# EFFICIENT STATE SPACE-BASED SIMULATION: AVOIDING REDUNDANCIES IN THE PROXEL METHOD

Robert Buchholz
Claudia Krull
Graham Horton

Otto-von-Guericke University
P.O. Box 4120
39016 Magdeburg, Germany
email: {robert|claudia|graham}@isg.cs.uni-magdeburg.de

**KEYWORDS**
Model analysis, Model evaluation, Approximation techniques, Markov chain, Discrete stochastic model

**Abstract**

The simulation of discrete stochastic systems is an important tool to make predictions on possible system behavior in science and industry. The most widely used simulation technique, the discrete event simulation, computes possible simulation results by using random numbers. Consequently, the simulation results are also only random numbers. Alternative state space-based simulation techniques can directly compute the actual system behavior. They are, however, computationally infeasible for most bigger models.
In this paper, we present a modification to one promising state-space simulation technique, the Proxel method. Our modification, called MultiProxel, uses a clustering approach on the set of possible discrete system states (markings) to avoid redundancies inherent to the Proxel method.
Experiments were conducted showing that the MultiProxel approach lets the simulation be performed between two and five times faster for realistic models, without any loss in accuracy. If no redundancies in the model can be exploited, the method has been demonstrated to incur only a small computational overhead.
The MultiProxel approach thus has the potential of making deterministic state-space analysis of existing models more efficient, and to enable the analysis of bigger models that more accurately reflect real systems.

## INTRODUCTION

Modeling and simulation of discrete stochastic systems (DSMs) is an important tool in Engineering, Logistics and other areas. It is used to analyze possible future behavior of existing systems, evaluate consequences of possible changes to real systems, or even analyze systems that do not yet exist.

The simulation technique of choice is usually the discrete event simulation (DES) which uses random numbers to generate a possible system development, and then performs numerous simulation runs with different random numbers in order to compute an accurate mean system behavior. The reliance on and influence of random numbers make DES often undesirable to use. For models containing rare events, DES is often unfeasibly slow.
An alternative to DES are state space-based simulation methods. Those can directly compute the mean system behavior without having to resort to replications or random numbers. These techniques, however, have their own disadvantages: they are either limited in their expressiveness (e.g. can only simulate Markovian systems) and thus are not applicable to the majority of discrete systems, or are computationally too expensive to be useful in practice.
One promising state space simulation method is the Proxel method in (Horton 2002, Lazarova-Molnar 2005). Its expressiveness is not limited to specific probability distributions and its superior efficiency on some models has been demonstrated in (Lazarova-Molnar and Horton 2004). However, the method is still unfeasibly slow for most models.
The goal of this paper is therefore to increase the efficiency of said method. This is done by developing an extension to that method which avoids a certain type of computational redundancy. This will make state space-based simulation feasible for more models. It will also allow for the analysis of bigger, more realistic Hidden non-Markovian Models (c.f. (Krull and Horton 2009a)), which use the Proxel method as their primary solution method.

## RELATED WORK

Since the influence of randomness on simulation results is sometimes undesirable, state space-based simulation techniques were developed that can compute the desired simulation result directly, without any influence of randomness.

Continuous-Time Markov Chains (CTMCs) are one major state-space modeling and solution approach (Bolch et al. 1998). Their results are not influenced by randomness and result computation is efficient. However, CTMCs limit all state transitions to be memoryless, i.e. to have an underlying exponential probability distribution. This limitation prevents CTMCs from accurately modeling most real systems and thus causes CTMCs to not be applicable to the majority of discrete stochastic systems.

One way to extend the applicability of CTMCs is to convert all general probability distributions of a model to a well-adjusted chain of exponential distributions, a so-called phase-type distribution. This way, many discrete models can be represented as CTMCs and can be solved using CTMC solution algorithms. The downsides of phase-type distributions are that they increase the model's state space, and that the computation of the chain parameters is time-consuming itself (Bobbio et al. 2002, Isensee and Horton 2005).

The Proxel method was developed in order to overcome the limitations CTMCs have. Simply extending CTMCs using the method of supplementary variables from (German 1995) in order to model age-dependent state change probabilities poses algorithmic difficulties and would generate stochastic matrices that are too big to be handled. Therefore, the Proxel method creates only the necessary parts of these matrices on the fly. This allows for the efficient simulation of some discrete stochastic systems, but is often unfeasibly slow.

One big disadvantage of the Proxel method compared to CTMCs is that due to the dynamic generation of relevant parts of the matrix, certain intermediate results need to be computed over and over again. In this paper, we are consequently developing a modification to the Proxel method that removes some of these redundancies. As a first step to this extension, we will briefly review the Proxel method. Since discrete stochastic models can often intuitively be represented as Generalized Stochastic Petri Nets (GSPNs) with arbitrary firing times, the Proxel definition borrows some terms from the definition of GSPNs. So, in the description of the Proxel method, the different discrete states are called "markings" and the state transitions between them are simply called "transitions"

## REVIEW OF THE PROXEL METHOD

The Proxel method facilitates state space-based simulation of DSMs with arbitrary probability distributions (Lazarova-Molnar 2005). The basic approach of the Proxel method is to divide simulation time into time steps of equal size and then to trace possible system developments for each time step in parallel.

The possible system developments are stored in tuples $(m, \tau, \Pi)$ called "Proxels". Here, $m$ represents the marking, i.e. the discrete system state. The entry $\tau$ is the age vector containing the most recent duration of inactivity for all active or otherwise relevant state transitions, i.e. the time during which a given state change could have happened, but did not. Finally, the entry $\Pi$ is the probability of the combination of marking and age vector.

Simulation starts by determining a suitable simulation time step and providing a Proxel for the start state, usually with an age vector $\tau$ containing only zeros, and with a probability of one.

The simulation is then iteratively performed for each time step: For each Proxel of a time step, the probabilities of all possible single[1] state changes during a time step (and the probability of no state change occurring at all) are computed using numerical integration of ordinary differential equations (Buchholz 2008). The outcomes of these state changes are then stored as Proxels for the next time step.

The pseudo-code for a single Proxel simulation step is shown in Algorithm 1. For a complete simulation, this algorithm simply needs to be executed iteratively for each time step till the desired end of the simulation, with the output *currentStep* of one time step being the input *inputProxels* of the next one.

**Algorithm 1**: Simulating a single time step using the standard Proxel algorithm

```
Data: inputProxels, Δt
foreach Proxel p ∈ inputProxels do
    Π_trans = getTransitionProbabilities( p.transitions,
    p.ageVector);
    stayProb = 1 - sum(Π_trans);
    if (stayProb > 0) then
        currentStep.addOrMerge
        (createInactivityProxel( p, Δt));
    foreach Transition trans ∈ p.transitions do
        currentStep.addOrMerge(
        createTransitionProxel(p, trans, Π_trans[trans]));
return currentStep;
```

One issue with a naive implementation of the Proxel algorithm is that each Proxel of a given time step causes multiple Proxels for the next time step to be created (one for each possible state change and one for inactivity), resulting in an exponential growth of the number of Proxels per time step. To curb this exponential growth, Proxel merging is used: all Proxels of a given time step that represent the same age vector and marking are merged into a single Proxel by accumulating their probability. Algorithmically, efficient merging requires an efficient way of finding these duplicate Proxels. In the current implementation, this is done by storing all Proxels of a give time step in a binary search tree instead of a generic container. This slightly increases the

[1] The Proxel method does not compute the probability of multiple state changes occurring during a single time step. This is a known source of inaccuracy, especially for big time steps.

time complexity of adding a single Proxel from $O(1)$ to $O(log(n))$, but reduces the time complexity for finding duplicates from $O(n)$ to $O(log(n))$.

### Reflections on the Performance of the Proxel Method

The Proxel method can efficiently simulate some models, but for most bigger models, the approach is too slow to be feasible.
To determine the major performance bottlenecks, our Proxel simulation software was profiled using the tool Valgrind (Nethercote and Seward 2007). Profiling showed that about 80% of the computation time of a Proxel simulation run is spent on computing the state transition probabilities and on inserting Proxels into the search tree. Thus, a reduction of the number of Proxels and the number of state transitions probabilities to be computed could significantly speed up Proxel computations.

## THE MULTIPROXEL APPROACH

### Observations Leading to this Project

The extensions to the Proxel method developed here to speed up Proxel computations are based on two observations on the behavior of a Proxel simulation:
First, it was found that while the state transition probabilities to be computed depend on all active state transitions (which are determined by the marking) and potentially all elements of the age vector, they do not actually depend on the marking itself. In the example of customers waiting in line to be served at a bank counter in Figure 1, different numbers of waiting customers need to be represented by different markings, but these numbers do not influence the probability of the current service ending or a new customer arriving in the next time step.
Thus, one may cluster these markings of a simulation model that agree in the types and parameters of all of their active state transitions. This allows for the transition probabilities to be computed only once for each cluster of markings (instead of for each marking individually, as is done so far). These transition probabilities are then applied to all markings of a cluster, preventing redundant probability computations.
Second, when dealing with these clusters, the target markings for a given state transition for markings of one cluster tend to fall into a single other cluster. Put another way, if the target marking of a marking $m_1$ through state transition $t$ is $m_1'$, then the target markings of other markings in the same cluster as marking $m_1$ through state transition $t$ are likely to fall into the same cluster as $m_1'$. In the bank example in Figure 1, all markings but the one for "0 Customers Waiting" form a cluster, and the state transitions for almost all of them

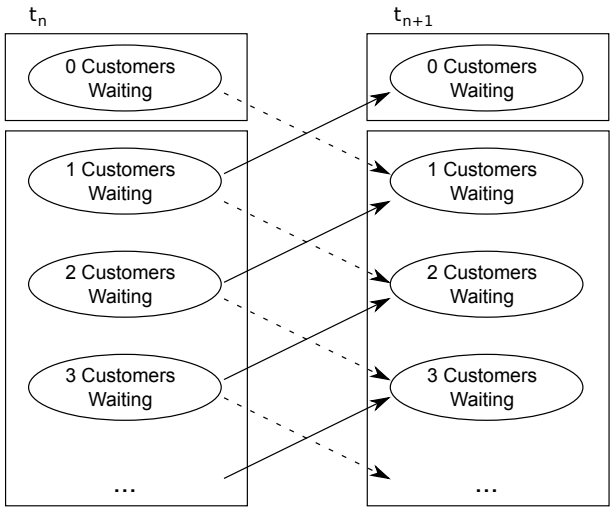


Figure 1: Schematic representation of the state space for the "Bank" example. Ovals represent markings, rectangles group them into clusters. Dashed lines represent the state transitions for "Customer Arriving", solid lines the ones for "Customer Served".

lead to markings inside a single cluster. This behavior can be exploited by modifying the Proxel data structure to contain information on all markings of a cluster instead of just a single marking. Then, many computations can be performed efficiently on clusters of Proxels. Both ideas together, the clustering of markings to prevent redundant computations, and the joint treatment of markings belonging to the same cluster, are the basis of our new MultiProxel approach.

### Defining the MultiProxel Approach

**Identifying Marking Clusters** One necessary preprocessing step for MultiProxel simulations is to find sets of markings with identical state transition behavior, i.e. markings that agree in the types and parameters of the probability distributions of all of their transitions. Our approach is to serially number all state transitions and then to assign a cluster ID number to each marking based on the state transitions active for that marking. Specifically, the cluster ID for a marking is the binary string $B = b_0b_1b_2...b_n$ where each $b_i$ is 1 if state transition $i$ is active for that marking, and is 0 otherwise. All markings with identical ID then form a cluster.

**MultiProxel Definition** Next, the MultiProxel data structure needs to be defined. While a Proxel was defined as the tuple $(m, \tau, \Pi)$ containing the marking, age vector and the probability of the combination, a MultiProxel should hold data on all markings of a cluster. Therefore, we define a MultiProxel as the tuple $(cluster, \tau, S)$ were *cluster* identifies the cluster that all markings of a MultiProxel belong to, and $S$ is a set of

$(m, \Pi)$ pairs, storing the probability $\Pi$ for each marking $m$ of *cluster*.
At any given point in simulation time, the set of all MultiProxels contains exactly the same information as would the set of all Proxels of a normal Proxel simulation. Each representation can easily be converted to the other one. Consequently, the simulation results will also be identical.

### The Algorithm

The usual Proxel simulation algorithm from (Lazarova-Molnar 2005) as shown in Algorithm 1 needs to be modified only slightly to make use of MultiProxels: transition probabilities are computed only once for each MultiProxel, and then are used for each marking that is part of the MultiProxel. And since the target states that can be reached through a single transition from all markings of a MultiProxel are likely to be concentrated in very few MultiProxels, only these few MultiProxels need to be located in the search tree.
The pseudocode for a single MultiProxel simulation step is given in Algorithm 2. As for the Proxel pseudocode in Algorithm 1, for a complete simulation this algorithm simply needs to be executed iteratively for each time step, with the output *currentStep* of one time step being the input $inputMultiProxels$ of the next one.

**Algorithm 2**: Simulating a single time step using the MultiProxel algorithm

```
Data: inputMultiProxels, Δt
foreach MultiProxel mp ∈ inputMultiProxels do
    Π_trans = getTransitionProbabilities(
    mp.cluster.transitions, mp.ageVector);
    stayProb = 1 - sum(Π_trans);
    if (stayProb > 0) then
        currentStep.addOrMerge
        (createInactivityProxel( mp, Δt));
    foreach Transition trans ∈ mp.cluster.transitions
    do
        mp_prev = NULL;
        foreach Marking m ∈ mp.cluster.markings do
            m_target = m.getTarget(trans);
            if (mp_prev.cluster = m_target.cluster) then
                mp_prev.addProbability(m_target,
                mp.probability(m) * Π_trans[trans]);
            else
                mp_prev = currentStep.addOrMerge(
                createTransitionProxel(mp, m_target,
                trans, Π_trans[trans]))
return currentStep;
```

Here, $\text{mp}_{prev}$ is used to cache the previous target MultiProxel to prevent multiple successive lookups of the same target MultiProxel and thus allowing to often find target MultiProxels in constant time.
In an efficient implementation, all markings of a cluster would be numbered serially so that the set of all $(m, \Pi)$ pairs of a MultiProxel can be implemented efficiently as a simple array that stores the probability of marking $i$ at index $i$, and can be accessed in $O(1)$.

**Impact** Conducting state space-based simulation using MultiProxels instead of Proxels allows for much more efficient simulations: with MultiProxels, each state transition probability needs only be computed once for each MultiProxel, no matter how many markings are part of the associated cluster. Thus, the overall number of transition probabilities to be computed is reduced substantially.
Also, as was stated before, with respect to a single state transition, the target markings of most markings of a cluster fall into a single other cluster. Consequently, for a given state transition, the target for most markings of a MultiProxel are also only a few or even a single MultiProxel. Thus, only those different MultiProxels needs to be found explicitly in the search tree with a time complexity of $O(log(m))$ in the number of MultiProxels. The probability for each marking can then be added to the MultiProxels in $O(1)$.
Finally, the MultiProxel approach will cause two different types of small computational overhead compared to the normal Proxel approach: First, a constant additional amount of time needs to be spend to create the clusters. Second, overhead by a constant *factor* is caused by MultiProxels, because they are more costly to compare to each other (since they contain more elements) and have to maintain $(m, \Pi)$ pairs with zero probability. The Proxel approach would simply not have created Proxels with a probability of zero, but MultiProxels need to be created as soon as there is some probability left in at least one of the markings it represents. This overhead, however, is not substantial compared to the potential benefits, as shown in the next section.

## EXPERIMENTS

Experiments were conducted to assess benefits and applicability of the MultiProxel algorithm. For that purpose, the following questions were investigated:

- How high is the overhead of the MultiProxel method?
- How high is the performance gain for typical models?
- How high is the performance gain for models particularly suited for the approach?

For each question, a suitable simulation model was chosen, and was simulated with both the Proxel and MultiProxel algorithms. For all experiments conducted,

the actual computed simulation results for the Proxel and MultiProxel simulations were identical, giving additional confidence that the MultiProxel approach indeed solves the problem at hand correctly.

**Experiment Setup**

Multiple models were simulated to asses the efficiency of the MultiProxel approach. Each model was simulated multiple times, varying the simulation step size and thereby the result accuracy. The particular step sizes for the experiments were chosen in order to yield computation times that are high enough to eliminate the random influence of concurrently running programs on the computation time.
For the same reason, all computation times measures are not the overall simulation duration (wall clock time), but the CPU usage times of the respective simulation processes. All experiments were conducted on a Core2Duo 3 GHz CPU. However, both simulation programs are single-threaded, using only one of the CPU cores. The CPU was manually set to always run at full speed in order to eliminate the influence of dynamic CPU throttling on the computation times.
The implementations of the Proxel and MultiProxel algorithms share most of their source code. They use identical code for Proxel storage, numerical integration, and probability distributions. Different code is used only when the differences in the algorithms make it necessary. This way, the differences in computation time can indeed be attributed to the different theoretical approaches with great confidence.

**Computational Overhead**

The question of computational overhead is important in order to determine which fraction of markings needs to be clusterable for the model to be simulated more efficiently using the MultiProxel approach. The overhead itself is best determined using simulation models where no markings can be clustered and thus the MultiProxel approach does not provide any benefits, but still imposes the overhead. This way, the difference in computation time between the Proxel and MultiProxel approaches for such a model yields the computational overhead.
One example of such a model is the three marking fully-connected model (cf. Figure 2). As the state transition probability distribution are all different, the model cannot be clustered. Therefore, of the three markings that are reachable in this model, each is the only marking in its respective cluster in a MultiProxel simulation.
The model was simulated with each algorithm and two different simulation step sizes. Each combination was simulated three times and the computation times of the three runs were then averaged to yield the results shown in Table 1.
For both simulation step sizes, the difference in computation time amounts to about 13%. Since clustering is not applicable to this model, these 13% of loss in efficiency represent the computational overhead of the MultiProxel method. And since the relative overhead of about 13% is the same factor for both step sizes, it can be assumed that no overhead by a constant *value* (as opposed to overhead by a constant *factor*) of the MultiProxel approach was observed. Thus, the constant overhead of clustering the markings on the computation time is negligible compared to the overall simulation time.

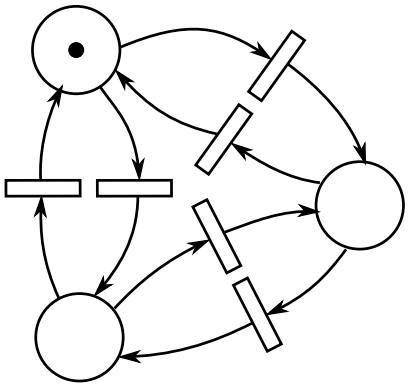

Figure 2: Stochastic Petri Net of the "Fully Connected" model.

| Step Size | Computation Time | | Overhead(+) |
|---|---|---|---|
| | Proxel | MultiProxel | Gain(-) |
| 2 | 3.7s | 4.2s | +13.9% |
| 1 | 29.4s | 33.1s | +12.6% |

Table 1: Computation Times for the "Fully Connected" Model.

**Memory Consumption Overhead**

The differences in memory consumption between the two algorithms are highly implementation-dependent and therefore are not analyzed here in detail. In a naive implementation, a simulation program would only allocate as much memory as is required to store all (Multi)Proxels, and a MultiProxel would only be as big as necessary to store its probability vector. Under these circumstances, a MultiProxel simulation with its more compact representation will always require equal or less memory than a Proxel simulation.
Our implementation, however, is optimized to reduce memory allocation time overhead and not size. To that end, it always allocates $2^n$ (Multi-)Proxels at a time, and all MultiProxels of a simulation have identical size, with a probability vector size corresponding to the maximum number of markings in a cluster. Thus, memory consumption overhead or reduction of the MultiProxel approach depends strongly on the number of clusters and the distribution of markings between them. Generally, in our implementation, memory consumption of the

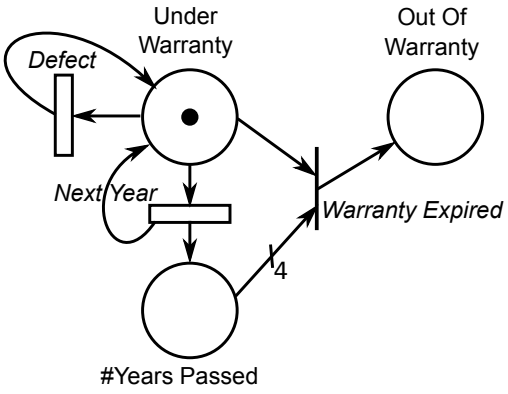


Figure 3: Stochastic Petri Net of the "Warranty" model.

| Step Size | Computation Time | | Overhead(+) |
|---|---|---|---|
| | Proxel | MultiProxel | Gain(-) |
| 5 | 12.5s | 5.5s | −55.9% |
| 2 | 79.7s | 37.3s | −53.1% |

Table 2: Computation Times for the "Warranty" Model.

MultiProxel method tends to be of the same magnitude as for the Proxel method.

### Performance on Typical Models

To analyze performance of the approach on typical models, we chose the "Warranty" model previously analyzed using the Proxel method in (Lazarova-Molnar and Horton 2004). It is a real-world simulation model measuring the costs incurred due to defects of a machine part under warranty. The model is actually used in industry and was not developed with MultiProxels in mind, making it a good candidate to assess MultiProxel performance under general conditions. Figure 3 shows a Petri Net of the model.

The model's state space consists of five markings (first, second, third, fourth year of warranty, and "out of warranty"), which can be folded into two clusters. Thus, a theoretical computation time reduction of up to $^3/_5$ can be assumed.

The actual computation times for different simulation step sizes are shown in Table 2. The simulation using the MultiProxel took about 55% less time in both cases, almost reaching the theoretical maximum of 60% for this model. Again, computation times between Proxel and MultiProxel differ by a constant factor, indicating that this is due to the constant-factor overhead of comparing and managing MultiProxel. Therefore, the additive step size-independent clustering overhead appears to be negligible compared to the overall simulation time even for models where clustering does succeed.

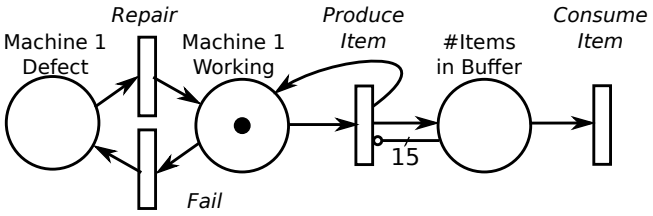


Figure 4: Stochastic Petri Net of the "Buffer" model.

### Performance on Suitable Models

Finally, the approach's performance on particularly suited models was assessed. Here, we chose a typical buffer dimensioning problem (cf. Figure 4): one machine in a factory produces items that another machine consumes, with both processes being influenced by randomness. The first machine may fail occasionally, or may sometimes produce items too slowly, so that the second machine would have to wait for input at certain times. In order to compensate this variability, a buffer is installed between both machines, allowing temporary storage of a certain number of items produced by the first machine. This model can be used to analyze the influence of different buffer sizes on the throughput of the whole system.

The model is particularly suited for MultiProxel simulations, because increasing the buffer size will only increase the number of markings, but not the number of clusters. Specifically, for buffers with a capacity of $n \geq 2$ items, the model consists of $2n + 2$ markings that can always be folded into only five clusters. Thus, the performance gain should increase when increasing the buffer size.

In our experiments, the "Buffer" model was simulated using only a single simulation step size, since previous experiments established that the step size does not impact the relative gain in efficiency. The size of the buffer, however, was varied between experiments to determine the influence of the state space size on the efficiency gain.

The results are shown in Figure 5. For very small buffer sizes, computation times of both approaches are almost identical. With increasing buffer size, however, the computation time for the Proxel algorithm increased much faster than that for the MultiProxel one. For a buffer size of 19 items, the computation time of the MultiProxel approach is about 80% less than that of the normal Proxel approach. For higher buffer sizes, the efficiency actually decreases slightly.

For this model, an 80% reduction in computation time seems to be the upper limit. The most likely reason for this limit is that when the buffer size is increased beyond a certain point, most markings of many MultiProxels become impossible to reach. Thus, many MultiProxels will be sparsely populated and represent the same information as would only a few Proxels, reduc-

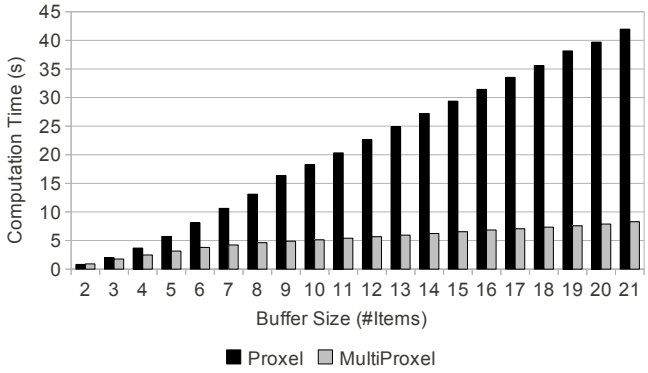


Figure 5: Computation Times for Different Buffer Sizes in the "Buffer" Model. Simulation Step Size is the Same in all Cases.

ing the MultiProxel performance gain over the Proxel approach.

## CONCLUSION AND OUTLOOK

In this work, we developed and implemented a variation of the Proxel method for the state space-based simulation of discrete stochastic models. The new MultiProxel approach promises to be more efficient than the standard Proxel approach, since it avoids certain redundant computations of the Proxel method.

The experiments have shown a relatively small computational overhead of the MultiProxel method compared to the Proxel method, if the model contains no redundancies that the MultiProxel approach may exploit. Further experiments have shown a reduction in computation time of about 50% for typical models, and of up to 80% for - still realistic - models with an advantageous state space structure. Memory consumption was explained to be highly implementation dependent, but to be similar for both approaches in comparable implementations.

The new approach is based on a fully automated clustering algorithm, requiring no tuning or additional user input beyond that of Proxel simulations. Consequently, simulation software can transparently replace their core Proxel algorithm by a MultiProxel one. Since the new approach imposes only a slight overhead and only in some cases, and demonstrated a remarkable performance increase otherwise, we recommend the new MultiProxel algorithm to be used as the default algorithm for a general-purpose Proxel simulator.

### Model Classes Benefiting from MultiProxel Approach

The MultiProxel approach is applicable to all discrete stochastic models, and for most models, the gains in efficiency will outweigh the slight computational overhead. Furthermore, there are some classes of systems that will particularly benefit from MultiProxel due to the structure of their state space:

**Generalized Stochastic Petri Nets** Most DSMs that are represented as a generalized stochastic Petri Net with arbitrary firing distributions have a particular model structure. Most transitions in GSPNs exhibit only three different types of behavior:

- They are active
- They are inactive, because there are not enough tokens in a place to satisfy firing conditions
- They are inactive since an inhibitor arc blocked them due to too many tokens being in a place

Thus, independent of how many tokens may populate a place, such a part of a model can be represented by three state clusters, while a standard Proxel simulation requires as many markings as there can by tokens in a place.

**Queuing Systems** Queuing systems (Law and Kelton 2000) are discrete stochastic systems that model queues (waiting lines) or networks thereof. Here, a queue consists of a buffer to store items, and one or multiple servers that process these items individually and afterwards either remove them from the system or forward them to another queue. Queuing systems have manifold applications, among others in the simulation of computer networks and logistics.

Similar to stochastic Petri Nets, the types of possible state transitions for queuing systems follow a simple structure. A queue may be:

- empty and thus cannot produce output.
- partially used and thus may receive input and produce output.
- full and thus cannot receive input.

A standard Proxel simulation would need to represent each possible number of items in a queue as an individual marking. The MultiProxel approach on the other hand requires only three clusters for the three cases mentioned above, independent of the queue's capacity.

**Hidden non-Markovian Models** Hidden non-Markovian Models (HnMM) are a new paradigm to model and analyze systems whose runtime behavior cannot be observed directly, but for which some system behavior produces observable output (Krull and Horton 2009a). Unlike older approaches, HnMMs allow for the hidden model to be any discrete stochastic system with arbitrary firing times. HnMMs enable the analysis of certain characteristics of the system behavior that produced a given sequence of observed output.

Currently, the majority of HnMMs is analyzed using a slightly modified Proxel method (Krull and Horton 2009b). Here, in many cases, the result of interest is some aggregate of the possible system behavior over a given simulation time, subject to the observed output. For example, in (Buchholz et al. 2010), the simulation tried to find the probable number of defective items produced by a certain machine. The current number of defects at a given situation (Proxel) and simulation time was encoded into the discrete state of each Proxel. This, however, means that the number of discrete systems states is increased by a factor corresponding to the quantity measured (in the example the maximum number of defective items produced by one machine). This quantity, however is only stored for statistical analysis of the simulation results. It has no influence on further system behavior. Consequently, using the MultiProxel approach to solve HnMMs, all system states that encode the same marking with different histories could be represented by a single state cluster. Thus, with the MultiProxel method, the computational overhead of HnMMs on the Proxel method can almost be eliminated.

**Benefits**

The increased efficiency of computations due to the MultiProxel approach result in shorter computation times to obtain the same accuracy as before, so that results are now available and may be used sooner. By reducing the simulation step size, results may also be obtained at higher accuracy in same time and thus allow for more accurate predictions within the same time frame. Finally, using the gain in efficiency, bigger, more realistic models may be simulated using state space-based techniques for the first time.

**Future Work**

The MultiProxel approach captures and prevents redundancies in situations where the transition probabilities for different markings for a given age vector are identical. It does not yet prevent redundancies for cases where age vectors between Proxels may differ, but transition probabilities are nevertheless identical, either by coincidence, or because the age vector contains ages of state transitions that are not currently active. Further research may also show ways to reduce redundancies in computations performed for different time steps, or even between slightly different models. It might also be advantageous to identify simulation intermediate results that are not necessarily identical, but very similar, and to exploit these similarities.

**References**


Bobbio A.; Horváth A.; and Telek M., 2002. *Phfit: A general phase-type fitting tool.* In *Proceeding of 12th Performance TOOLS*. 82–91.

Bolch G.; Greiner S.; de Meer H.; and Trivedi K.S., 1998. *Queuing Networks and Markov Chains.* John Wiley & Sons, New York.

Buchholz R., 2008. *Improving the Efficiency of the Proxel Method by using Variable Time Steps.* Master's thesis, Otto-von-Guericke Universität Magdeburg.

Buchholz R.; Krull C.; Horton G.; and Strigl T., 2010. *Using Hidden non-Markovian Models to Reconstruct System Behavior in Partially-Observable Systems.* In *3rd International ICST Conference on Simulation Tools and Techniques.*

German R., 1995. *Transient Analysis of Deterministic and Stochastic Petri Nets by the Method of Supplementary Variables.* In *Proceedings of the 3rd International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 95).* IEEE Computer Society, 394–398.

Horton G., 2002. *A New Paradigm for the Numerical Simulation of Stochastic Petri Nets with General Firing Times.* In *European Simulation Symposium.* SCS European Publishing House, Dresden, Germany.

Isensee C. and Horton G., 2005. *Approximation of Discrete Phase-Type Distributions.* In *Annual Simulation Symposium 2005.* San Diego.

Krull C. and Horton G., 2009a. *Hidden non-Markovian Models: Formalization and Solution Approaches.* In *Proceedings of 6th Vienna International Conference on Mathematical Modelling.* Vienna, Austria.

Krull C. and Horton G., 2009b. *Solving Hidden non-Markovian Models: How to Compute Conditional State Change Probabilities.* In *21st European Modeling and Simulation Symposium.* Santa Cruz de Teneriffe, Spain.

Law A.M. and Kelton W.D., 2000. *Simulation, Modelling and Analysis*, McGraw-Hill, chap. 1. 3rd ed., pp. 94–98.

Lazarova-Molnar S., 2005. *The Proxel-Based Method: Formalisation, Analysis and Applications.* Ph.D. thesis, Otto-von-Guericke Universität Magdeburg.

Lazarova-Molnar S. and Horton G., 2004. *Proxel-Based Simulation of a Warranty Model.* In *European Simulation multiconference.* SCS European Publishing House 2004.

Nethercote N. and Seward J., 2007. *Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation.* In *Proceedings of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI 2007).* San Diego, California, USA.