Comparing Different Pruning Strategies for the Evaluation Task of Virtual Stochastic Sensors

Dávid Bodnár*, Claudia Krull

Institut für Simulation und Graphik, Otto-von-Guericke-Universität Magdeburg, Universitätsplatz 2, 39106 Magdeburg, Germany; **david.bodnar.ovgu@gmail.com*

SNE 35(1), 2025, 41-48, DOI: 10.11128/sne.35.tn.10727
Selected ASIM SST 2024 Postconf. Publication: 2024-12-10
Received Revised: 2025-02-28; Accepted: 2025-03-03
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna
ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. Virtual Stochastic Sensors calculate statistically relevant estimates in indirectly observable discrete stochastic systems. This is done by the proxel-based analysis that aims to reconstruct the relevant part of the state space with an iterative process. Strategically removing non-relevant proxels from the analysis (pruning) to reduce runtime overhead might potentially affect the results. And while the impact on the decoding problem has already been analysed in detail, the effect on the evaluation problem was not yet discussed.
The paper discusses three pruning strategies and compares their properties in case of the evaluation task. The theoretical statements are empirically proven using a car rental agency model in form of a Conversive Hidden non-Markovian Model.
The results show that in case of well chosen parameters all three pruning strategies are able to reach the same accuracy. The major difference between the strategies is due to their runtime properties which need to be carefully aligned with the use-case to reach optimal behavior. Based on the results the *fixed number of proxels pruning* strategy provides highly predictable execution time, while the *fixed threshold pruning* is very good at discovering a broader spectrum of the state space. The *variable pruning* is a very good trade-off between the previous strategies enabling lower thresholds and thorough state space analysis while maintaining acceptable execution times at the cost of more complex parametrisation.

Keywords: Virtual Stochastic Sensor, Hidden non-Markovian Model, Proxel-based Simulation, Pruning, Evaluation

Introduction

Virtual Stochastic Sensors (VSSs), introduced in [1], utilize the proxel-based analysis [2] to analyse partially observable discrete stochastic systems. The proxels, which represent a given system state at a given point in time, build a so-called proxel tree to reconstruct and represent the relevant part of the state space during the analysis. The insignificant part is pruned away.

There are different pruning strategies to define what is insignificant. In this paper three of them, *fixed threshold pruning*, *fixed number of proxels pruning* and *variable pruning*, will be discussed in detail. Similarly to [3] we aim to give an overview of the influence of the different pruning strategies on the evaluation results of VSSs. This is motivated by the generalisation and further development of the Change Adaptation Algorithm (CAA) described in [4].

As in [3], this paper utilizes the same car rental service model, presented in [5], for the evaluation. In this model, customers arrive in a premium or ordinary queue based on their membership. They use the same door for entering and leaving the shop area and this door is being observed (opening creates a signal) by the analysis. A single employee is serving both queues. Premium customers have priority over ordinary ones, but a customer is always served to the end if the processing has been started. Both queues are limited to 50 customers. Figure 1 shows the Augmented Stochastic Petri Net (ASPN) [5] of the system. ASPNs are modified Petri nets [6] that can model the emission of symbols when a transition fires. This model will also be used as an example in the following to demonstrate the different concepts and tools used in this paper.

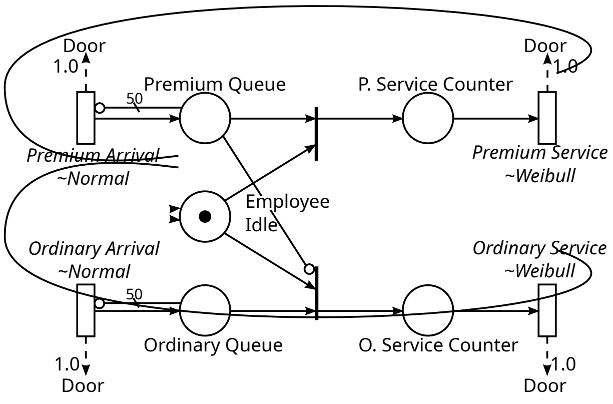


Figure 1: Car rental service Example as ASPN.

This paper compares three pruning strategies in theory and using the above presented experiment for the evaluation problem of Conversive Hidden non-Markovian Models (CHnMMs). The results show that after a given threshold there is no significant difference in the resulting probabilities, however, other major properties vary for each strategy so choosing one or the other for a given problem will always depend on the use-case.

1 Related Work

Virtual Sensors (VSs) [7] aim to collect system information that is hard or expensive to obtain in a direct way. If we combine them with stochastic processes a so-called VSS can be constructed which calculates statistically relevant estimates of system paramters that are similarly hard, inefficient and/or expensive to measure.

In this section, an introduction will be provided to such a stochastic process, the so-called CHnMM [8] and its solution algorithm, the proxel-based analysis. Additionally, proxel merging will be discussed.

1.1 Conversive Hidden non-Markovian Model

The concept of CHnMMs originates from the well-known Hidden Markov Models (HMMs) [9], where a hidden or partially hidden system is analyzed by probabilistic symbol emissions. This idea was extended to Hidden non-Markovian Models (HnMMs) by [10] to overcome the limitations of the discrete-time Markov chain in the background. HnMMs use arbitrary continuous distribution functions to describe the state changes and to create time dependence between them.

Similarly to HMMs, the HnMMs also try to solve the evaluation and the decoding problem. The decoding problem, finding the most probable generator state sequence to a given trace, and the impact of the different pruning strategies on it have already been discussed in [3]. This paper is focusing on the evaluation problem, e.g. finding the probability that a trace has been generated by a given model.

A specific subclass of HnMMs are the so-called CHnMMs which allow additional performance optimizations of the algorithm due to the fact that every state change results in an observable symbol emission. Those optimizations are powerfull enough to make CHnMMs the perfect experiment environment to test and verify new ideas in an efficient way. That is why this paper also limits its scope to CHnMMs.

In case of the presented car rental shop the hidden internal system state is the length of (the number of customers in) the ordinary and premium queues. Every state change (a customer entering or leaving the queue) results in a door opening (signal emission). This effectively means that by solving the evaluation problem, the goal of this project is to find the probabilty of a given door-opening signal protocol.

1.2 Proxels-based analysis

CHnMMs also need a solution algorithm. However, instead of using the forward algorithm [11] or the Viterbi algorithm [12] the so-called proxel-based analysis [2] can be used. The technique utilizes the encapsulation of a possible system state description (system state m, age vector $\overrightarrow{\tau}$, the probability of that state p, a timestamp t, etc.) into a so-called proxel object as it can be seen in Equation 1. One can use a collection of those proxels to describe all possible system states at a given point in time, including the ages of the relevant non-Markovian transitions. Then by creating a parent-child relationship between the timesteps one can derive the possible system states for the future timesteps in an iterative way. Of course, the proxel definition can be extended to carry additional information through the analysis. Refering back to the example model, a specific proxel represents a given number of customers in the premium and ordinary queues (m), the durations since the last customers entered each queue and while one is being served ($\overrightarrow{\tau}$) and the probability (p) of ending up in the represented state at the current simulation time (t).

$$P_x = \left(m, \overrightarrow{\tau}, p, t\right) \tag{1}$$

The mentioned parent-child relation is described by the Hazard Rate Function (HRF) in Equation 2 using the Probability Density Function (PDF) $f(\tau)$ and the Cumulative Distribution Function (CDF) $F(\tau)$ of a possible state change. The equation describes the current state change rate for a given state change if this one has been active for τ and has not happened yet. So in case of our example model this translates to for example, what is the probability of a customer service being finished in this timestep, if he/she is being served right now. Or similarly, what is the probability of the next customer entering the shop now?

$$H(\tau) = \frac{f(\tau)}{1 - F(\tau)} \tag{2}$$

To keep the proxel tree at a reasonable size, very unlikely proxels can be pruned from the tree. Different strategies exist to perform this operation. They will be further discussed in Section 2.

Proxel probabilities are represented on a logarithmic scale to preserve precision as in case we represent them on normal scale, they would disappear due to arithmetic underflow very soon after the analysis was started.

1.3 Merging proxels

In case of the evaluation problem, we are not interesed in the history of a given proxel, because we are only trying to compute the probability of a given system state at time t. This means that after a couple of timesteps after starting the proxel-based analysis we can find proxels in the proxel tree, that represent the same system state, but they have been generated by different routes. These proxels can be merged by adding their probabilities (which are logarithmic probabilities, as stated before, represented by $\tilde{p}_1$ and $\tilde{p}_2$, where $\tilde{p}_1 > \tilde{p}_2$) using the Kingsbury-Rayner formula [9] shown in Equation 3. By performing this merging operation one can keep the size of the proxel tree under control very efficiently.

$$\tilde{p}_1 +_{log} \tilde{p}_2 = \tilde{p}_1 - \ln\left(1 + e^{-(\tilde{p}_2 - \tilde{p}_1)}\right) \tag{3}$$

The Kingsbury-Rayner formula additionally provides an opportunity to efficiently add probabilities that are on a different magnitude.

An example of a merging opportunity is presented in Figure 2, where the discovered state space of a three state system ($m0$, $m1$, $m2$) is reperesented over three timesteps. The two proxels marked with red at $t = 2\Delta$ represent the same internal system state with different probabilities but they were reached through different routes, so they can be merged.

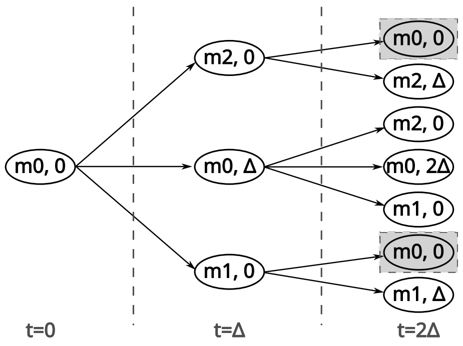


Figure 2: Example of mergable proxels.

As stated before, in case of the evaluation problem only the current system state is a subject of interest. This means, that in many cases different customer enter sequences (P - premium customer, O - ordinary customer), for example "PPOPO" and "POPPO", can be represented by a single common proxel.

Merging significantely reduces the number of existing proxels over time. This also means that much less strict pruning configurations can be used than for example in case of the decoding task. These will be shown later in Section 3.2.

Until now, we have introduced CHnMMs and the proxel-based analysis as two base concepts of our research. We also gave a brief overview about the possibility of merging as the main method to keep the proxel tree under control in case of the evaluation problem. In the next section the concept of pruning and the different pruning strategies will be introduced as the main reseach interest of this paper.

2 Pruning Strategies for Evaluation

Pruning is a concept of the proxel-based analysis, in which the algorithm classifies a part of the proxel tree as irrelevant for further analysis and removes it. The different strategies encapsulate a set of logical steps to be performed on the proxel tree in order to identify and prune the irrelevant proxels.

There are two major events that can render a proxel based analysis infeasible. One of them is state space explosion, when the proxel tree grows exponentially and reaches a state where the next step is enormously expensive to compute or the ressources of the computer running the analysis are fully consumed. The other one is the exact opposite, the proxel tree can die out. In this case the remaining proxels, for example due to extreme pruning, encounter an observed symbol that cannot be emitted by their current system state and with that the proxel becomes impossible and is removed from the analysis. If all the proxels are removed from the tree, the tree dies out. Merging and a pruning strategy needs to be devised to prevent both of these extreme cases from happening

As already mentioned at the end of the previous section, merging does an excellent job at keeping the proxel tree under control. But sooner or later the analysis reaches the point where additional intervention is needed in the form of pruning.

Still, merging makes it possible to use less strict pruning strategies than in case of the decoding problem [3]. This also results in the fact that state space explosions or died out proxel trees are not common for simple models. However, with increasing system complexity a state space explosion might occur, but it is extremely unlikely.

In this section a brief overview will be given of the three investigated pruning strategies. But before diving into the details, we should list the properties of a good pruning algorithm.

These are in case of the evaluation problem in our experience, the following:

1. High proxel processing throughput
2. Small amount of lost (pruned) probabilty
3. Prevents state space explosion
4. Scales the proxel tree up and down depending on how „interesting” the current part of the trace is
5. Guarantees predictable execution times
6. Is easily customizable for different needs
7. Is overall robust, which means that it does not react very differently to similar traces

These criteria will be used to evaluate the different pruning strategies.

2.1 Fixed Threshold Pruning

The *fixed threshold pruning* is a simple concept defining a pruning probability threshold $p(P_{pruned,t_i})$. Below that value every proxel is considered to be irrelevant and is then removed from the proxel tree.

The threshold is defined compared to the probability of the most probable proxel $\max(p(P_{x,t_i})$ in the proxel tree for a given time t_i. It is described by a ratio r as it can be seen in the Equation 4. The parameter r is a freely selectable value with the limits $0 < r \leq 1$.

$$p(P_{pruned,t_i}) < r \max(p(P_{x,t_i})) \qquad (4)$$

This pruning strategy is the most vulnerable to state space explosions and other instabilities as described in [3]. However, combined with merging it is possible to use extremely low pruning thresholds, as it will be shown in Section 3.2.

This strategy scales the proxel tree overall well and as a simple algorithm, it provides a high proxel processing throughput. But due to the unpredictable number of proxels in the proxel tree, the execution time has a high fluctuaction when the threshold is low.

2.2 Fixed Number of Proxels Pruning

The *fixed number of proxels pruning* is another simple strategy for keeping, as the name suggests, only a predefined number of proxels at the end of every timestep keeping the higher probability proxels and pruning the less likely ones. This property is the major advantage of the strategy, as it is very easy to parametrize and very robust against disturbances. However, the algorithm itself has some drawbacks.

The optimal proxel storage in case of the evaluation problem is a hash table [13], because due to the merging described in Section 1.3 one wants to retrive proxels as efficiently as possible. The cost of it is the hash tables's average search complexity, $O(1)$. However, a hash table cannot be sorted, so in order to perform the pruning, one needs to put all the proxels into a sorted array-like structure which is an additional overhead to the already complex sorting operation of $O(n \log(n))$ [13].

Of course, one can perform some implementation tricks, like storing a pointer to the proxel in the proxel storage instead of the object itself, to speed up the moving and sorting operation significantly. But in the end, one still needs to perform the moving and sorting of the entries and with a potentially higher number of proxels to keep, these can become too expensive for the analysis. Similarly, not all tricks might be universally available in every programming language.

This strategy provides a simple single parameter customization and very predictable execution times, but it fails to scale the proxel tree, so one really needs to find the perfect parameter with this strategy before running a long-time analysis.

2.3 Variable Pruning

Variable pruning was introduced in [4] due to the limitations of the *fixed threshold pruning* in case of the decoding problem. It is basically combining the previously discussed pruning strategies by creating a relationship between the current number of proxels in the proxel tree and the pruning threshold.

This is done by defining a minimum number of proxels (r_{min}) - pruning threshold ($\#P_{min}$) pair, which prune the really unlikely proxels. Additionally, one selects a maximum number of proxels (r_{max}) - pruning threshold ($\#P_{max}$) pair to drastically prune the tree if the proxel tree becomes too large. The two points must be connected by a strictly monotonically increasing continuous function ($r(\#P_{x,t_i})$) to guarantee a smooth transition between the two behaviors. The definition is shown in Equation 5:

$$r = \begin{cases} r_{min} & \text{if } \#P_{x,t_i} < \#P_{min} \\ r(\#P_{x,t_i}) & \text{if } \#P_{min} \leq \#P_{x,t_i} \leq \#P_{max} \\ r_{max} & \text{if } \#P_{x,t_i} > \#P_{max} \end{cases} \quad (5)$$

Even though the strategy is slightly more complex than the previous strategies due to the high factor of customization (different minumum and maximum pairs, different equations), it is a good trade-off between the advantages and disadvantages of the previous strategies. One can use overall much lower pruning thresholds while maintaining high throughput, lower risk of state space explosion and reducing the fluctuation of execution time compared to the *fixed threshold pruning*.

3 Experiments

In this section, the previously discussed properties of the different pruning strategies will be shown in an empirical way. First the experiment setup and the parametrization will be briefly discussed before describing the experiment results in details.

3.1 Experiment Setup

A car rental service, presented in the introduction, was fed with the same input data as presented in [3] previously to make the result easily comparable.

A Personal Computer (PC) equipped with an AMD Ryzen 7 3800X and 64 GB of RAM has been used for the experiment execution. The RAM was sufficient to prevent swapping, which made the experiments performed easily comparable. The implementation code utilized the C++20 standard and it was compiled using GCC 11.4.0 with the highest optimization level enabled. The application was containerized using Docker. The PC was running Manjaro Linux with the kernel 6.1.69.

3.2 Parameter Selection

The experiment has been run 1620 times for every given pruning strategy with a defined parametrization, which includes 162 different model parameters with 10 randomly generated traces each. These models include only stable models, further discussed in [5].

The different pruning strategies were parametrized based on different logic. The *fixed threshold pruning* strategy (abbreviated with „th" in the following figures) was tested with various thresholds in the range of $[1e-1, 1e-150]$. Similarly, a wide range of sizes between $[50, 2500]$ were used to test the *fixed number of proxels pruning* strategy (abbreviated with „size" in the following figures). For the *variable pruning* (abbreviated with „var" in the following figures) five different equations were used, all with the minimum threshold (r_{min}) of $1e-300$ for proxel tree sizes below 1000 ($\#P_{min}$), and a maximum threshold (r_{max}) of 0.1 above proxel tree size of 100000 ($\#P_{max}$). All equations describe a mapping between the pruning threshold and the logarithm of the current tree size. We are omitting the equations here due to space reasons.

Figure 3 visualizes the previous equations between the previously described minimum and maximum pairs to make them easier to follow for the reader. Please be aware that the x and the y axis are using logarithmic scale.

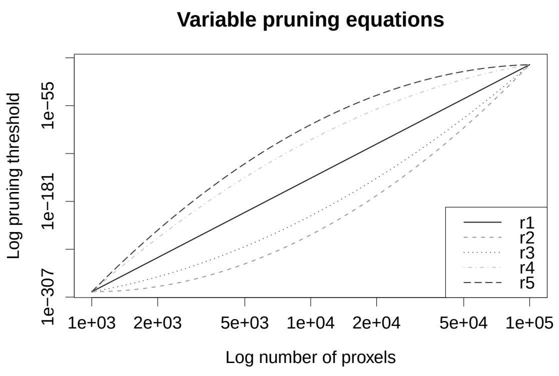


Figure 3: Variable pruning equations.

The different equations make it possible to scale the *variable pruning* with different „speed" between the minimum and maximum values. As the results in the next subsection will show, this was already enough to reach significantly different behavior.

3.3 Experiment Results

For easier representation, the experiment results were visualized in the Figures 4 - 6 in a unified way. Yellow background („th" on the X-axis) marks the *fixed threshold pruning* strategies with different thresholds in scientific notation. Green background („size" on the X-axis) marks the *fixed number of proxels pruning* strategies with different fixed proxel tree sizes, while blue background („var" on the X-axis) marks the *variable pruning* strategies with different equation IDs.

During the tests, we did not experience any state space explosions, even with extremely small pruning thresholds. This means that the merging efficiently eliminates this problem, however, this is not a proof that in case of a more complex model we would not experience any problems with the *fixed threshold pruning* strategy, as this is the most vulnerable strategy regarding state space explosions.

Figure 4 visualizes the performance criteria of the experiment. Here we see that the *fixed threshold pruning* needs to deal with an extreme uncertainty regarding the number of proxels with decreasing pruning threshold. This leads to a similar fluctuation/variation in the execution time. The *variable pruning* efficiently copes with this problem while using a significantly lower minimal threshold. The *fixed number of proxels pruning* maintains predictable and low execution times.

In case of the proxel processing efficiency (Figure 4, top right) the *variable pruning* and the *fixed threshold pruning* with lower thresholds outperform the *fixed number of proxels pruning* by about $25 - 30\%$. This results from the fact that sorting the proxel tree is an operation that is hard to parallelize.

The last graph in the Figure 4 visualizes the probability lost through pruning. Please be aware that the probabilities are visualized on a logarithmic scale. One would like to minimize that in order to get the possibly most complete analysis of the state space. However, the first steps in case of the analysis play a crucial role in this case, because the proxel probabilities are decreasing drastically over the analysis time domain. This means that the proxels pruned away first basically determine the amount of probability lost. As we can see in the picture, the *fixed number of proxels pruning* has a really high spread for these values.

This shows that it could potentially throw away important proxels. The *fixed threshold pruning* copes with the problem well at lower pruning thresholds.

The *variable pruning* provides the best values with somewhat higher spread than the *fixed threshold pruning*. This is the indicator of being a good trade-off between the two other techniques.

Evaluation tries to compute the probability that a given trace has been generated by a given model. Interestingly enough, most of the strategies provided the same result to that question as it can be seen in Figure 5. Only the *fixed threshold pruning* strategies with the thresholds $1e-1$ and $1e-2$ failed to reach the same results, but in these cases only 1 or 2 proxels survived the timesteps on average. In case of the fixed pruning threshold $1e-10$, which is the first test case that came to the common solution, on average about 35 proxels survived the timestep after pruning, so we expect that a *fixed number of proxels pruning* with values under 50 could have also reached this result.

Not being able to reach a better result by processing more proxels has to do with the fact that the proxel probabilities are represented on a logarithmic scale. At the end of the analysis a subset of proxels became dominant (they had significantly higher probabilities) over the remaining proxels. That part of the proxel tree computationally defined the results.

This means that there is a sweet spot in the computation and with an optimal number of proxels one is able to compute the end result of the analysis very efficiently. In our case this sweet spot is somewhere between $35 - 50$ proxels. However, this cannot be stated universally, because more complex systems might have a higher optimal proxel number. Generally said, with a threshold based pruning strategy one can get to the optimum more easily than with the *fixed number of proxels pruning* strategy, because it is very hard to find the optimal number of proxels without performing multiple experiments.

Another important aspect is to keep enough proxels in the tree to support diversity and to prevent the proxel tree from dying out if something very unexpected happens, for example due to very strong pruning all the existing proxels become impossible in the next timestep. This quality can be visualized with the pruning ratio, so which amount of the proxels are kept on average after the pruning step. This can be seen in Figure 6.

Here we see that the strategies that failed to reach the common probability result have thrown away more than 50% of the proxels on average from timestep to timestep.

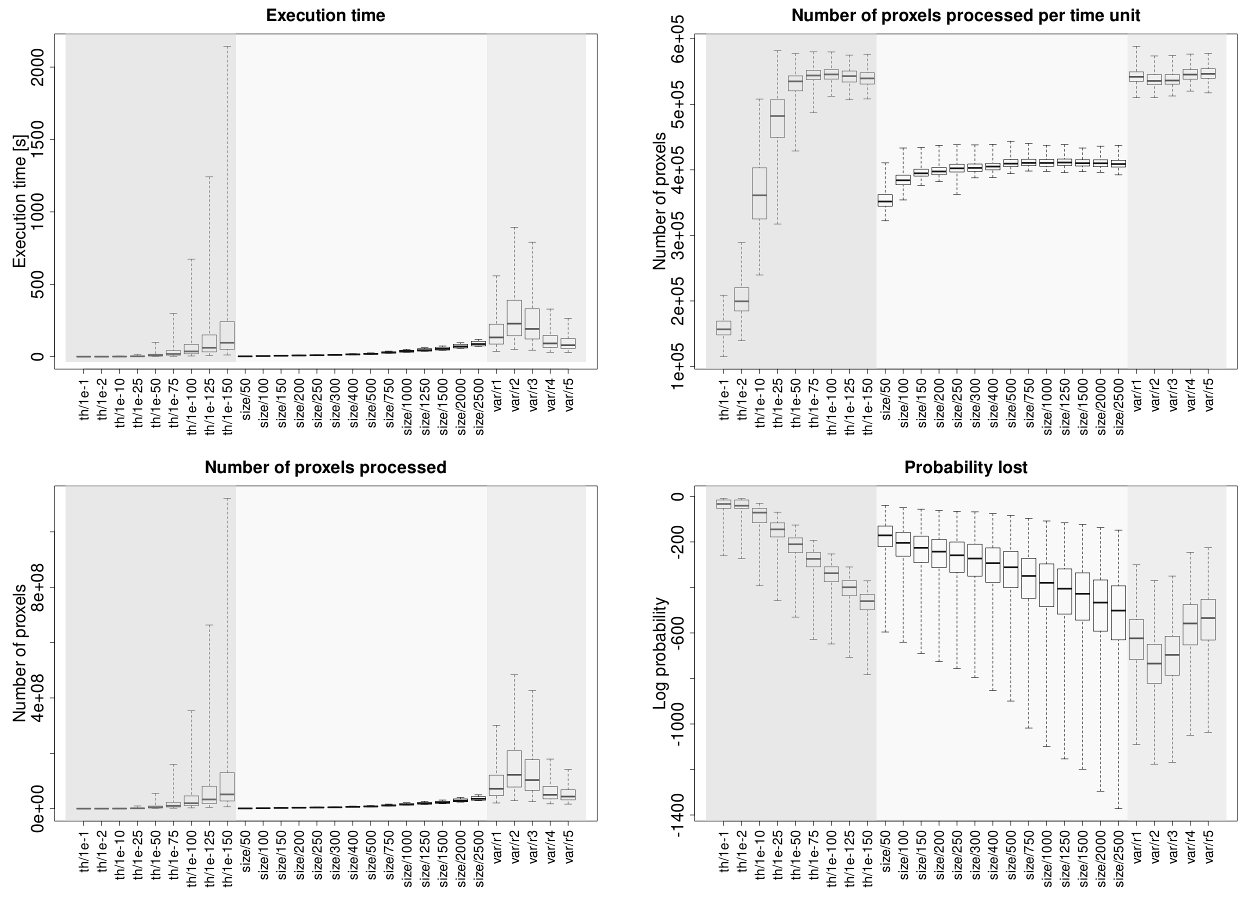
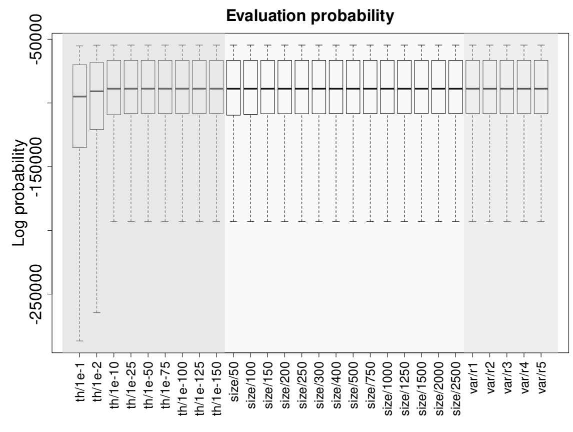


Figure 4: Quality Measures of Different Pruning Strategies.
Note: Yellow background („th" on the X-axis) marks the *fixed threshold pruning* strategies with different thresholds.
Green background („size" on the X-axis) marks the *fixed number of proxels pruning* strategies with different sizes.
Blue background („var" on the X-axis) marks the *variable pruning* strategies with different equations.



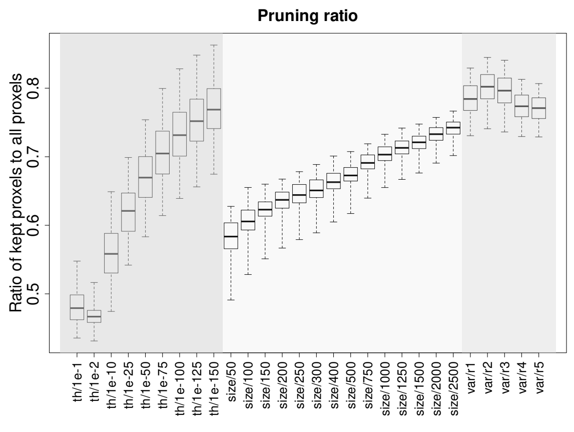


Figure 5: Probability of a given door protocol.
Note: Information about the colors and abbreviations can be found in the note below the caption of Figure 4.

Figure 6: Pruning rate with different prunings.
Note: Information about the colors and abbreviations can be found in the note below the caption of Figure 4.

Generally, this is caused by the relatively small number of proxels. Of course, with the increasing number of proxels the different strategies are keeping a higher amount of proxels. This generally means also higher fault tolerance toward very unlikely events in the proxel-based analysis and also indicates higher robustness for similar traces.

4 Discussion & Conclusion

The goal of this paper was to give a general overview of different pruning strategies for the evaluation problem and to compare their properties.

All three of the analyzed pruning strategies are suitable for the evaluation problem. Problems like state space explosion or dying out proxel trees are very unlikely, as merging keeps the proxel tree under control enabling the use of less strict pruning strategies compared to the decoding problem [3]. We have not encountered them during our experiments. This generally also shows that the pruning strategy has a smaller impact on the results in case of the evaluation problem, as long as they are parametrized in a reasonable way.

There is generally an optimum where the proxel-based analysis reaches the final evaluation result with a minimum number of proxels. However, finding this sweet spot is not trivial and it surely requires multiple test runs, which is not always possible in the real world.

In case of all three strategies the parameters need to be chosen carefully. But with a good rule of thumb solution like „one should keep at least $50 - 100$ proxels on average, but one should aim for more” one should be on the safe side to get a good general solution with acceptable execution times. However, for more complex systems a higher number of proxels might be needed.

The *fixed number of proxels pruning* performs good enough to be a robust solution if one values highly predictable execution times over other properties. However, the proxel processing throughput might be a limiting factor. If the desired property of the analysis is to explore the state space as thoroughly as possible, one should use the *fixed threshold pruning* or tweak the *variable pruning* to speed up the execution time and get some additional advantages, like less lost probability, more diverse proxel tree, etc.

From a practical point of view there is no real limitation that would prevent the user from utilizing any of these pruning strategies in an experiment with artificial data or in real world use-cases.

References

[1] Krull C, Buchholz R, Horton G. Virtual Stochastic Sensors: How to gain insight into partially observable discrete stochastic systems. *The 30th IASTED International Conference on Modelling*. 2011.

[2] Lazarova-Molnar S. *The Proxel-Based Method: Formalisation, Analysis and Applications*. Magdeburg: Otto-von-Guericke-Universität. 2005.

[3] Bodnár D, Krull C. Comparing Different Pruning Strategies for the Decoding Task using Virtual Stochastic Sensors. In: *The European Simulation and Modelling Conference 2023*, eds Vingerhoeds R, de Saqui-Sannes P. Toulouse: EUROSIS-ETI. 2023; pp. 37–42.

[4] Bodnár D, Krull C. Adapting to Change of Model Transitions in Proxel Based Simulation of CHnMMs. In: *Proceedings Langbeiträge ASIM SST 2022, 26. Symposium Simulationstechnik*, edited by Breitenecker F. Vienna: TU Wien. 2022; pp. 101–108.

[5] Krull C. *Virtual Stochastic Sensors: Formal Background and Example Applications: Reconstructing the Behavior of Partially Observable Discrete and Hybrid Stochastic Systems*. Shaker. 2021.

[6] Bobbio A, Puliafito A, Telek M, Trivedi KS. Recent Developments in Non-Markovian Stochastic Petri Nets. *Journal of Systems Circuits and Computers*. 1998; 8(1):119–158.

[7] Wilson E. Virtual sensor technology for process optimization. 1997. Presentation at the ISSCAC 1997.

[8] Buchholz R. *Conversive Hidden non-Markovian Models*. Magdeburg: Otto-von-Guericke-Univ. 2012.

[9] Fink GA. *Markov Models for Pattern Recognition*. London: Springer London. 2014.

[10] Krull C, Horton G. Hidden non-Markovian Models: Formalization and solution approaches. *6th Vienna Intern. Conference on Mathematical Modelling*. 2009.

[11] Rabiner L. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*. 1989;77(2):257–286.

[12] Viterbi A. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *Information Theory, IEEE Transactions on*. 1967; 13:260 – 269.

[13] Skiena SS. *The Algorithm Design Manual*. Cham: Springer International Publishing. 2020.

Publication Remark. This contribution is the revised version of the conference version published in *Tagungsband Langbeiträge ASIM SST 2024*, ARGESIM Report AR 47, ISBN ebook: 978-3-903347- 65-6, p 113-120, DOI: 10.11128/arep.47.a4704