised. Besides, cloning a pane can be useful for comparison of distinct selections and schedulers, which supports decision-making beyond a single pane's PCP (**DG-2**). Lastly, we export/import features for the content of panes, which enable users to completely off-load their progress from a browser tab and start directly from any state or state selection that they had reached before (**DG-5**).

Implementation: The Frontend of PMC-Vis is powered by the *Cytoscape.js*[6] library (version 3.25.0 [9]) for the graph views, enhanced through several of its add-ons (e.g., for the various graph layouts), alongside the well-established *D3.js*[7] library (version 7.8.4) for the PCPs. The content server of the application uses *Express.js*[8] and the visual style adapts *Fomantic-UI*[9] and *sweet alerts*[10]. This selection of libraries and frameworks is also motivated by **DG-4**, as the load and rendering times are kept as short as possible. Furthermore, the choice to implement the Frontend as a web application was motivated by **DG-5**, as for end-users the only requirement to use the tool is access to a modern web-browser.

4 Usage Scenarios

To validate our results with respect to **DG-1** and **DG-2**, we illustrate the usage of PMC-Vis on the basis of an example model of a self-adaptive Server Management System (SMS). This model has a total of 93588 states, of which 84 are initial states. Roughly, the model describes how a number of *tasks* with

[6] `https://js.cytoscape.org/`
[7] `https://d3js.org/`
[8] `https://expressjs.com/`
[9] `https://fomantic-ui.com/`
[10] `https://sweetalert2.github.io/`

differing workload must be distributed to a number of *servers*. Tasks can be distributed to various servers, but a single *server* can only compute for one *task* at a time. The workload that a server can manage depends on its hardware and software settings. Roughly speaking, the behavior of this model can be split onto 3 cyclic *phases*: (1) generating tasks, (2) re-configuring, and (3) assigning work. These phases repeat until some termination criteria is reached (e.g., a set number of phases have been completed).

With this example SMS, we will illustrate three individual scenarios. For simplicity, we consider the scenarios separately, although, in reality, users may work on their tasks in parallel or seamlessly switch between them. Scenario 1 deals with the model configuration, in which the user is interested in finding a good server setup. A server setup describes how many servers and what hardware to use. Scenario 2 covers the model exploration, which aims at developing an understanding of its behavior and structure. Finally, scenario 3 focuses on finding managing strategies that fulfill typical SMS objectives: In this example, (1) finishing all tasks at the end of a phase and (2) minimizing ongoing costs (e.g., energy costs).

Scenario-1: Model Configuration To find a good initial configuration for our model (i.e., finding a server setup satisfying the users specifications), the user must select the initial state(s) of the MDP that represent the best configuration(s) for the SMS, and refine this selection until the best candidate is found. This task is central to **DG-2**. Once the user opens the model in PMC-Vis, the graph view and attribute view of a single pane are populated with the 84 initial states of our model, as seen in Figure 1*a*. In our example, these are the PMC-results for our main criteria, maximized and minimized over the MDP:

– The probability of finishing all assigned tasks ($PrMin/PrMax\_Happy$).
– The expected energy consumption over the task ($Least/Most\_Energy$).
– The accrued violations for tasks not fulfilled ($Least/Most\_Violations$).

All the SMS parameters that cannot change at runtime, like server number and type of each server, are encoded in the initial states. In other words, at this point the PCP provides an overview of all initial configurations and their properties. This allows us to hone in on configurations that fulfill our criteria, i.e., we are now able to select the configurations that we deem as satisfactory, by brushing on the PCP. In our case, the user would want to select the maximum probability of finishing all tasks (by brushing over $PrMaxHappy$ as close to 1 as possible), and from the resulting options, a second selection should be made for minimal energy consumption (brush over the lowest values in $MostEnergy$ or $LeastEnergy$). This is the state shown in Figure 3*b*. When hovering over an axis, an area surrounding the cursor position shows a preview with a small tooltip indicating the number of poly-lines that would be selected if the user brushes over the preview area. Thus, by hovering, the user sees that the current selection contains 9 configurations. The user can then note (as is visible in Figure 3) that the property $MostViolations$ further refines this selection onto three groups at values 0, 3 and 6. Naturally, the user would decide to minimize the violations,

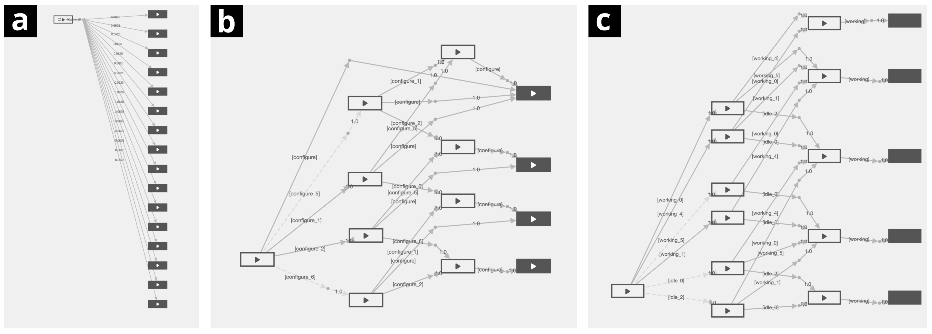


Fig. 4: 3 Phases of our model shown as patterns on graph views: (a) Generating tasks, (b) re-configuring servers, (c) assigning work to each server.

and by brushing over the lines at value 0, the selection is refined to only 3 best candidates. With this, the user has found a set of good server setups. It is worth noting that the patterns discernible in the PCP also point to relations between the PMC-results, which may be insightful for the user. For instance, it is possible to see that some configurations that keep violations at minimum are expected to use a lot of energy (connections from low values in $LeastViolations$ to high values in $MostEnergy$). Likewise, it can be seen that the minimum probability of finishing all tasks ($PrMinHappy$) is the same for all configurations.

Scenario-2: Model Exploration This scenario is primarily related to **DG-1**. Instead of randomly selecting initial states to explore, the previous scenario has yielded a small set of initial configurations that are worth exploring. Starting from these configurations it is possible to explore how the model develops model by expanding the state nodes.

Repeatedly expanding within the same pane is helpful to see the general structure of the model, but it would soon lead to many irrelevant states cluttering the view. For example, once a set of initial configurations has been chosen, the rest are no longer relevant to the user, meaning that if *Scenario-1* yielded 3 promising initial states, 81 will remain in the pane without purpose. While it would be possible to hide irrelevant states, actions and edges, the user may want to adjust their selection later, or they may want to explore sub-optimal paths in the model when debugging. Using our multi-pane approach, the user can create a checkpoint on their current graph and expand the new nodes to explore onto a new pane, without affecting the state of the previous panes.

By progressively expanding the graph in either the same pane or multiple, the user can see the general structure of the MDP through the patterns that appear in the graph view, as illustrated in Figure 4. Here we can see the three phases of the SMS in order as introduced earlier, with the corresponding patterns of (Figure 4a) a single MDP action assigning tasks to be computed over a even distribution, (Figure 4b) a number of sequential reconfiguration choices, which may (or may not) affect the SMS and its prospects, and (Figure 4c) tasks are assigned to each server, until all servers have started, leading back to task generation (Figure 4a) for each.

Scenario-3: Model Reconfiguration & Strategies. This scenario involves both **DG-1** and **DG-2**. In this Scenario, the user wants to find a good strategy for managing the servers. This means finding a strategy that re-configures the system into a fitting state (in phase 2), as well as assigns the right servers to the right tasks (for phase 3). In order to find such a strategy, the user must understand the possible outcomes of certain actions in the graph and compare the alternatives. Once a decision of interest has been identified, PMC-Vis provides multiple ways to inspect and compare the actions:

Per-node details: A tooltip with properties of interest from a node can be accessed through shift + click. If the user wants to change which properties appear in these tooltips and PCPs, the settings sidebar (visible in Figure 1d) provides a selection of all available properties. Possible properties include per-state PMC-results, variable/parameter configurations, state labels and reward functions.

PCP comparison: Nodes can be selected depending on the behavior set in the settings sidebar (only states, only actions, or both simultaneously). Action nodes summarize the properties of all states reachable from them, and, similar to states, they can be inspected in the PCPs (as described in subsection 3.3). In a process similar to *Scenario-1*, the user can compare selected actions by properties of interest to make a decision on which path to follow to continue exploring.

Scheduler choices: In order to understand how non-determinism is resolved for maximum or minimum PMC-results, the user can check the extracted scheduler information from the model checker (as described in subsection 3.2). By selecting the property of interest in the scheduler settings, PMC-Vis will highlight all edges related to the actions a scheduler optimizing this property could take. For example, selecting $PrMaxHappy$ as the scheduler property will indicate which actions to take to ensure that all system tasks will be terminated successfully. This not only allows the user to quickly discard sub-optimal actions and states, but also enables a initial understanding of optimal strategies for single goals with minimal effort. From the set of actions and states marked by the scheduler, it is also possible to refine the selection using the PCP as described in *Scenario-1*.

Multiple Schedulers: Although only one scheduler can be selected at a time for a pane, multiple SMS objectives may need to be considered simultaneously (e.g., since a higher number of tasks completed goes along with higher energy consumption). To find a fitting strategy, we need to combine the strategies of multiple schedulers. While it is possible to define properties that combine our objectives, or use Pareto functions to find compromises, the user is hardly able to influence the amount of trade-off applicable. For example, one may not be interested in a equal compromise between energy consumption and task fulfillment, but instead want to favor one side over the other. With this in mind, PMC-Vis allows cloning panes as an alternative which we believe suffices to support such multi-scheduler decisions in most cases. As stated earlier, each PCP can be manually set to select configurations optimal to multiple simultaneous criteria. With multiple simultaneous panes showing the same decision space, a side-by-side comparison of the PCP selections as well as the scheduler highlighting on the graph view can be inspected.

Exporting the strategy Aided by the previously discussed features, the user has accrued a sequence of decisions that altogether constitute the desired strategy to manage the servers, until the final states of the MDP are reached. However, the free exploration of the MDP in several panes means that not only the desired choices, but also several dangling nodes and half-explored paths may be present. To support the user in organizing their work across several panes PMC-Vis allows marking nodes of interest (via context menu), which can then be exported (i.e., downloaded as JSON) independently of the pane content for all panes at once. This exported sequence may contain gaps within the MDP, either because the user did not mark all the relevant nodes that they found, or because it was unclear which states best suited the strategy at certain points. For such cases, it is possible to import the working strategy onto a pane of PMC-Vis, and in a similar process to *Scenario-2*, the user may continue to explore the gaps in the system until a strategy has been completed.

5 Performance

In this section we discuss the performance of PMC-Vis to assess our work towards **DG-3** and **DG-4**. Our goal was to provide both fast build time and smooth interaction with the MDP regardless of its total size. For this experiment, we created a number of models with similar structure, but exponentially increasing size (measured in number of states of the MDP), that all can be solved for the same two properties:

- The minimum probability of an event happening.
- The maximum expected number of steps taken until that event.

Model Computation The times for expensive processes triggered in the Backend server are summarized in Figure 5*a* by model size. In this figure, we see the overhead posed by the creation and initial insertion into the database (*build DB*) compared to only executing the PMC procedures (*build PRISM*). This overhead is mainly bound to the database writing speed, where we perform around 40-50 operations per millisecond. However, all operations scale similarly w.r.t. model size, meaning we can compute PMC-results normally with Prism. The disk-storage space for the database peaked at 1.4GB for our largest example, where each database entry contains either a state or a transition.

Model Exploration To ensure a smooth user experience while exploring the model in the Frontend, we discuss the access times of the Backend server, depending on the size of the model. Figure 5*b* shows the average response time

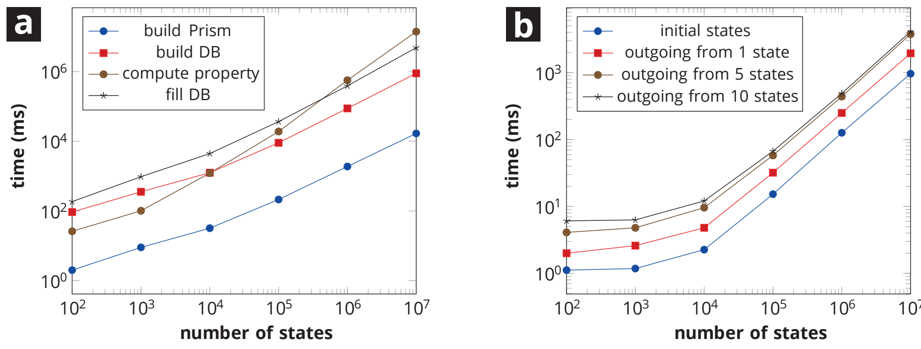


Fig. 5: (a): Times for single model computation. (b): Average response time (gathered over 10,000 responses). These experiments were carried out on a server with Intel(R) Xeon(R) L5630 CPUs with in total 8 physical cores and 189GB of DDR3 RAM.

for a random set of 1, 5 and 10 states (out of the entire state space) for different requests the Frontend could make. Since the Backend answers these request through database queries, access times rise with database size, which correlates with model size. On the other hand, we can see that even with a model with 10^7 states we keep response times for smaller requests around a second, and were able to answer larger requests in few seconds.

The only factor that influences Frontend performance (besides the wait time for Backend responses) is the number of elements (nodes and edges) loaded and shown on a pane at once. For instance, on a laptop with Intel(R) Core(TM) i7-7500U CPU and 16GB of RAM, and using a Chromium 114.0.5735.133 browser, a single graph view continues to operate smoothly with over 500 nodes. However, we do not foresee users working with that much content on a single pane, and we encourage (1) creating new panes often to isolate graph expansions and (2) making use of the export features, which enable users to avoid hard limits of the Frontend regardless of the model size.

6 Conclusion

We contributed our approach to support exploration of Markov decision processes and probabilistic model checking (PMC) configuration and reconfiguration tasks, realized as the PMC-Vis tool. We described how the combination of degree-of-interest (DOI) graphs and parallel coordinates plots (PCPs) with efficient model checking and retrieval methods results in support for users understanding model behaviour and conjunct PMC-results. Then, we showed how our multi-pane approach alleviates the limitations of such approaches, by allowing the DOI graphs to off-load content whenever desired. Combined with our methods to store and retrieve PMC-results effectively, we enabled a fluent user experience on models of both smaller and larger sizes. On the basis of three scenarios, we have shown how PMC-Vis can be used to answer various formal verification questions, especially in regard to the (re)configuration tasks. Lastly, we provided performance measures that illustrate how PMC-Vis handles the scalable nature of models without compromising user experience.

PMC-Vis presents a significant step towards our design goals, yet different aspects of scalability and usability must be further explored and improved towards our vision of a fully integrated platform for PMC workflows. For instance, the comparison and strategy definition tasks present large opportunities for more intricate support via automatic recommendations and intricate adaptable visualizations. We foresee that further formal model and verification methods would benefit from extensions of our approach. Thus, we look forward to further generalizing PMC-Vis in various directions towards an IDE for automata-based operational models, model checking of functional and nonfunctional properties and functionalities for various synthesis questions, including further features for what-if analysis.

Data-Availability Statement. PMC-Vis and the used models are open source and available on our supplementary web page at `https://imld.de/pmc-vis` and in the reproduction package at Zenodo[11].

Acknowledgements. This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy: EXC-2068, 390729961 – Cluster of Excellence "Physics of Life" and EXC 2050/1, 390696704 – Cluster of Excellence "Centre for Tactile Internet" (CeTI) of TU Dresden, by DFG grant 389792660 as part of TRR 248 – CPEC (see `https://cpec.science`) and by the German Federal Ministry of Education and Research (BMBF, SCADS22B) and the Saxon State Ministry for Science, Culture and Tourism (SMWK) by funding the competence center for Big Data and AI "ScaDS.AI Dresden/Leipzig".

References

[1] J. Abello, F. van Ham, and N. Krishnan. "ASK-GraphView: A Large Scale Graph Visualization System". In: *IEEE TVCG* 12.5 (2006), pp. 669–676. DOI: `10.1109/TVCG.2006.120`.

[2] K. Andrews, M. Wohlfahrt, and G. Wurzinger. "Visual Graph Comparison". In: *IV 2009*. 2009, pp. 62–67. DOI: `10.1109/IV.2009.108`.

[3] C. Baier, H. Hermanns, and J.-P. Katoen. "The 10,000 Facets of MDP Model Checking". In: *Computing and Software Science: State of the Art and Perspectives*. Springer International Publishing, 2019, pp. 420–451. ISBN: 978-3-319-91908-9. DOI: `10.1007/978-3-319-91908-9_21`.

[4] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[5] S. K. Card, B. Shneiderman, and J. D. MacKinlay. *Readings in Information Visualization—Using Vision to Think*. Series in Interactive Technologies. Morgan Kaufmann Publishers, 1999. ISBN: 1-55860-533-9.

[11] The artifact DOI will be included for Camera Ready version.

[6] J. Dai and J. Cheng. "HMMEditor: a visual editing tool for profile hidden Markov model". In: *BMC Genomics* 9.1 (2008), S8. ISSN: 1471-2164. DOI: 10.1186/1471-2164-9-S1-S8.

[7] N. Elmqvist et al. "ZAME: Interactive Large-Scale Graph Visualization". In: *2008 IEEE PacificVis*. 2008, pp. 215–222. DOI: 10.1109/PACIFICVIS.2008.4475479.

[8] V. Filipov, A. Arleo, and S. Miksch. "Are We There Yet? A Roadmap of Network Visualization from Surveys to Task Taxonomies". In: *CGF* (2023). DOI: 10.1111/cgf.14794.

[9] M. Franz et al. "Cytoscape.js 2023 update: a graph theory library for visualization and analysis". In: *Bioinformatics* 39.1 (2023). ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btad031.

[10] M. Gleicher. "Considerations for Visualizing Comparison". In: *IEEE TVCG* 24.1 (2018), pp. 413–423. DOI: 10.1109/TVCG.2017.2744199.

[11] H. Goldsby et al. "A Visualization Framework for the Modeling and Formal Analysis of High Assurance Systems". In: *ACM/IEEE MODELS*. LNCS. Springer, 2006, pp. 707–721. DOI: 10.1007/11880240_49.

[12] R. Görke, T. Hartmann, and D. Wagner. "Dynamic Graph Clustering Using Minimum-Cut Trees". In: *Algorithms and Data Structures*. Springer, 2009, pp. 339–350. ISBN: 978-3-642-03367-4.

[13] T. P. Gros et al. "TraceVis: Towards Visualization for Deep Statistical Model Checking". In: *ISoLA 2020*. LNCS. Springer International Publishing, 2021, pp. 27–46. DOI: 10.1007/978-3-030-83723-5_3.

[14] F. van Ham and A. Perer. ""Search, Show Context, Expand on Demand": Supporting Large Graph Exploration with Degree-of-Interest". In: *IEEE TVCG* 15.6 (2009), pp. 953–960. DOI: 10.1109/TVCG.2009.108.

[15] C. Hensel et al. *The Probabilistic Model Checker Storm*. 2020. arXiv: 2002.07080 [cs.SE].

[16] G.J. Holzmann. "The model checker SPIN". In: *IEEE TSE* 23.5 (1997), pp. 279–295. DOI: 10.1109/32.588521.

[17] T. Horak and R. Dachselt. "Hierarchical Graphs on Mobile Devices: A Lane-based Approach". In: *CHI MobileVis Workshop*. 2018.

[18] T. Horak et al. "Visual Analysis of Hyperproperties for Understanding Model Checking Results". In: *IEEE TVCG* 28.1 (2022), pp. 357–367. ISSN: 1941-0506. DOI: 10.1109/TVCG.2021.3114866.

[19] J. Johansson and C. Forsell. "Evaluation of Parallel Coordinates: Overview, Categorization and Guidelines for Future Research". In: *IEEE TVCG* 22.1 (2016), pp. 579–588. DOI: 10.1109/TVCG.2015.2466992.

[20] G. Kamhi, L. Fix, and Z. Binyamini. "Symbolic Model Checking Visualization". en. In: *FMCAD*. LNCS. Springer, 1998, pp. 290–302. ISBN: 9783540495192. DOI: 10.1007/3-540-49519-3_19.

[21] J.-P. Katoen et al. "The ins and outs of the probabilistic model checker MRMC". In: *Performance Evaluation* 68.2 (2011), pp. 90–104. ISSN: 0166-5316. DOI: https://doi.org/10.1016/j.peva.2010.04.001.

[22] A. Kerren, H. C. Purchase, and M. O. Ward, eds. *Multivariate Network Visualization*. Springer International Publishing, 2014. DOI: 10.1007/978-3-319-06793-3.

[23] M. Kwiatkowska, G. Norman, and D. Parker. "PRISM 4.0: Verification of Probabilistic Real-Time Systems". In: *CAV '11*. LNCS. Springer, 2011, pp. 585–591. DOI: 10.1007/978-3-642-22110-1_47.

[24] T. von Landesberger et al. "Visual Analysis of Large Graphs: State-of-the-Art and Future Research Challenges". In: *CGF* 30.6 (2011), pp. 1719–1749. DOI: 10.1111/j.1467-8659.2011.01898.x.

[25] Y. Liu et al. "HybridVis: An adaptive hybrid-scale visualization of multivariate graphs". In: *JVLC* 41 (2017), pp. 100–110. ISSN: 1045-926X. DOI: 10.1016/j.jvlc.2017.03.008.

[26] S. McGregor et al. "Facilitating testing and debugging of Markov Decision Processes with interactive visualization". In: *IEEE VL/HCC '15*. 2015, pp. 53–61. DOI: 10.1109/VLHCC.2015.7357198.

[27] C. Nobre et al. "The State of the Art in Visualizing Multivariate Networks". In: *CGF* 38.3 (2019), pp. 807–832. DOI: 10.1111/cgf.13728.

[28] M. Pfannkuch and S. Budgett. "Markov Processes: Exploring the Use of Dynamic Visualizations to Enhance Student Understanding". In: *JSE* 24.2 (2016), pp. 63–73. DOI: 10.1080/10691898.2016.1207404.

[29] W. J. Raseman, J. Jacobson, and J. R. Kasprzyk. "Parasol: an open source, interactive parallel coordinates library for multi-objective decision making". In: *EMS* 116 (2019), pp. 153–163. DOI: 10.1016/j.envsoft.2019.03.005.

[30] Y.-Q. Tan et al. "VecRoad: Point-Based Iterative Graph Exploration for Road Graphs Extraction". In: *2020 IEEE/CVF CVPR*. 2020, pp. 8907–8915. DOI: 10.1109/CVPR42600.2020.00893.

[31] Y. Wang et al. "Structure-aware Fisheye Views for Efficient Large Graph Exploration". In: *IEEE TVCG* 25.1 (2019), pp. 566–575. DOI: 10.1109/TVCG.2018.2864911.