# Using dtControl to process schedulers produced by the Modest Toolset

Laurens van der Wal
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
l.e.vanderwal@student.utwente.nl

## ABSTRACT

The MODEST TOOLSET is a tool that can analyse models and produce schedulers based on these models. These schedulers, also known as controllers or strategies, represent which action should be taken in a specific state in the model, to reach the optimal reward. When the models get more complex, the resulting schedulers become too complex for humans to easily analyse them. To improve this, there are tools like DTCONTROL, a toolset that processes schedulers using decision tree learning algorithms. We propose to use this DTCONTROL to process schedulers produced by the MODEST TOOLSET and evaluate the effectiveness of this processing, to make analysis of these schedulers much easier.

## 1. INTRODUCTION

The modelling language MODEST was introduced in 2001. In 2012 the MODEST TOOLSET was introduced, incorporating analysis of stochastic hybrid systems and special cases thereof [17], so the models written with MODEST could be analysed. Since then, it has been applied in several case studies. The MODEST TOOLSET supported the model-based analysis of electric vehicles [13], and the probabilistic modelling and verification of the MODEST TOOLSET were applied in reliable network-on-chip system design [21].

The MODEST modelling language is very useful in the modelling *Markov Automata*. Like in the modelling of Bitcoin attacks to try and optimize the attacks [19].

The MODEST TOOLSET itself processes models written in a few different formats. These include Modest, JANI, and PRISM. The toolset will check these models and produce the state space accordingly. Based on this state space it produces an optimal scheduler. The scheduler controls the model, it restricts behavior so that all scheduling requirements of the model are met. The state space is a set of all the different states of the model. Transitions occur between these states by performing actions. The states themselves are defined by the combination of values of the variables and the steps in the process of the system that we are modelling.

One of the shortcomings of model checkers like the MODEST TOOLSET has, is that the number of possible states generated by the model can be very high [20]. This number of states can make it difficult for a human interpreter to get any useful information out of the generated schedulers. We can address this issue by finding a way to process the schedulers to make them easier to interpret.

dtControl is one way to process schedulers. It is a tool that represents the schedulers as decision trees, a tree-like model of decisions and their consequences,and applies decision tree learning to minimize this decision tree, resulting in an overall decrease of decision nodes [2]. This system was later improved into dtControl 2.0 [3]. In the related papers different algorithms are used to process the schedulers. In the paper it is concluded that their new processing technique MaxFreq is efficient at processing schedulers. The decrease in decision nodes that DTCONTROL achieves is quite promising, as the resulting decision trees always have less decision nodes than there are states in the original scheduler. Thus, dtControl is a candidate to help interpreting the schedulers produced by the MODEST TOOLSET. Figure 1 gives an example of a scheduler, that has been processed by dtControl.

Prior to this paper, the dtControl toolset does not support schedulers produced by the MODEST TOOLSET as input

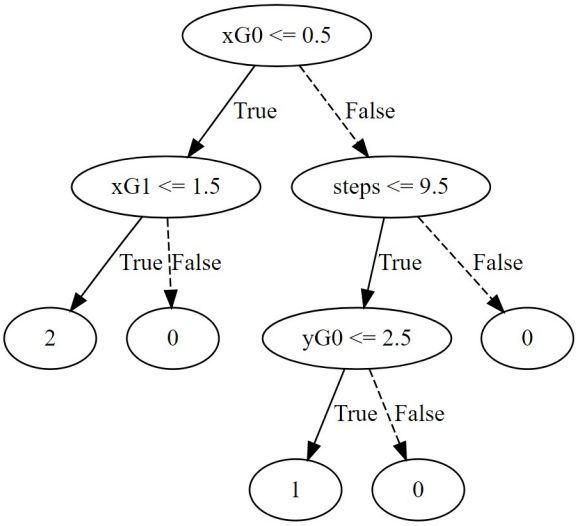


**Figure 1. The processed scheduler of pacman.v1.jani, used with variable MAXSTEPS=20, as used in Section 7. In this figure, there are two types of nodes; decision nodes and state nodes. Decision nodes check whether a variable (depicted by $xG0 <= 0.5$ in the topmost node), fulfills a condition and determines to which variable the model will evaluate next, until it reaches a leaf node. Leaf nodes depict the action that should be executed in this state, as determined by the scheduler.**


$28^{th}$ *Twente Student Conference on IT* Febr. $2^{nd}$, 2018, Enschede, The Netherlands.

for its processing. This means that either support for the MODEST TOOLSET should be implemented in dtControl or support for dtControl into the MODEST TOOLSET, before any testing can occur. However, since dtControl offers ways of integrating new formats in their interpreter [4], the aim of this research is implementing support for the schedulers produced by the MODEST TOOLSET into DTCONTROL.
In this paper we present a method to aallow dtControl process the Modest schedulers, and in the process making the schedulers easier to interpret. We checked this method on correctness, by comparing if the scheduler and decision tree produce the same action in the same state, and efficiency, by comparing the number of state-action pairs in the scheduler to the number of decision nodes in the decision tree, and it should be available for use in future studies where model-checking, through the use of the MODEST TOOLSET, is applied.

# 2. PROBLEM STATEMENT

In this paper, the main question that remains to be answered is as follows:

- Is dtControl an effective tool to use in making schedulers generated by models which the MODEST TOOLSET more compact, to make the schedulers more easily interpretable for humans?

To solve this problem, we define the following subproblems. Solving them will lead us to answering the main question:

1. Can dtControl correctly process a scheduler produced by the MODEST TOOLSET?
2. How effective is dtControl at decreasing the number of decision nodes, when applied to Modest models?

# 3. RELATED WORK

There are many different ways to represent models and schedulers. These representations encode all possible states and actions within a model. One such representation is a binary decision diagram, which has been a popular approach to represent both models and schedulers [6,7,11,22,23]. These binary decision diagrams have a tree structure with Boolean functions representing the nodes, and are thus a bit more compact than the traditional scheduler. However, this does not mean that they are easier to read. To improve this decision trees can be used to represent schedulers [7].
These decision trees are more efficient, as they can be trained to memorize which features are important for the scheduler, and which are not important. The algorithm then uses this to construct the tree based on information gain, resulting in a more compact representation which can be more easily analysed.
These decision trees have been applied in other places in the past, such as to find to find reusable Homomorphisms in a Markov decision process [24]. Or to represent the scheduler for two player games [8].In these examples the training algorithm produces error-free representations.

# 4. THE MODEST TOOLSET

The MODEST TOOLSET [18] is a comprehensive suite of tools for quantitative modelling and verification. The primary forms of input language for analisys are MODEST and JANI [9]. The Toolset offers support for *Markov Automata* in several of its tools, of which the one that will be used in this paper is **mcsta** [10].

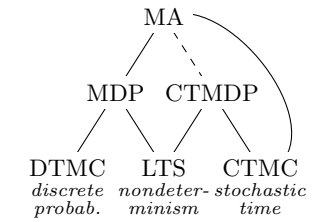


**Figure 2. The MA family tree [19]**

## 4.1 Markov Automata

*Markov Automata*, or MA for short, were introduced in [12] as a version of Segala's probabilistic automata with continuous time [14]. This type of automaton is closed under parallel composition and hiding. The Automaton has two types of transitions; one is a probabilistic transition and labeled with an action(like an LTS), and the other is a Markovian transition and labeled with a positive real number between 0 and 1, representing the probability of that transition occuring (like a DTMC). The relation between MA, LTS and DTMC is further illustrated in Figure 2. In this paper the focus will be on Markov Decision Processes, or MDP for short, which are MA that moves only in discrete time steps instead of continuous time. This is in contrast to a Continuous Time Markov Decision Process , or CTMDP, which moves in continuous time.
The following definitions are reused from [14]. We let $Act$ be the universe of actions that has the empty action: $\tau \in Act$, and Distr($S$) the set of distributions over the countable set S.

DEFINITION 1. *A Markov automaton (MA) is a tuple* $M = (S, A, \rightarrow, \Rightarrow, s_0)$ *where S is a nonempty, finite set of states with initial state* $s_0 \in S, A \subseteq Act$ *is a finite set of actions and*

- $\rightarrow \subseteq S \times A \times$ *Distr(S) is the probabilistic transition relation, and*
- $\Rightarrow \subseteq S \times \mathbb{R}_{>0} \times S$ *is the Markovian transition relation.*

We abbreviate $(S, \alpha, \mu) \in \rightarrow$, where by $s \xrightarrow{\alpha} \mu$ and $(s, \lambda, s') \in \Rightarrow$ by $s \overset{\lambda}{\Rightarrow} s'$. A MA can travel between states via the Markovian and probabilistic transitions. If $s \xrightarrow{\alpha} \mu$, it will leave state $s$ through the execution of action $a$ and arrives in some state $s' \in S$, with the probability $\mu(s')$. If $s \overset{\lambda}{\Rightarrow} s'$, the automaton will move from s to s′ at rate $\lambda$, unless there is a transition available from s labeled as the empty action $\tau$. In the latter case it will always be taking that transition without delay.

DEFINITION 2. *A path in an MA is an infinite sequence* $\pi = s_0 \xrightarrow{\sigma_0,\mu_0,t_0} s_1 \xrightarrow{\sigma_1,\mu_1,t_1} \ldots$ *with* $s_i \in S, \sigma_i \in$ Act $\cup\{\perp\}$ *, and* $t_i \in \mathbb{R}_{\geq 0}$.

For $\sigma_i \in Act, s_i \xrightarrow{\sigma_i,\mu_i,t_i} s_{i+1}$ denotes that the MA has moved from $s_i$ to $s_{i+1}$ through action $\sigma_i$ after residing $t_i$ time units in s, with a probability $\mu(s_{i+1})$. On the other hand, $s_i \xrightarrow{\perp,\mu_i,t_i} s_{i+1}$ denotes that a Markovian transition led to $s_{i+1}$ with probability $\mu(s_{i+1}) = \mathbf{P}(s_i, s_{i+1})$, which

denotes the probability of getting to $s_{i+1}$ from state $s_i$. Such a path $\pi$ is a resolution of all stochastic, probabilistic, and nondeterministic choices. The set of all paths that end in a state in the MA M is denoted by $\Pi_fin(M)$. [19]

To be able to find an optimal path, we need something to evaluate paths with. This is done by defining properties for a model. For example these properties evaluate based on maximizing a certain probability, or minimizing the expected number of time the model takes. Based on these properties we can assign reward values to paths. The next definition is adapted from [19].

DEFINITION 3. *Let M be a MA. We define a scheduler as a function:* $\varsigma : \Pi_fin(M) \to TR(M)$ *where* $TR(M)$ *denotes the set of all transitions. We write* $\mathfrak{S}(M)$ *for the set of all schedulers of M.*

So in short, a scheduler is a function that takes a path in a MA M, and outputs all the transitions that it takes. This scheduler is deterministic.
If this scheduler is then applied on MA M, it will remove the nondeterminism, leaving us with a stochastic process with paths that can be measured and assigned probabilites according to rates $\lambda$ and distributions from Distr($S$) in the MA. For these schedulers we are interested in some properties [19]:

- **Reachability probabilities:** Given a set of goal states $G \subseteq S$, we compute the probability of the set of paths that terminate at a state in $G$
- **Expected accumulated rewards:** We compute the expected value of the random variable that assigns to $\pi$ the value rew($\pi_{fin}$), where $\pi_{fin}$ denotes the shortest prefix of $\pi$ with a state in $G$.
- **Long-run average rewards:** We compute the expected value of the random variable that assigns to path $\pi$ the value

$$\lim_{i\to\infty} \text{rew}(\pi_{\leq i}/\text{dur}(\pi_{\leq i})$$

.

## 4.2 Model checking

For the purposes of this report, we want to find the optimal scheduler for a property of a model. The MODEST TOOLSET offers the **mcsta** tool for this purpose. **mcsta** is an explicit-state model checker. It evaluates the properties of schedulers as described in **Section 4.1** in the following ways:

- **Reachability probabilities and expected accumulated rewards:** These properties are evaluated by **mcsta** through the use of the value-iteration [15], linear programming, and interval iteration [5, 16] algorithms [10]. It also provides BRTDP as in [1], for which simulations with the uniform probabilistic scheduler are used to explore parts of the state space of the model. It runs a batch of simulation runs, then interval iteration is applied to compute the bounds [10].
- **Long-run average rewards:** These properties are evaluated by **mcsta** through the use of two algorithms: one based on a reduction to a linear program and another algorithm based on value-iteration [15] [10].

Through these evaluations the MODEST TOOLSET is able to process a MA and output an optimal scheduler. The input for this can contain multiple properties that are evaluated individually. The MODEST TOOLSET then produces a scheduler that describes per property which action to take in which state to get the optimal rewards for the model. It is also possible to use mcsta to only check the model for a specific property.

# 5. DTCONTROL

DTCONTROL is a comprehensive open-source tool for the post-processing of schedulers into compact and more interpretable representations [2]. It contains various decision tree learning algorithms that can be applied to several scheduler formats; (i) a raw comma-separated values (CSV) format with each row consisting of a vector of state variables concatenated with a vector of input variables; (ii) a sparse matrix format used by SCOTS; and (iii) the raw strategy produced by UPPAAL STRATEGO [2], PRISM, and STORM [3].

## 5.1 Decision Trees

A decision tree, or DT for short is a tuple $(T, \lambda, \rho)$ over the domain $X$, which is the set of states, with the set of labels $\mathcal{U}$, where $T$ is a finite full binary tree, which is a binary tree where every node has exactly 0 or 2 children), $\lambda$ assigns a label $u \in \mathcal{U}$, and $\rho$ assigns to every inner node, which is a node with exactly 2 children and is from now on referred to a decision node), of the tree a predicate, a boolean function which returns either 0 or 1 [2].
To use the decision tree, one needs to input a state $x$. The variables of state $x$ are then used in the decision nodes, and it will move through the DT. It starts at the root node $n_r$, it evaluates $\rho(n_r)$ and transitions based on the output of the function, where at the next node executes this step again until it arrives at a leaf node. Once one arrives at a leaf node $l$, the result of the decision tree will be the label $\lambda(l)$ as assigned to that node by $\lambda$.
For example, take a look at **Figure 1**. This figure represents the DT to find the optimal action a player should execute in a game of Pacman. In this case $U = \{0, 1, 2\}$ (0 is down, 1 is left, and 2 is the empty action) and X is the possible states the player can end up in. Let's say this DT is used to find out for a state $x = xG0 = 1, steps = 9, yG0 = 2$. First the root is evaluated, and as xG0 is bigger than 0.5, it travels down the transition labeled false. Next, $steps <= 9.5$ returns True, so it travels down the transition labeled True. Afterward $yG0 <= 2.5$ returns False, so it travels down the transition labeled False. Finally it ends up at a leaf node labeled **0**, so the action that should be chosen in this state is **0**.

## 5.2 Decision Tree Learning for representation of schedulers

The DT learning algorithms that are used by DTCONTROL follow the same underlying structure. For a finite set $C \subseteq X \times \mathcal{U}$ of feature-labeled pairs. This returns a DT that represents C precisely; so for every par $(x, u) \in C$ the leaf node for x has the label u. Here, C is a scheduler, x is a state, and u is an action [2].
To minimize a DT, the entropy of a scheduler C, denoted by `entr(C)`, should be minimized by splitting according to a predicate. So, for some $C \subseteq X \times \mathcal{U}$,

$$\texttt{entr(C)} := -\sum_{u \in \texttt{U}} p_u log(p_u)$$

where $p_u := \frac{|\{(x,u) \in C\}|}{|C|}$ is the empirical probability that label u is in C, and $|\cdot|$ is the cardinality of a set. [2].
The underlying algithm used works recursively in the following way:

- if `entrC` $= 0$, thus all state-label pairs share the same label we return a DT $(T, \lambda, \rho)$ where T has only 1 leaf node r, where $\lambda(r) = y$ and $\rho$ has no domain.
- `entrC` $\neq 0$, C will need to be split. C is split based on 1 predicate $P \in$ `PREDS`, where `PREDS` is the set of predicates for the scheduler. The predicate that minimizes the entropy of C after it is split will be picked. The perfect option in this case would be a predicate that splits the Decision Tree into 2 homogenous parts. After this the algorithm will recursively apply itself to the left subtree and the right subtree in the same manner [2].

In this research, we will be focussing on the MaxFreq preset. This preset processeses a DT by grouping together data points that share the same label.
For this purposes, dtControl can output the DT in the following formats:

- a `DOT`-file that can be converted into a visual representation of the DT.
- a `C`-file that contains a `classify()` function that is a functional representation of the DT.
- a `JSON`-file.

# 6. IMPLEMENTATION

The dataset-loader in the dtControl system is a class that retrieves the relevant information from a scheduler. To implement a new one, a new dataset-loader class extending the dataset-loader base class. For this extension, only one method needs to be implemented; `_load_dataset(self,file)`.

We have implemented such a dataset-loader class for schedulers produced by the Modest Toolset.
This dataset-loader requires two files; a scheduler file and a state-space file.
The state-space files begin by listing all variables in the model, so the dataset-loader gets all the variable names from here. For our implementation we took inspiration from the PRISM dataset-loader that is already implemented.
The scheduler file contains a list of state-action pairs for a property in the model. The action that is listed for each state is the optimal action to optimize the rewards. The dataset-loader goes through each state and collects the values for each variable. These values are then combined in a list that is appended to a NumPy array. Next for each state it collects the action for that state in a separate NumPy array. Then it specifies some metadata for both NumPy arrays. Finally, it returns the metadata and the two NumPy arrays.

When model-checking using the Modest Toolset it is possible to process multiple properties of a model at the same time. This results in a scheduler file that contains schedulers for multiple properties. Due to limitations in dtControl it is not possible to process multiple schedulers at once. This means that if the dataset-loader encounters a second property in a scheduler file it will stop processing the file and only use the data it has gathered so far. If a user still wants to use this scheduler file, they must move the schedulers to a separate file.
The Modest Toolset can process different kinds of properties. One of these properties checks the model for each value of a certain variable, like in the bitcoin-attack.modest model [19]. In this case it iterates over the values between 0 and 10000 for the remaining reward variable, so for each value of this variable a scheduler is generated. The dataset-loader evaluates this property by encoding this remaining reward as a variable like the other variables in the model. This implementation is used for the experiments as laid out in **Section 7**.

# 7. EXPERIMENTS

The models used for this are obtained from the Quantitative Verification Benchmark Set [20]. These experiments were executed on a 64-bit Windows 10 System running Python 3.8. For the us of storm the docker image `movesrwth/storm:travis` was used.

## 7.1 Checking Correctness

It is necessary to show that dtControl creates correct decision trees for the input schedulers, so we could guarantee correctness to future users. The approach taken for this was to utilize the decision trees implemented in C, as output by the dtControl processing using the MaxFreq preset. Next, we went through the scheduler generated by the Modest Toolset. For each state-action pair in the scheduler we used the classify() function from the decision tree `C`-file with the variables from the state, and check that the output action is the same as in the state-action pair. We did this for a number of schedulers. When finding no discrepancies, we have an indication that dtControl creates correct decision trees for schedulers produced by the Modest Toolset.

## 7.2 Checking Effectiveness

We wanted to evaluate the effectiveness of dtControl on making schedulers produced by the Modest Toolset, so we could find out if it made sense to apply dtControl to these schedulers. To check this effectiveness, the number of states in the scheduler was compared to the number of decision node in the decision tree generated from that scheduler. This was done for a number of models and we analysed the difference.
Next, schedulers for these same models were also generated using storm. These schedulers were then processed into a decision tree using dtControl. Next, we compared the resulting DT against the DT resulting from the scheduler produced by the Modest Toolset from the same model. This comparison shows the effectiveness of the processing of schedulers produced by the Modest Toolset relative to the processing of schedulers produced by storm. As the storm schedulers cannot be processd using MaxFreq, we instead used the default preset for both schedulers to make a fair comparison.

## 7.3 Used commands

For the generating of schedulers using the Modest Toolset we used the command:

```
modest mcsta <model-name>.<filetype> -E "<variable-
values>" --props <property(optional)> --scheduler
<model-name>.modest --statespace
<model-name>_states.modest
```

For the generating of schedulers using storm we used the command:

```
storm --jani <model-name>.jani
--janiproperty <property(optional)> --constants
<variable-values> --exact --timemem --buildstateval
```

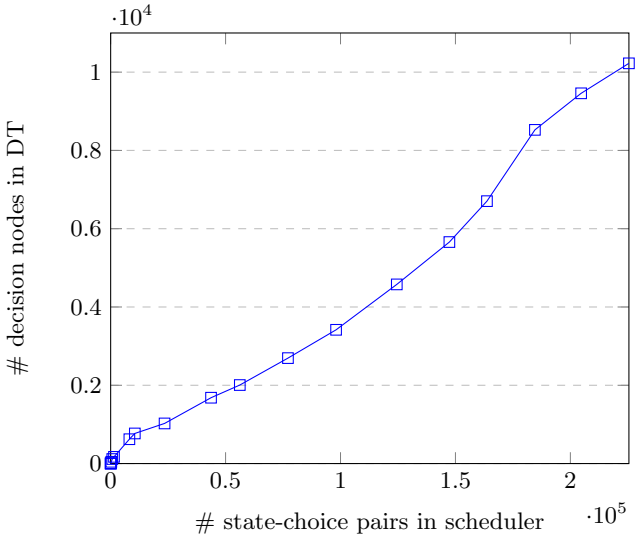


**Figure 3. This plot shows the relation between the number of state-choice pairs in the scheduler as generated by the Modest Toolset, and the number of decision nodes in the DT as processed by dtControl for the pacman.v1.jani model. In this plot the data of Table 1 is used.**

| Number of state-choice pairs in scheduler, and number of decision nodes in resulting DT using MaxFreq | | | | | |
|---|---|---|---|---|---|
| Input model | Input variables | Property evaluated | State-choice pairs in scheduler | Decision nodes in DT | # of errors |
| pacman.v1.jani | MAXSTEPS=5 | crash | 6 | 1 | 0 |
| pacman.v1.jani | MAXSTEPS=10 | crash | 21 | 2 | 0 |
| pacman.v1.jani | MAXSTEPS=15 | crash | 27 | 2 | 0 |
| pacman.v1.jani | MAXSTEPS=20 | crash | 101 | 4 | 0 |
| pacman.v1.jani | MAXSTEPS=25 | crash | 278 | 45 | 0 |
| pacman.v1.jani | MAXSTEPS=30 | crash | 702 | 116 | 0 |
| pacman.v1.jani | MAXSTEPS=35 | crash | 1458 | 171 | 0 |
| pacman.v1.jani | MAXSTEPS=40 | crash | 8140 | 622 | 0 |
| pacman.v1.jani | MAXSTEPS=45 | crash | 10516 | 769 | 0 |
| pacman.v1.jani | MAXSTEPS=50 | crash | 23459 | 1025 | 0 |
| pacman.v1.jani | MAXSTEPS=55 | crash | 43673 | 1682 | 0 |
| pacman.v1.jani | MAXSTEPS=60 | crash | 56192 | 2006 | 0 |
| pacman.v1.jani | MAXSTEPS=65 | crash | 77134 | 2694 | 0 |
| pacman.v1.jani | MAXSTEPS=70 | crash | 98075 | 3417 | 0 |
| pacman.v1.jani | MAXSTEPS=75 | crash | 124489 | 4577 | 0 |
| pacman.v1.jani | MAXSTEPS=80 | crash | 147364 | 5659 | 0 |
| pacman.v1.jani | MAXSTEPS=85 | crash | 163710 | 6704 | 0 |
| pacman.v1.jani | MAXSTEPS=90 | crash | 184577 | 8525 | 0 |
| pacman.v1.jani | MAXSTEPS=95 | crash | 204686 | 9459 | 0 |
| pacman.v1.jani | MAXSTEPS=100 | crash | 225471 | 10225 | 0 |
| bitcoin-attack.modest | MALICIOUS=0.1,CD=6 | P_MWinMax | 500000 | 26 | 0 |
| firewire_abst.v1.jani | delay=3 | rounds | 7 | 1 | 0 |
| firewire_abst.v1.jani | delay=3 | time_max | 68 | 1 | 0 |
| firewire_abst.v1.jani | delay=3 | time_min | 68 | 1 | 0 |

**Table 1. This table describes the results of the experiment as described in Section 6.1. Each model described is checked by the Modest Toolset according to the variables and property as described in the second and third column. The fourth and fifth column describe a comparison between the number of state-choice pairs in the scheduler and the number of decision nodes in the DT, obtained by making the scheduler more compact using the MaxFreq preset. The last column describes the number of errors encountered when checking if the DT is a correct representation of the scheduler.**

| Number of state-choice pairs in scheduler, and number of decision nodes in resulting DT using the default preset | | | | |
|---|---|---|---|---|
| Input model | Input variables | Property evaluated | Decision nodes in DT resulting from the MODEST TOOLSET | Decision nodes in DT resulting from STORM |
| pacman.v1.jani | MAXSTEPS=5 | crash | 1 | 21 |
| pacman.v1.jani | MAXSTEPS=10 | crash | 2 | 179 |
| pacman.v1.jani | MAXSTEPS=15 | crash | 2 | 840 |
| bitcoin-attack.modest | MALICIOUS=0.1,CD=6 | T_MWinMin | 4 | 8 |
| firewire_abst.v1.jani | delay=3 | rounds | 2 | 12 |
| firewire_abst.v1.jani | delay=3 | time_max | 2 | 16 |

**Table 2. This table describes a comparison between the processing of schedulers resulting from both the Modest Toolset and storm. It compares the number of decision nodes in the DT resulting from the processing using the default preset.**

```
--buildchoiceorig --exportscheduler <model-name>
.<variable-values>.<property(optional)>.storm.json
```

To then apply DTCONTROL to the scheduler by processing the schedulers generated by the MODEST TOOLSET using the MaxFreq prese we used the command:

```
dtcontrol --input <model-name>.modest --use-preset
maxfreq
```

and using the default preset:

```
dtcontrol --input <model-name>.modest --use-preset
default
```

To change models written using MODEST into a JANI file so STORM can model-check it we used the command:

```
modest moconv <model-name>.modest -OF JANI
-o <model-name>.jani
```

For the processing of the schedulers generated by STORM using the default preset we used:

```
dtcontrol --input <model-name>.storm.json --use-
preset default
```

# 8. RESULTS

## 8.1 Checking Correctness

For the results of the experiment as laid out in **Section 6.1**, we refer to **Table 1** for the results. For the verification of correctness, the last column is of interest. This column shows the number of errors in the DT when compared to the scheduler. To find this number we compared for each state which action the scheduler picked, and which one the DT picks, if these differ we count it as an error. As can be seen, none of the generated DT's contain any errors. This leads us to believe that DTCONTROL can correctly processed schedulers generated by the MODEST TOOLSET.

## 8.2 Checking Effectiveness

For the evaluation of effectiveness as laid out in **Section 6.2** we refer to **Table 1** for a benchmark of the processing of the schedulers generated by the MODEST TOOLSET. We observe that the number of decision nodes is always lower than the number of state-choice pairs.

For example, take pacman.v1.jani used with MAXSTEPS=100. Here, the number of decision nodes in the DT is about 20 times smaller than the number of state-choice pairs. It can be seen that the number of decision nodes is still large enough that the DT is not easily read. However, in slight of this fact we still consider the processing to vastly increase the readability of the scheduler.

It also shows that some models are processed better than others. This is very obvious for the bitcoin-attack.modest model, where the DT contains only 26 decision nodes, but the original scheduler has 500000 state-action pairs. This extreme difference can be attributed to the property that is used for this scheduler, as we discussed in the last paragraph **Section 7**. Due to this type of property the scheduler has a lot more state-choice pairs. Duplicate state-choice pairs also appear in this scheduler, only differentiating on the remaining reward. And due to these duplicate pairs DTCONTROL can easily group these together in order to make the DT more compact.

Overall, it can be concluded that the resulting DT's are much more managable than the scheduler representation from the output of the MODEST TOOLSET.

For deeper analysis of the pacman.v1.jani model, we refer to **Figure 3**, where for each value of MAXSTEPS the number of state-choice pairs is plotted against the number of decision nodes. This plot might suggest that for this model, the relation seems to be linear in nature. However, the shape of the plot at the latter half might indicate exponential as well. To get a more resounding answer on the trend of this plot more data needs to be collected. We suggest to decrease the step-size in the variation of MAXSTEPS and expanding it to go past MAXSTEPS = 100 until a clear trend arises.

For the comparison between the processing of models generated by the MODEST TOOLSET and STORM we refer to **Table 2**. As is visible in this table, the DT resulting from the MODEST TOOLSET scheduler contains considerably less decision node than the DT resulting from the STORM scheduler.

This result, however, had to be obtained by using the default preset, instead of the MaxFreq preset. This is because the MaxFreq was not applicable to the STORM schedulers. Due to this we can't say for sure that the findings are actually representable of the effectiveness of DTCONTROL on these schedulers. To be able to say anything on this topic there needs to be analasys of both the schedulers using the MaxFreq preset. So for now, nothing can be said about the effectiveness of DTCONTROL on schedulers produced by respectively the MODEST TOOLSET and STORM.

# 9. CONCLUSION

We have presented a dataset loader for the schedulers produced by the MODEST TOOLSET for the processing tool DTCONTROL. As shown in **Section 7** dtControl can correctly represent schedulers produced by the MODEST TOOLSET, so the answer to the first subquestion posed in **Section 2** is yes, DTCONTROL can correctly process schedulers produced by the MODEST TOOLSET. However, to answer the second subquestion in **Section 2** on how effective DTCONTROL is when making schedulers produced by the MODEST TOOLSET more compact compared to other types of datasets, more analysis needs to be performed. We recommend to focus on finding a way to apply the MaxFreq preset to the scheduler produced by STORM. To answer the main research question posed inm **Section 2**; yes, it can be concluded that DTCONTROL is an effective tool to use in making schedulers produced by the MODEST TOOLSET more compact, as the number of decision nodes is only a fraction of the number of state-action pairs in the original scheduler, and thus vastly increasing the readability for humans.

# 10. REFERENCES


[1] P. Ashok, Y. Butkova, H. Hermanns, and J. Křetínský. Continuous-Time Markov Decisions Based on Partial Exploration. In S. K. Lahiri and C. Wang, editors, *Automated Technology for Verification and Analysis*, pages 317–334, Cham, 2018. Springer International Publishing.

[2] P. Ashok, M. Jackermeier, P. Jagtap, J. Křetínský, M. Weininger, and M. Zamani. DtControl: Decision Tree Learning Algorithms for Controller Representation. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, HSCC '20, New York, NY, USA, 2020. Association for Computing Machinery.

[3] P. Ashok, M. Jackermeier, J. Křetínský, C. Weinhuber, M. Weininger, and M. Yadav. dtControl 2.0: Explainable Strategy Representation via Decision Tree Learning Steered by Experts. In J. F. Groote and K. G. Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 326–345, Cham, 2021. Springer International Publishing.

[4] P. Ashok, M. Jackermeier, C. Weinhuber, and M. W. Revision. dtControl Developer Manual. [Online; accessed 30-April-2021].

[5] C. Baier, J. Klein, L. Leuschner, D. Parker, and S. Wunderlich. Ensuring the Reliability of Your Model Checker: Interval Iteration for Markov Decision Processes. In R. Majumdar and V. Kunčak, editors, *Computer Aided Verification*, pages 160–180, Cham, 2017. Springer International Publishing.

[6] A. Bohy, V. Bruyère, and J.-F. Raskin. Symblicit algorithms for optimal strategy synthesis in monotonic Markov decision processes. *Electronic Proceedings in Theoretical Computer Science*, 157, 07 2014.

[7] T. Brázdil, K. Chatterjee, M. Chmelik, A. Fellner, and J. Kretínský. Counterexample Explanation by Learning Small Strategies in Markov Decision Processes. In *Proceedings of the 27th International Conference on Computer Aided Verification, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 158–177. Springer International Publishing, 2015.

[8] T. Brázdil, K. Chatterjee, J. Křetínský, and V. Toman. Strategy Representation by Decision Trees in Reactive Synthesis, 2018.

[9] C. E. Budde, C. Dehnert, E. M. Hahn, A. Hartmanns, S. Junges, and A. Turrini. JANI: quantitative model and tool interaction. In A. Legay and T. Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part II*, volume 10206 of *Lecture Notes in Computer Science*, pages 151–168, 2017.

[10] Y. Butkova, A. Hartmanns, and H. Hermanns. A Modest Approach to Modelling and Checking Markov Automata. In D. Parker and V. Wolf, editors, *Proceedings of the 16th International Conference on Quantitative Evaluation of Systems (QEST 2019)*, Lecture Notes in Computer Science, pages 52–69, Netherlands, 2019. Springer. 16th International Conference on Quantitative Evaluation of Systems, QEST 2019, QEST 2019 ; Conference date: 10-09-2019 Through 12-09-2019.

[11] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk. A storm is Coming: A Modern Probabilistic Model Checker, 2017.

[12] C. Eisentraut, H. Hermanns, and L. Zhang. On Probabilistic Automata in Continuous Time. *2010 25th Annual IEEE Symposium on Logic in Computer Science*, pages 342–351, 2010.

[13] A. Graf-Brill, A. Hartmanns, H. Hermanns, and S. Rose. Modelling and certification for electric mobility. *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pages 109–114, 2017.

[14] D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer. Modelling, Reduction and Analysis of Markov Automata. In K. Joshi, M. Siegle, M. Stoelinga, and P. R. D'Argenio, editors, *Quantitative Evaluation of Systems*, pages 55–71, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[15] D. Guck, M. Timmer, H. Hatefi, E. Ruijters, and M. Stoelinga. Modelling and Analysis of Markov Reward Automata. In F. Cassez and J.-F. Raskin, editors, *Automated Technology for Verification and Analysis*, pages 168–184, Cham, 2014. Springer International Publishing.

[16] S. Haddad and B. Monmege. Interval Iteration Algorithm for MDPs and IMDPs. *Theoretical Computer Science*, 735:111 – 131, July 2018.

[17] A. Hartmanns. MODEST - A unified language for quantitative models. *Proceeding of the 2012 Forum on Specification and Design Languages*, pages 44–51, 2012.

[18] A. Hartmanns and H. Hermanns. The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification. In E. Ábrahám and K. Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 593–598, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[19] A. Hartmanns and H. Hermanns. A Modest Markov Automata Tutorial. In M. Krötzsch and D. Stepanova, editors, *15th International Reasoning Web Summer School on Explainable Artificial Intelligence (RW 2019)*, Lecture Notes in Computer

Science, pages 250–276. Springer, 2019. 15th International Summer School 2019 ; Conference date: 20-09-2019 Through 24-09-2019.

[20] A. Hartmanns, M. Klauck, D. Parker, T. Quatmann, and E. Ruijters. The Quantitative Verification Benchmark Set. In *Proc. 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'19)*, volume 11427 of *LNCS*, pages 344–350. Springer, 2019.

[21] B. Lewis, A. Hartmanns, P. Basu, R. Shridevi, K. Chakraborty, S. Roy, and Z. Zhang. Probabilistic Verification for Reliable Network-on-Chip System Design. In K. Larsen and T. Willemse, editors, *Formal Methods for Industrial Critical Systems*, Lecture Notes in Computer Science, pages 110–126. Springer, Aug. 2019. 24th International Conference on Formal Methods for Industrial Critical Systems, FMICS 2019, FMICS 2019 ; Conference date: 30-08-2019 Through 31-08-2019.

[22] A. Miner and D. Parker. *Validation of Stochastic Systems: A Guide to Current Research*, volume 2925 of *Lecture Notes in Computer Science (Tutorial Volume)*, chapter Symbolic Representations and Analysis of Large Probabilistic Systems, pages 296–338. Springer, 2004.

[23] R. Wimmer, B. Braitling, B. Becker, E. M. Hahn, P. Crouzen, H. Hermanns, A. Dhama, and O. Theel. Symblicit Calculation of Long-Run Averages for Concurrent Probabilistic Systems. In *2010 Seventh International Conference on the Quantitative Evaluation of Systems*, pages 27–36, 2010.

[24] A. Wolfe and A. Barto. Decision tree methods for finding reusable mdp homomorphisms. *Science*, 01 2004.