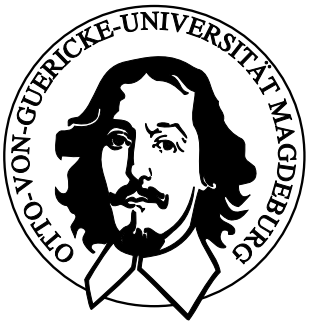
Otto-von-Guericke-University Magdeburg



Department of Computer Science
Lehrstuhl für Simulation

Master's Thesis

Submitted as Diplomarbeit in the Diplomstudiengang Informatik

---

Improving the Efficiency of the Proxel Method by using Variable Time Steps

---

Supervisors: Prof. Dr.-Ing. Graham Horton
Dr. Claudia Krull

Second Referee: Prof. Dr.-Ing. Jana Dittmann

Author: Robert Buchholz
Student of Computer Science

Magdeburg, September 25, 2008

Abstract

The purpose of this thesis was to develop and implement an algorithm for the simulation of stochastic models with variable time step sizes based on the proxel method.

The proxel method is a relatively new state-space based approach for simulating stochastic models. It strives to be more efficient than earlier approaches that are based on discrete-event simulation.

The previous implementations of the proxel method used constant time steps to trace all possible events in a system at equidistant points in time. This approach works efficiently on a small class of simulation models, but still has severe limitations on the size of the model that can be simulated in an acceptable time span.

In this thesis, the proxel method was extended with support for variable time step sizes in the hope that this added flexibility would translate to a more efficient way of simulating stochastic models. In order to enable variable time steps, a simulation algorithm was devised and criteria were developed that can decide on whether a certain time step size negatively impacts the simulation accuracy and therefore needs to be subdivided.

The algorithm and all criteria were then implemented, tested on multiple models, and compared to each other and to an implementation of the standard proxel algorithm. These experiments showed that some criteria for variable time steps can indeed increase the accuracy obtained in a given amount of time remarkably, but also that there are still some difficulties in reliably determining the accuracy of the results obtained.

During the development of the algorithm and criteria for variable time steps, additional ways of substantially increasing the efficiency of the proxel method have been developed and are presented in this thesis.

Contents

Chapter 1

Introduction

This introduction serves as an overview of the remaining thesis. Section 1.1 lays down the reasons for dealing with stochastic systems at all. Section 1.2 then explains why improvements to the current methods for solving stochastic systems are necessary and why the introduction of variable time steps to the proxel method might be particularly useful. The goals set for this thesis are then introduced in Section 1.3, while Section 1.4 lists the tasks thought necessary to achieve these goals.

1.1 Background

This thesis is concerned with increasing the efficiency of the simulation of stochastic models.

Simulation of Stochastic Models Stochastic models are models influenced by randomness. Most systems in our natural or man-made environment have at least some stochastic aspects and are therefore modelled or approximated by stochastic systems. The reason that the simulation with stochastic models has become a valuable tool is that since the model behaves very similar to the real system, the influence of changes to the real system can be tested on the model before they are actually implemented or even before the real system exists.

Simulation with stochastic models is especially useful and in widespread use in manufacturing and engineering. In these areas, observing changes in the real system would either take too long (e.g. simulation of a warranty period), be too dangerous (e.g. impact of reduced safety measures) or too costly (e.g. setting up a new production line).

Since the simulation results of stochastic models have become quite important, it became paramount to be able to accurately and quickly simulate stochastic systems. This, however, is still a challenge.

Monte Carlo/ Discrete Event Simulation If a model was completely deterministic, there would only be one possible outcome which could be determined with reasonable

effort. Since most models are stochastic, however, their output is a random variable and numerous different outcomes are possible, in most cases an infinite number of them. Thus, one is usually interested in a statistic measure of these output variables, most commonly the expected value.

A simple approach to simulate stochastic models is the Monte Carlo simulation: A single possible outcome of the system can easily be simulated by selecting a possible deterministic value for each stochastic element of the model. The value is chosen at random out of all possible values for that element and the probability distribution of the element determines the probability with which a certain element is chosen. Since the output is only one possible outcome, numerous replications (simulation runs) with differently chosen values for the stochastic elements must be conducted to calculate statistic measures (average value, standard deviation, confidence intervals). The exact number of simulation runs necessary depends on the desired accuracy, as well as on structural properties of the model in question.

More precisely, to retrieve meaningful statistic results, all possible outcomes of a model must occur in the result set. For example, in a model where one event has a probability of only 10^{-6}, a million replications are necessary on average for that rare event to occur even once. Many more million replications are necessary to determine the probability of that rare event with a reasonable accuracy. Thus, accurately determining simulation results through discrete event simulation is not feasible for many models.

Proxel Simulation The proxel method was devised by Graham Horton [3] in 2002. It is a promising new approach that can significantly reduce the computation time necessary for the simulation of some models. Instead of having numerous replications of a discrete-event system to be able to observe all possible outcomes, the proxel method deterministically tries to trace all possible paths through the stochastic model in parallel. Solving a model exactly would require setting up and solving systems of partial differential equations, but the proxel method approximates the solution by discretizing the time to steps of constant size and simulates all possible outcomes for each time step[1]. This does not require handling differential equations at all, but only simple function evaluations.

The proxel method has the advantage that simulation results are deterministic (albeit only an approximation) and hence no replications are necessary. Furthermore, the simulation results obtained from simulating with two different step sizes can be used to extrapolate a result than can be more accurate than both separate results. However, the advantage of determinism comes at a price. For many models, the

[1] Actually, the proxel method only *tries* to simulate all possible outcomes, but will ignore those cases where two or more events could have occurred one after another during a single time step. This is one of the error sources inherent to the proxel method.

number of different states that need to be tracked increases exponentially with the simulation time. This, in turn, increases the computational complexity by at least the same factor and often makes it impossible to compute simulation results with acceptable accuracy within given memory and time constraints.

1.2 Motivation

As explained in the previous section, even the proxel approach is too slow to adequately simulate some models. It might, however, be possible to significantly improve its efficiency: The current implementations of the proxel method use a single step size to simulate a complete model. This simplyfies implementation and understanding of the algorithm, but is not an optimal behavior in terms of simulation efficiency as the following simple example demonstrates:

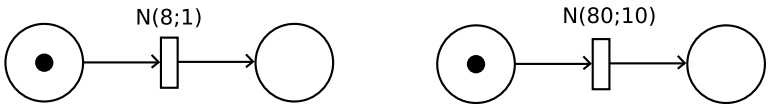


Figure 1.1: Two simple Petri nets with very similar behavior

The two Petri nets[2] in Figure 1.1 are just scaled versions of each other. The flow of probability in the second model is slower by a factor of ten, but otherwise the models are identical. Therefore, if the first model is simulated using the proxel method with a step size of Δt and the second model with a step size of 10 Δt, they both should produce the same relative error.

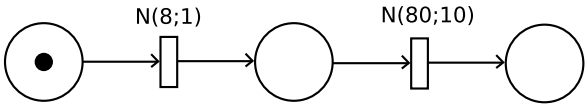


Figure 1.2: Concatenation of the two separate Petri nets from Figure 1.1

When the two models are concatenated as done in Figure 1.2, a problem surfaces: With a constant step size, the second transition will be simulated with about ten times the accuracy of the first one. But since the accuracy of the overall simulation result depends on the least accurately simulated part of the model, this high precision for the second transition will barely reduce the overall error. Indeed, it would make sense to simulate the second transition with a bigger step size than the first one so that both are simulated with about the same error. This would reduce computation time without significantly impacting the result.

One promising way to speed up the proxel method is too use variable time steps instead of a constant one: The accuracy of a proxel simulation result usually depends

on the chosen time step size. But the actual time step size necessary to achieve a certain level of accuracy depends on the number and frequency of events. Thus, it often varies during the simulation of a single model.

A good algorithm could determine the step size necessary for the desired level of accuracy in each situation separately and would simulate the step with exactly that step size. Since the number of possible outcomes grows (often exponentially) with the number of steps already taken, variable time steps can ensure than only as few steps as necessary are simulated consecutively. Thus the number of overall steps is reduced to a minimum necessary for the desired level of accuracy and the growth of the number of states is greatly reduced.

Extending the proxel method with variable time steps was first attempted by Wickborn and Horton [12, 13]. The results reported in these papers look promising, but the papers lack a detailed description of the algorithm used and do not report numbers that would allow a comparison of the algorithms with constant (CTS) and variable time step (VTS) sizes.

1.3 Goals

The purpose of this thesis is to document the development, implementation and testing of an algorithm that allows simulation based on the proxel method with variable time steps. All these steps are taken with the goal of determining and explaining the following issues:

1. Under which conditions are variable time steps superior to constant ones?
2. What are the quantitative benefits of using variable time steps?
3. What is the best way to determine the time step size under which conditions?
4. Is Richardson's extrapolation [10] applicable to the results of variable time steps?

The first question stems directly from the motivation of this thesis. Hopefully, variable time steps will show to be superior in every situation. But if that is not the case, it is important to know for which classes of models the VTS algorithm is likely to be faster and for which ones the CTS algorithm should be used.

The answer to the second question should be a sound statement about by which factor the simulation with variable time steps can be faster than the current approach with constant time steps. This is important, because variable time steps will likely increase the algorithm complexity and thus make it more time-consuming and error-prone to implement. These drawbacks can only be justified if the speed increase is substantial.

Third, since the VTS algorithm allows to use different simulation step sizes in different situations during a single simulation run, criteria must be developed that determine the exact step size to be used in a given situation. These criteria are likely to perform differently in terms of simulation efficiency, i.e. accuracy achievable with given time constraints. The quality of these criteria must be assessed just as for the VTS algorithm itself.

Finally, Richardson's extrapolation method is widely used for the CTS proxel simulation. It combines the results of two simulation runs with different step sizes in order to obtain a result that is more accurate than the two initial results. This increases the accuracy of the algorithm without requiring a higher computational effort, and allows to estimate the level of accuracy of the results. Thus, it is an important question whether Richardsons method of extrapolation is applicable to the results obtained with variable time steps as well. If the accuracy of the simulation results cannot be assessed without comparing its results to pre-computed highly accurate (and therefore computationally expensive to obtain) results, the algorithm cannot be used in productively used simulation applications.

1.4 Tasks

In order to attain the goals listed above, the following tasks are deemed necessary:

- Development of a low-overhead VTS algorithm
- Development of criteria that can determine time step sizes
- Error-Free and non-discriminatory implementation of the CTS and VTS algorithms
- Experimentally testing the criteria on different simulation models

The fundamentals of the simulation of stochastic models need to be evaluated (cf. Chapter 2) before any of these goals can be approached.

Afterwards, the development of a VTS algorithm is the first step (cf. Chapter 3) to be taken. While the basic idea of the VTS approach may be clear, the implementation is not. Allowing arbitrary step sizes at arbitrary times can easily increase the simulation complexity to a point where a VTS algorithm is far slower than a CTS one. So the VTS algorithm must combine flexibility in choosing the time step sizes with a low computational and memory overhead compared to the CTS algorithm.

After an algorithm is found that can handle variable time step sizes, criteria that can decide which time step size to use in a given situation need to be developed.

When solutions to these two theoretical tasks are found, the VTS algorithm and the criteria, as well as a CTS algorithm need to be implemented (cf. Chapter 4)

so that the new approaches can be tested and compared to the traditional one. The algorithms should be be implemented error-free and non-discriminatory. Error-freeness is obviously an important prerequisite for the experimental results to have any significance. While it can never be guaranteed, errorfreeness can be tested by comparing the simulation results of the CTS and VTS algorithms to each other, to analytical solutions (if those can be obtained) or to previous results retrieved for the same models by different implementations. Implementing the algorithms in a non-discriminatory way means to put the same effort into optimizing both algorithms, so that differences in the simulation running times can actually be attributed to the differences between the algorithms and not simply to the varying quality of the implementations.

As soon as both algorithms have been implemented, the VTS algorithm along with the implemented step size criteria needs to be tested on different simulation models and the results should be compared to each other and to those of the CTS algorithm (cf. Chapter 5). This is necessary in order to assess the quality of the VTS approach in general and of the individual criteria in particular.

After all tasks stated have been completed, the experimental results need to be evaluated with respect to the questions stated as goals (cf. Section 1.3) to form a sound opinion on the viability of applying variable time steps to the proxel method (cf. Chapter 6).

Chapter 2

Fundamentals

This chapter describes the state-of-the-art of the simulation of stochastic models. Section 2.1 introduces the basic underlying differential equations with which stochastic systems can be described. Since these are not usually solvable analytically, Section 2.2 gives an overview on how to numerically approximate them. The proxel approach explained in Section 2.3 uses these approximations to efficiently simulate stochastic models.

Since extending the proxel method to support variable time steps requires an understanding of the error made per simulation step, the known error sources for the proxel approach are briefly explained in Section 2.4. Finally, Richardson's method (Section 2.5) is introduced as a method that allows for the derivation of more accurate results without reducing the step size and impulse rewards (Section 2.6) are shown as a way to compute more meaningful simulation results.

2.1 Transitional Behavior of Stochastic Systems

The transitional behavior of a single state transition is described by its *probability density function* (pdf)[11]. The probability that the transition fires in a given time interval $[t_1, t_2]$ can be calculated with the formula

$$\Pi_{leave} = \int_{t_1}^{t_2} pdf(t)dt$$

The probability of the transition to have fired before time t is given by its *cumulative distribution function* (cdf). The relationship between the pdf and the cdf of a given distribution is the following:

$$cdf(t) = \int_{0}^{t_1} pdf(t)dt$$

This formula allows the exact computation of the transitional behavior of a single transition as long as it is the only active transition. Stochastic models, however, almost always contain more than a single transition and usually more than one is

active at a given time. For this majority of cases, the active transitions "compete" for the probability to leave through them and the actual leaving probability depends on all active transitions. Hence, the *cdf*s can no longer be used to compute the leaving probability through a single transition for a certain time step. The only property of the system that can be computed for all transitions independently is the rate at which probability leaves the current state through each transition under the condition that the transition has not yet fired. This rate function is called the *hazard rate function*[1] (hrf) and is computed as [3]:

$$\mu(t) = \frac{pdf(t)}{1 - cdf(t)}$$

Using the hazard rate, the dynamic behavior of a single transition is defined by

$$\frac{d\Pi}{dt} = \Pi'(t) = -\Pi(t)\ \mu(t) \tag{2.1}$$

which means that the rate of change of the probability to stay in a state is proportional to the hazard rate function value and the remaining probability mass at that time. This equation is an *ordinary differential equation* (ODE), and hence complicated to evaluate. But its strength is the fact that it does not make any assumptions about how probability has left the current state in the past. As long as the probability $\Pi(t_0)$ for any t_0 is known, the system is well defined and $\Pi(t)$ can be computed for any $t > t_0$. Knowing the hazard rate functions of all transitions of a stochastic model and iteratively applying Equation 2.1 to all active transitions theoretically allows for the simulation of the complete model.

2.2 Numerical Solution of Ordinary Differential Equations

The core computation of the proxel method is solving the ordinary differential equation:

$$\Pi'(t) = \frac{d\Pi}{dt} = -\Pi(t)\ \mu(t)$$

This is a first order ODE, i.e. one of the form $y'(t) = f(t, y(t))$. For the computation of the transition probabilities, a $y(t)$ is given and $y(t + \Delta t)$ needs to be computed. Since analytical computation of $y(t + \Delta t)$ is not always possible, numerical approximation must be used.

The current implementations of the proxel algorithm use Euler's method or trapezoid integration to approximate the probability. Euler's method simply takes the

[1] The hazard rate function is sometimes also called the *instantaneous rate function*, *failure rate function* or *mortality function*. But in this thesis, the term *hazard rate function* is used exclusively.

slope at time t as an approximation for the slope of the whole interval. Hence, Euler's solution is

$$\begin{aligned} k &= -\Pi(t) \times \mu(t) \\ \Pi(t + \Delta t) &= \Pi(t) + k\ \Delta t \end{aligned} \tag{2.2}$$

Trapezoid integration is not actually a method to solve ODEs, but one to compute the area under a graph. It is used here, because its results are usually sufficiently close to the solution of the ODE and because it circumvents a numerical problem when using Euler's method. Namely, that $\mu(0) = 0$ for most probability distributions and hence the approximation of the first ODE step is always $\Pi(\Delta t) = \Pi(0)$. The formula for the trapezoid approximation is

$$\begin{aligned} k &= -\Pi(t) \times \frac{1}{2}(\mu(t) + \mu(t + \Delta t)) \\ \Pi(t + \Delta t) &= \Pi(t) + k\ \Delta t \end{aligned} \tag{2.3}$$

It computes slope estimates for t and $t + \Delta t$, but assumes that $\Pi(t + \Delta t) = \Pi(t)$ to estimate the slope at $t+\Delta t$. As this assumption does not usually hold for probability distributions, the error introduced can be quite severe.

The Class of Runge-Kutta Methods Numerous methods for solving ODEs more accurately exist. The most popular ones are the class of Runge-Kutta methods [1]. All of those methods compute numerical approximations of various $y(t^*)$ for $t \leq t^* \leq t + \Delta t$ to compute the slopes $k_{t^*} = \Pi'(t^*)$. A weighted sum of those slopes is then calculated to approximate the average slope in the interval $[t, t + \Delta t]$ and to compute $\Pi(t + \Delta t) = y_{n+1}$ with the help of the average slope. All Runge-Kutta methods follow the same pattern:

$$\begin{aligned} k_i &= f\left(t_n + c_i \Delta t,\ y_n + \Delta t \sum_{j=1}^{s} a_{i,j} k_j\right) \\ y_{n+1} &= y_n + \Delta t \sum_{i=1}^{s} b_i\ k_i \end{aligned} \tag{2.4}$$

The methods differ by their number of slope estimates k_i and different coefficients $a_{i,j}$, b_i and c_i. The coefficients are usually arranged in matrix called the Butcher table:

c_i	a_{ij}				b_i
c_1	a_{11}	a_{12}	$\dots$	a_{1s}	b_1
c_2	a_{21}	a_{22}	$\dots$	a_{2s}	b_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
c_s	a_{s1}	a_{s2}	$\dots$	a_{ss}	b_s

For the sake of clarity, $a_{i,j}$ above the main diagonal with values of zero are usually omitted.

Euler's method does fit this pattern and can in fact be regarded as the lowest-order Runge-Kutta integration method.

c_i	a_{ij}	b_i
0	0	1

Slightly more accurate is Heun's method, a second-order ODE integration method that is also older than the development of the Runge-Kutta class of integration methods, but fits this pattern as well:

c_i	a_{ij}	b_i
0		$\frac{1}{2}$
1	1	$\frac{1}{2}$

For the case of computing the remaining probability $\Pi(t)$, Heun's method translates to the algorithm

$$\begin{aligned} k_1 &= \Pi_{stay}(t)\ \mu(t) \\ k_2 &= (\Pi_{stay}(t) - k_1\ \Delta t)\ \mu(t + \Delta t) \\ \Pi_{stay}(t + \Delta t) &= \Pi_{stay}(t) - \frac{k_1 + k_2}{2}\Delta t \end{aligned} \tag{2.5}$$

The most popular Runge-Kutta method, however, is a four stage, fourth order method simply called RK4. It is widely used because of its relatively high accuracy and yet low computational complexity. The Butcher table for RK4 is

c_i	a_{ij}			b_i
0				$\frac{1}{6}$
$\frac{1}{2}$	$\frac{1}{2}$			$\frac{1}{3}$
$\frac{1}{2}$	0	$\frac{1}{2}$		$\frac{1}{3}$
1	0	0	1	$\frac{1}{6}$

Error Estimation for Runge-Kutta Methods RK4 is sufficiently accurate for most applications. But while the order of the error of RK4 is known ($O(n^4)$), it might be desirable to estimate the actual error incurred per step. The simplest algorithm to do so is to solve the ODE over the whole interval $[t, t+\Delta t]$ twice: the first time with a single step and the second time with two steps, separately computing the solution for the interval $[t, t+\frac{1}{2}\Delta t]$ and then the interval $[t+\frac{1}{2}\Delta t, t+\Delta t]$. The difference between the single-step RK4 and the two-step RK4 results is then used as an error estimate. This approach, however, is computationally expensive, since it requires three complete RK4 integration steps instead of just one.

A more efficient approach is to compute two different numerical approximations with different orders of error for a given interval. The result with higher order is usually used as the numerical approximation and the difference between the two results is used as the error estimate. This only requires at most two complete integration steps compared to a single one when not computing error estimates. The computation can be made even more efficient with so-called *embedded* Runge-Kutta integration schemes. Here, the two integration methods use approximations at the same points t^* and hence an embedded Runge-Kutta integration steps requires only as many function evaluations of the function f as its higher-order method.

A very simple embedded Runge-Kutta integration scheme is ODE12, which uses Euler's and Heun's method for the two estimates. Its modified Butcher table is

c_i	a_{ij}	b_i	b_i^*
0		$\frac{1}{2}$	1
1	1	$\frac{1}{2}$	0

Where the b_i are the coefficients for Heun's method and the b_i^* are the Euler coefficients.

A far more accurate method is that of Dormand and Prince using fourth and fifth order integration methods. It is usually simply called ODE45 and has the following coefficients:

c_i	a_{ij}						b_i	b_i^*
0							$\frac{35}{384}$	$\frac{5179}{57600}$
$\frac{1}{5}$	$\frac{1}{5}$						0	0
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					$\frac{500}{1113}$	$\frac{7571}{16695}$
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$				$\frac{125}{192}$	$\frac{393}{640}$
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$			$-\frac{2187}{6784}$	$-\frac{92097}{339200}$
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		$\frac{11}{84}$	$\frac{187}{2100}$
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0	$\frac{1}{40}$

With this method, it should be possible to reliably compute transition probabilities and their errors in most cases.

2.3 Introduction to the Proxel Method

Numerical Approximation Computing the exact solution of Equation 2.1 is complicated at best and even impossible for transitions without an analytical solution for their cdf (and hence hrf), like the widely used normal and lognormal distributions. Therefore, the only feasible alternative is to approximate the solution numerically.

As seen in the previous section, the probability to leave a state during a time interval $[t, t+\Delta t]$ can be approximated numerically. Starting at time $t = 0$ with one or multiple initial system states and iteratively computing all possible transitions for fixed time steps of size Δt leads to the proxel method.

A proxel is defined[2] in [7] as the tuple $P = (S, t, R, p)$ where S again is a tuple $S = (m, \tau)$. The elements have the following meaning:

- S, the system state. It consists of
  - m for "marking" signifies the situation the system is in. For the simulation of stochastic Petri nets (SPNs)[2], m may be setup up to contain the actual *marking* of the SPN. However, other definitions of m are possible as long as there is a separate value of m for each reachable situation.
  - τ is the *age intensity vector* or simply the *age vector*. It is a vector storing the durations for all transitions to have been active since they last fired or became deactivated (for race-enable transitions).
- t is the simulation time at which the proxel exists.
- R stores the route, i.e. the complete list (or even tree when merging proxels) of predecessor proxels.
- p, finally, is the absolute probability to be in the state S at time t.

With these proxels, any stochastic system can be simulated. The complete standard proxel algorithm is shown below in pseudocode.

[2] A different definition is given in [3] omitting the R element. The definition given here is the most comprehensive one found

Algorithm 1: Pseudocode for the standard proxel algorithm.

```
Data: initialProxels, Δt, endTime
Output: nextStepProxels
currentProxels = initialProxels;
for step = 0 to (endTime/Δt) do
    nextStepProxels.clear();
    foreach Proxel P ∈ currentProxels do
        sumProbability = 0;
        foreach Transition trans ∈ P.activeTransitions do
            Proxel pSucc = P.createSuccessor(trans, Δt);
            sumProbability += pSucc.Probability;
            nextStepProxels.addOrMerge(pSucc);
        if sumProbability < P.Probabiliy then
            Proxel pInactive = P.ageBy(Δt);
            pInactive.Probability = P.Probability - sumProbability;
            nextStepProxels.addOrMerge(pInactive)
    currentProxels = nextStepProxels;
return nextStepProxels
```

To simulate a stochastic system using the proxel method, separate proxels are created for all initial states, that is for all states the system can be in at time t_0. Afterwards, for each proxel that exists at t_0, the transition probabilities for all transitions that are active at t_0 are computed for the interval of size Δt. The transition probabilities are then used to create all possible proxels at time $t_0 + \Delta t$, including those representing inactivity during the whole interval. For the created proxels, their markings reflect the new situation and their age vector is a modified version of their parent's age vector, where the age of all active transitions has been increased by Δt and the age of the transition that fired to create the proxel as well as that of all deactivated race-enable transitions is set to zero. Their t values are set to the end of the current time step, i.e. to Δt, their route is derived by adding the parent proxel to the route of the parent proxel, and p is set to the computed transition probability multiplied by the probability of the parent proxel.

With the complete set of proxels for time $t_0 + \Delta t$, the computation step can be repeated for the interval $[t_0 + \Delta t, t_0 + 2\,\Delta t]$ and all following intervals until the end of the simulation. The set of all proxels at t_{end} represents all possible states the system can be in at t_{end} and their probabilities. These can be analyzed to retrieve the desired simulation results.

For practical applications, the route R is usually not needed and is therefore omitted. Also, since the simulation is done iteratively for each time step, all proxels for a given time step share the same value of t. Consequently, t is usually only stored once for each time step and not for the individual proxels. Omitting t and R and

merging the individual values of S directly into the proxel definition yields the much simpler definition for a proxel of $P(m, \tau, p)$.

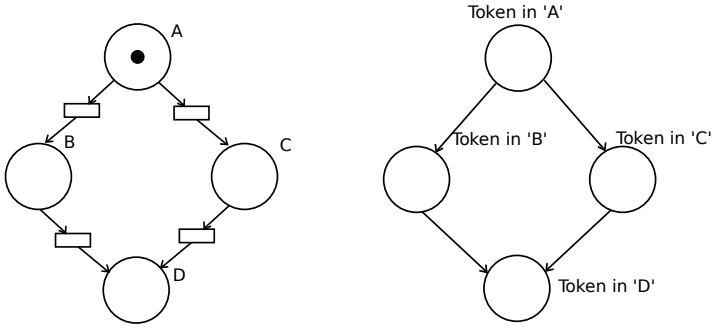


Figure 2.1: A small Petri net with three states and the corresponding state space.

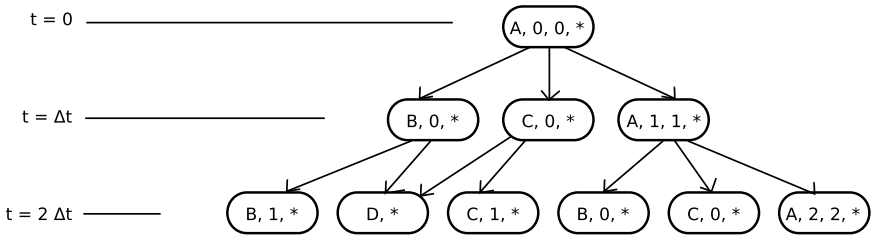


Figure 2.2: The graph of proxels for the model from Figure 2.1 for the first two simulation steps.

Figure 2.1 shows a simple Petri net with four transitions and four places, one of which is a sink ("D"). The proxel graph for the first two simulation steps of this model is shown in figure 2.2. The elements R and t of the proxels have been omitted for the reasons described before. The first value for each proxel is the marking, in this case simply the name of the state the only token is in. The asterisk in each proxel is a placeholder for the probability to be in the state at that time. The actual value depends on the probability distributions of the four transitions, but since they are not relevant for this explanation they have been omitted for the sake of clarity. All other values constitute the age vector. Since all transitions are race-enable, only the ages of the currently active transitions have to be stored. All other transition ages will always be zero and can therefore be omitted. Hence, the age vector contains two elements for proxels with marking "A" (for the transitions leading to "B" and "C"), one element for proxels with markings "B" or "C" (for the transition leading to "D"), and no elements at all for proxels with marking "D".

As the routes R are no longer part of the proxel, it can happen that two proxels

with identical marking and age vector $P_1(m, \tau, p_1)$ and $P_2(m, \tau, p_2)$ are created at time t and hence represent the same system state at the same time, reached through different routes. For the model in Figure 2.1, this happens for the proxel corresponding to the state that the token has just reached the place "D", which can be reached from the places "B" and "C". Instead of treating these two cases as different proxels and computing their successors separately, the two proxels can simply be merged into a single proxel $P(m, \tau, p_1 + p_2)$, saving computational effort and memory. While merging proxels may seem like an irrelevant feature, it is actually the mechanism that makes proxel simulation of many models feasible: Most proxels spawn at least two child proxels, one for the firing of an active transition and one for inactivity. Thus, without merging, the number of proxels would double for each time step, making it impossible to simulate models with more than about 40 time steps. With proxel merging however, the number of proxels usually grows much slower and for most models even converges to a constant value. Mathematically, the proxel graph containing merged proxels is no longer a tree, but a layered directed acyclic graph.

2.4 Known Sources of Error in Proxel Simulations

The proxel approach for the simulation of stochastic models is only an approximation of the model's behavior. Instead of solving a system of partial differential equations, it approximates the behavior by discretizing the time into steps of size Δt and approximates the transition probabilities in each interval by numerical integration. This leads to at least two sources of inaccuracies[7]:

Violation of the Basic Assumption The proxel approach simulates the possible state changes for a certain state during the interval $[t, t + \Delta t]$ by calculating the transition probabilities during that interval for all transitions that are active at time t, but not for those that have become active some time after t. Hence, proxel simulation assumes that no transition can become active and fire during a time interval, and consequently that no more than one subsequent state change can occur during a single time step. This assumption is called the basic assumption of proxel simulation. For most models, this assumption is violated and introduces a first order ($O(n)$) error, meaning that the magnitude of the error is proportional to the step size chosen.

Integration Error Since the computation of the transition probabilities for a time step is carried out using a numerical approximation, an integration error is introduced. The current implementations of the proxel algorithm use Euler integration introducing a second order ($O(n^2)$) local truncation error (error per time/integration step) and hence a first order ($O(n)$) global truncation error to the simulation result.

More accurate integration methods exist, but they have not yet been used for proxel simulations. Until now, the rationale was that the biggest source of error will determine the overall simulation accuracy and that reducing the integration error will not increase the accuracy unless the error introduced by violating the basic assumption is reduced as well.

2.5 Richardson Extrapolation

The proxel method approximates a simulation result $f(x, h)$ with a step size h. When reducing the step size h, the error of the simulation result is usually also reduced. But reducing the step size increases the computation time necessary and therefore cannot be repeated indefinitely.

A somewhat different approach to reducing the error is to use simulation results computed for multiple step sizes to derive more accurate results. As one example, one could plot the simulation results $f(x, h)$ against the step size h and use the plot to estimate the simulation result for even smaller step sizes, for example if all plotted points lie close to a straight line. But there also exists a mathematical theory behind this idea, which allows for an automated computation of these estimates: Richardson's method of extrapolation.

As first described by Richardson in 1927 [10], when a function $f(x)$ is approximated depending on a step size h, the relation between the (unknown) exact solution and the approximation $\Phi(x, h)$ can be written as the infinite sequence

$$\Phi(x, h) = f(x) + hf_1(x) + h^2 f_2(x) + h^3 f_3(x) + ...$$

where all $f_i(x)$ are unknown error terms. If the order of the error (*not* the error itself) made by the computation of $\Phi(x, h)$ is known, all error terms with a smaller error vanish. For an error of order n, the equation above is reduced to

$$\Phi(x, h) = f(x) + h^n f_n(x) + h^{n+1} f_{n+1}(x) + ...$$

For small enough values[3] of h, all terms $h^i f_i(x)$ with $i > n$ are substantially smaller than $h^n f_n(x)$ and can therefore be neglected, leaving

$$\Phi(x, h) \approx f(x) + h^n f_n(x) \tag{2.6}$$

Here, $f_n(x)$ is still unknown, but computing $\Phi(x, h)$ for two different step sizes h_1 and h_2 yields two equations with the structure of Equation 2.6, and these can be solved

[3] There is no universal formula to determine what h is "small enough" and so this has to be found through experiments.

for $f_n(x)$ and then be equated, eliminating the $f_n(x)$ term. Solving the resulting equation for $f(x)$ yields

$$f(x) = \frac{h_2^n \, \Phi(x, h_1) - h_1^n \, \Phi(x, h_2)}{h_2^n - h_1^n} \tag{2.7}$$

This is an extrapolation of the real result $f(x)$ using the two approximations $\Phi(x, h_1)$ and $\Phi(x, h_2)$ with step sizes h_1 and h_2, respectively. Since the highest order error term $h^n f_n(x)$ was eliminated during computation, the result is usually more accurate than any of the approximations used to calculate it.

However, it must be noted that the computation of $f(x)$ by Richardson's method relies on a weak assumption: It assumes that h is so small that all error terms other than the first one are minuscule. But since the error terms depend not only on a power of h, but also on the unknown values $f_n(x)$, this assumption might not hold and cannot even be checked. Thus, the results of the Richardson extrapolation cannot be assumed to always be more exact than the estimates used to compute them.

The same approach that led to Equation 2.7 can also be followed using more than two estimates for $f(x)$ and more than two different step sizes to eliminate additional error terms and gain even more accurate results. However, since the basic assumption of Richardson's method does not necessarily hold, this could also amplify the error instead of reducing it.

Applicability to the Proxel Method Since the proxel method calculates results based on a time step size Δt, the Richardson extrapolation is applicable to the proxel method. According to a thorough analysis of the proxel method [7], the error introduced is of first order. Using Richardson's approach, this highest order error term can be eliminated. The two step sizes are usually chosen as an arbitrary Δt and $\frac{\Delta t}{2}$. There are no exact formulas to determine the minimally acceptable step size for which the results of the Richardson extrapolation are more accurate than those of the values computed, but [7] gives heuristics based on the mean firing time of all transitions in the system.

Given two simulation results $\Phi(\Delta t)$ and $\Phi(\frac{\Delta t}{2})$, Equation 2.7 yields for a first order error

$$f = \frac{\frac{\Delta t}{2}\Phi(\Delta t) - \Delta t \Phi(\frac{\Delta t}{2})}{-\frac{\Delta t}{2}} = 2\Phi(\frac{\Delta t}{2}) - \Phi(\Delta t)$$

Which corresponds to a linear extrapolation of the theoretically exact value of $\Phi(0)$. This way, a result is obtained which is usually much more accurate than $\Phi(\frac{\Delta t}{2})$, without significantly increasing the computational effort.

2.6 Impulse Rewards

With the methods and facilities introduced so far, the simulation results measured can only be the probability of the system to be in a certain state at a given time. The expressiveness of this result set is somewhat limited. Instead, one is often interested in the number of times a transition has fired during a simulation run. This performance measure is called an *impulse reward* [2]. An impulse reward can be triggered along with any state change and changes the value of a counter outside of the simulation model (i.e. the value of the counter cannot have an influence on the behavior of the model). The increment can be a constant (e.g. to simply count the number of occurrences of the event) or can be a computational result based on the time the event was triggered. The property of the performance counter to stand outside of the simulation model allows it to hold values other than positive integers (as would be the case for SPNs) and enables complex computations on the counter value. For state-space simulation methods like the proxel method, this has the additional advantage that the performance counter does not enlarge the state space and therefore does not contribute to the state-space explosion.

Using multiple counters with different increments for the same event can be useful to calculate more complicated results. For example, if one counter measures the number of times a transition has fired and another one counts the sum of these firing times, the quotient of both value represents the average firing time.

2.7 Summary

This chapter introduced the underlying mathematical concepts that describe stochastic models as well as the proxel method as a promising method to solve these models that is being modified for this thesis.

The chapter also explained the errors introduced by the proxel method. These are important, because the VTS approach tries to limit the overall error of the simulation result and thus needs to control the errors incurred per time step.

Furthermore, two ways of retrieving more accurate results for proxel simulations without having to reduce the simulation step size were introduced: Richardson extrapolation and more exact numerical integration methods. Richardsons method of extrapolation is an important way for the CTS algorithm to retrieve more accurate results and will hopefully show to be applicable to the VTS method as well. Higher-order integration methods can help reduce and thus control individual errors.

In the next chapter, this information is used to develop a proxel algorithm that supports variable time steps.

Chapter 3

Variable Time Steps in Proxel Simulations

This chapter is concerned with the theory of variable time steps for the proxel method. Implementing variable step sizes requires the completion of several subtasks:

First, a simulation algorithm must be created that supports varying time steps during a single simulation run. The task can be split further into the decision on how to allow the time domain to be split (Section 3.2), deciding on what entities to apply the time steps to (Section 3.3) and finally to create the simulation algorithm based on these two decisions (Section 3.4).

Second, criteria must be developed that can determine the optimal step size for a given transition in a given situation (for example the number, type and age of active transitions). Ideally, a single criterion would suffice. But as no criterion was known to work beforehand, multiple possible criteria are implemented to be tested later in Section 5.6. Since developing step size criteria requires in-depth understanding of the error sources in proxel simulation and an estimation of their magnitude, the different error types are explored in detail in section 3.1. The criteria are then developed and explained in Section 3.5.

Finally, the importance of being able to apply Richardson's method of extrapolation to the VTS algorithm is explained in Section 3.6.

3.1 Analysis of Errors Inherent to the Proxel Method

In this section, the error sources inherent to the proxel method are explained in detail. In addition to the errors introduced in Section 2.4, a newly discovered error source is covered. The reasons for the proxel method to exhibit each error are explained. Afterwards, ways of reducing and estimating the (possibly reduced) magnitude of the errors are presented.

Error through Violation of the Basic Assumption

As detailed in Section 2.4, proxel simulation introduces an error by assuming that only a single consecutive state change can happen per time step. If two or more

consecutive state changes would happen but only one is simulated, probability will move slower through the system than it should analytically.

Mitigating the Effects There is no known way to reduce this effect for a given model other than to reduce the time step size. Reducing the step size helps insofar as it is less likely for two consecutive state changes to occur during a smaller time step than it is during a bigger one.

Estimating the Error The error estimate follows the assumption that the error introduced is essentially that of failing to simulate exactly two consecutive state changes during a time step. This assumption seems reasonable, since for more than two state changes to happen consecutively, two have to happen first.

Thus, the error estimate of the probability that two state changes can happen during the interval $[t, t + \Delta t]$ is computed as the worst-case: the case that the first state change occurs instantaneously at time t and the second one can happen at any time between t and $t + \Delta t$. Hence, the error estimate is computed as follows:

- for each active transition i at time t, the transition probability $P_i(X < t + \Delta t | X > t)$ is computed. A temporary target proxel with probability P_i is created. It exists still at time t and has the same age vector as the parent proxel, with the one exception that the age of the transition that fired to create it is cleared.
- for all of those target proxels, the probabilities of all possible state changes during the next Δt are computed.
- the probability sum of all of those 2nd-level proxels is the estimate for the probability that two state changes can happen during a single time step

Thus, the magnitude of the error through violation of the basic assumption (BA error) can be estimated and this estimate can be used to influence the time step size used in the VTS algorithm.

Error through Numerical Integration

As explained briefly in Section 2.4, proxel simulation introduces an error through numerical integration of the differential equation for the probability to stay in the current state

$$\frac{d\Pi}{dt} = -\Pi(t) \times \mu(t) \tag{3.1}$$

That equation only holds if there is only one single active transition. For a stochastic system with the active transitions T_1 through T_n, the probability Π to leave the

current marking through transition n firing is given by

$$\frac{d\Pi_n}{dt} = \Pi_{stay}(t) \times \mu_n(\tau_n)$$

where τ_n is the age of transition n, i.e. the time span that the transition has been active since it last fired or got disabled. If the transition has been active ever since $t = 0$ and has never fired, τ_n equals t.

The probability to stay in the current state is consequently given by the differential equation:

$$\frac{d\Pi_{stay}}{dt} = -\sum_{i=1}^{n}(\Pi_i(t) \times \mu_i(\tau_i))$$

For proxel simulations with constant time steps, it made sense to assume that the error made by violating the basic assumption is of the same order as that of the Euler integration [7], and that it does not make sense to improve the integration accuracy without being able to reduce the other error term. But with variable time steps and separate error estimates for the different error types, there will likely be situations where the error through violation of the basic assumption is negligible compared to the Euler integration error. In these cases, one could either simulate with a smaller step size (forfeiting all advantages of variable time steps), or increase the integration accuracy through other means and simulate with a bigger step size.

Mitigating the Effects Higher-order integration methods (cf. Section 2.2) can be used to reduce the error made by integrating ordinary differential equations (ODE error) remarkably, if only a single transition is active at a time. This, however, is not the case for most systems and consequently the accuracy gain over the Euler method often marginal (cf. experiments in Section 5.3).

Fortunately, the Runge-Kutta method (cf. Section 2.2) can be modified to accommodate for a Π_{stay} whose value is changed by multiple transitions: For multiple active transitions, the corresponding transition probabilities ODEs have to be solved for the same time interval, and they all use approximations of $\Pi_{stay}(t^*)$ at the same points in time t^*. Thus, it makes sense to compute a shared approximation of $\Pi_{stay}(t^*)$ for all active transitions.

The computation of all $k_{1,i}$, i.e. of the k_1 for the ith transition, depends only on the probability $\Pi_{stay}(t)$ and hence no accuracy is lost when computing them separately. Having those, the probability to leave the state through each transition before t^* can be computed and $\Pi_{stay}(t^*)$ is simply the remaining probability mass, i.e.

$$\Pi_{stay}(t^*) = \Pi_{stay}(t) - (t^* - t)\sum_{i=1}^{n} k_{1,i}$$

With this shared estimate for $\Pi_{stay}(t^*)$, the approximations for all $k_{2,i}$ can be computed, and these can in turn be used to compute the shared Π_{stay} approximate for the next point in time.

For the Heun method (cf 2.2), the complete modified algorithm to compute the transition probabilities for n concurrently active transitions is as follows:

$$\begin{aligned} k_{1,j} &= \Pi_{stay}(t)\ \mu_j(\tau_j) \\ k_{2,j} &= (\Pi_{stay}(t) - (\Delta t \sum_{i=1}^{n} k_{1,i})\ \mu_j(\tau_j + \Delta t) \\ \Pi_{leave_j}(t+\Delta t) &= \frac{\Delta t}{2}(k_{1,j} + k_{2,j}) \\ \Pi_{stay}(t+\Delta t) &= \Pi_{stay}(t) - \frac{\Delta t}{2}\sum_{i=1}^{n}(k_{1,i} + k_{2,i}) \end{aligned} \tag{3.2}$$

All other Runge-Kutta integration methods (e.g. RK4, ODE45) can be modified in the same way to accurately compute the transition probabilities for multiple concurrently active transitions. The Euler and Trapezoid integration methods do not rely on estimates of Π_{stay} and therefore do not need to be modified.

Estimating the Error As shown in Section 2.2, the ODE error can be estimated either by computing the result for the given interval with different steps (e.g. one or two steps) or with integration methods of different order. The latter is part of *embedded Runge-Kutta* integration methods (ODE12, ODE45) while the former can be applied to all integration methods, but is computationally more expensive.

For this project, the following integration methods were implemented. Where applicable, a method was implemented to parallely solve systems of ODEs as well as single ODEs:

- Euler's method
- Trapezoid integration
- Fourth order Runge-Kutta (RK4)
- Combined Euler and Heun methods (ODE12)
- Dormand-Prince method (ODE45)

Concluding, the integration error per time step can be made almost arbitrarily small by using a variety of higher-order integration methods. Using embedded Runge-Kutta methods, the error can also be estimated with very low computational overhead. This estimate may be suited to influence the time step sizes used for the VTS algorithm.

Error through Non-Deterministic Behavior During a Time Step

The existing literature on proxel simulation mentions only the two error sources explained in the previous sections. But a simple experiment suggests that there is at least one additional error source:

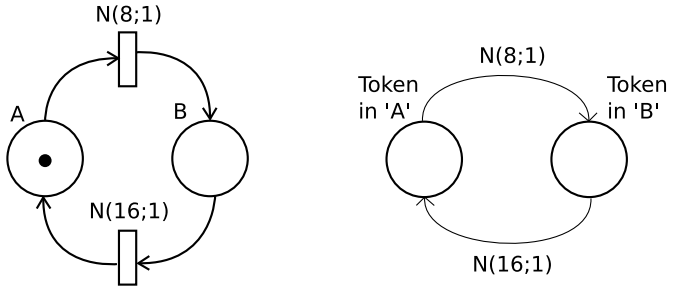


Figure 3.1: A Circular Petri net and the corresponding state space

For the simple model in Figure 3.1, the probability in each state can only leave its marking through a single transition and no transition is ever deactivated before all probability has left the state. Hence, the transition probabilities can be computed analytically using the cdf (cf. Section 2.1) and do not need to be approximated, completely eliminating the integration error. Also, the two normal distributions have a rather small standard deviation and hence fire in a very narrow age interval. The probability for the transition with $N(8,1)$ to fire with an age $\tau \leq 1$ is only about 1.28×10^{-12}, the probability for the other transition to fire with that age is even smaller. Hence, when simulating with time steps of size $\Delta t = 1$, the error through violation of the basic assumption is smaller than 1.28×10^{-12} per time step. So the simulation result for the time at which the stationary solution is reached ($t \approx 4000$) should differ by no more than

$$1.28 \times 10^{-12} \times 4000 = 5.12 \times 10^{-9}$$

from the analytical solution. And yet, according to the simulation results with the standard proxel method for $\Delta t = 1$, the stationary solution of the probability to be in state "A" would be about 0.659996, while the analytical solution (and the value the simulation results converge to for $\Delta t \to 0$) is exactly $\frac{2}{3}$. The difference between the two is bigger than predicted by several orders of magnitude.

Since the two known error sources were eliminated for this model, there must exist at least one additional source of error to explain the discrepancy. It seems likely that this error is the result of the proxel method's inherent limitation to only represent the system at times that are multiples of the step size Δt: Instead of a continuous flow of probability, the transition is simulated as a set of deterministic transitions,

one for each time step firing exactly at the end of the time step interval. This leads to at least two different errors:

- Impulse rewards are only triggered at the end of an interval instead of continuously during the whole interval. As a consequence, the triggered rewards may not adequately represent the whole range of possible triggering times and may exhibit a bias error, since they are always triggered too late.
- Age values for all transitions are always multiples of the step size instead of being distribution functions representing all possible continuous transition times. Consequently, the age values recorded are always smaller than would be correct, since transitions are simulated as having fired at the end of an interval instead of at any time during the interval.

Both contribute to the simulation error in ways that are still unclear.

Mitigating the Effects The inability to represent the system's behavior during a time step is inherent to the proxel method, but some of the resulting errors can be reduced.

The first resulting error is caused by the selection of the point in time at which an *impulse reward* is triggered. Since the computation for an impulse reward may depend on the exact time of triggering, a good estimate for that time is important for high simulation accuracy.

The conservative approach to determine that time is based on the observation that the system was in one state at time t and is in a new state at time $t+\Delta t$. Since time $t+\Delta t$ is the first time at which the system is observed to be in the new state, it makes sense to trigger the impulse reward at time $t+\Delta t$.

However, another approach is possible: It is known that the transitional behavior is based on a continuous distribution function, and hence the transition could have fired at any point in time during that interval. Since only one impulse reward should be triggered to represent all of those possible transition times, the time chosen to trigger the reward should be the average firing time during that interval. The average firing time would be the time at which half of the probability that is going to leave the state in the current time step has already left the state. Its exact value depends on the number and type of all active transitions and hence is rather complicated to evaluate. But for small time steps (relative to the active transitions' rates), a reasonable approximation is to use the midpoint of the interval $[t, t+\Delta t]$, i.e. $t+\frac{1}{2}\Delta t$.

Following the same line of argument leads to another possible improvement: When a transition has fired during a time step, the resulting proxel usually "inherits" the ages of all race age transitions from its preceding proxel and it sets the ages of all newly activated race-enabled transitions to zero. But as seen above, the state

change can happen at any time during the interval $[t, t + \Delta t]$. Furthermore, it was suggested that this behavior can be reasonably approximated by the mean firing time, estimated as $t + \frac{1}{2}\Delta t$. So the transition has fired on average at time $t + \frac{1}{2}\Delta t$, but the resulting proxel is only being created at the end of the time step. Thus, at time $t + \Delta t$, all transitions that got disabled for the target proxel have been active for about $\frac{1}{2}\Delta t$ before they got disabled, and all transitions that got enabled for the target proxel are already active for $\frac{1}{2}\Delta t$. These findings can easily be incorporated into the proxel algorithm to improve its accuracy. The age τ_i of the ith transition of the successor proxel S should be derived from the parent proxel P with by the following rules

- for recently activated race-enable transitions: $S.\tau_i = \frac{1}{2}\Delta t$
- for recently activated race-age transitions: $S.\tau_i = P.\tau_i + \frac{1}{2}\Delta t$
- for recently deactivated race-age transitions: $S.\tau_i = P.\tau_i + \frac{1}{2}\Delta t$

The age of recently deactivated race-enable transitions is always set to zero, so there is no need to increase them by $\frac{1}{2}\ \Delta t$ beforehand. The rules for deriving all other age values do not need to be changed.

More accurate approximations for the mean firing time are possible, for example by iteratively calculating the transition probabilities for the interval $[t, t + t_{mean}]$ and varying t_{mean} until the leaving probability for that interval is half the leaving probability of $[t, t + \Delta t]$.

However, computing a more accurate mean firing time implies that there will be numerous different firing time estimates during a single simulation run. As a consequence, the age values of different proxels would be increased by those different values and hence would no longer all be multiples of the step size. Thus, the probability to have two proxels a time t with the same marking and the same age vector would be close to zero. But the efficiency of the proxel method crucially depends on its ability to identify and merge those identical proxels (cf. Section 2.3). So increasing the accuracy of the firing time estimates would come at the cost of vastly accelerating the state-space explosion and therefore would actually reduce the overall efficiency instead of increasing it.

Thus, when modifying the firing time to increase the simulation accuracy, $t + \frac{1}{2}\Delta t$ should be used as the new firing time.

Estimating the Error The error through non-deterministic behavior during a time step (ND error) is barely understood and consequently is very difficult to estimate. One first crude estimate is based on the basic error source, i.e. on the fact that in a proxel simulation, a transition fires only either at the end or in the middle (when modifying the firing times) of a simulation step, while in the actual system, the

transition's firing time is usually stochastically distributed over the whole interval. Thus, the more probability leaves the state at the beginning of the time interval (and at the end, if the firing time is set to be in the middle of the interval), the higher is the inaccuracy introduced by the ND error.

To estimate the ND error, the following algorithm is used:

Procedure `getNDErrorEstimate(`*transition, age,* Δt`)`

Input: transition, age, Δt
begin
  fireTime = $\frac{5}{10}$
  sumDeviation = 0
  sumProbability = 0
  foreach $pos \in \{\frac{1}{10}, \frac{3}{10}, \frac{5}{10}, \frac{7}{10}, \frac{9}{10}\}$ **do**
    prob = transition.getCdfValue(age + (pos + $\frac{1}{10}$)*Δt) -
    transition.getCdfValue(age + (pos - $\frac{1}{10}$)*Δt)
    sumProbability += prob
    sumDeviation += prob * $|pos - fireTime|$
  return *sumDeviation/sumProbability*
end

The time interval in question is split into five equally-sized subintervals (five was chosen as trade-off between accuracy and computational overhead) and the transition probabilities are computed for all sub intervals using the difference of the cdf values of the interval borders. Each probability is multiplied by the distance of the interval center to the firing time (either the center or the end of the time interval) and the products are summed up.

The ND error estimate is simply that sum divided by the probability to leave during the whole interval. This division turns the result into a relative error, independent of the actual probability that leaves the state. This is important, because the transition probability computations via the cdf are wrong by an almost constant factor for all situations where more than one transition is active. Since numerator and denominator of the result both contain that factor, the division cancels it out.

The resulting estimate does not represent an error probability, but is only a characteristic number. It simply increases when more probability leaves the state further away from the firing time. Thus, it cannot be compared directly to the other error estimates.

But still, this estimate may be suited to determine suitable step sizes for the VTS algorithm.

3.2 Possible Time-Division Schemes

The main task for this thesis is to develop and implement a proxel algorithm with variable time steps. The first step in this process is to determine how to subdivide the time domain. This decision is a very difficult one with far-reaching consequences. A good time-division scheme should have the following properties. It should:

- allow for a dynamic subdivision of the time domain at any point in simulation time
- maximize the number of points in simulation time at which proxels can be merged
- limit the number of possible age vectors
- allow for an efficient merging of proxels

The first criterion describes the degree of flexibility of the subdivision scheme. The least flexible division of time would be constant time steps. The most flexible one would allow each transition at each time to have an arbitrary time step size. Since variable time steps should adapt as well as possible to the current system state and the currently active transitions, a more flexible subdivision of time should allow for a more accurate result for a given number of proxels.

The second criterion is an important one, because merging proxels is the only known way to limit the state-space explosion. Since proxels can only be merged when existing at the same point in time, the number of points in time where multiple proxels exist should be maximized.

The third criterion is also concerned with the possibility of merging proxels, since the efficiency of the proxel algorithm crucially depends on the ability to find and merge proxels with the same system state, i.e. the same marking and identical age vectors. The fewer different age vectors can exist, the higher is the chance that the age vectors of two proxels with the same marking are identical and hence the two proxels can be merged.

Finally, the scheme must support an efficient way of finding proxels that could be merged. For CTS, all merge candidates were proxels at the same time step (cf. Section 2.3), i.e. proxels inside the same container. Consequently, to identify proxels with identical system states, only a single container needed to be searched. For variable time steps, a time-division scheme might have multiple containers into which a new proxel could be merged. In that case, each insertion would need to scan through all of these containers before adding a proxel.

No single subdivision scheme can perfectly fulfil all requirements, because the requirements contradict each other: For example, an arbitrary subdivision of time fulfils the first criterion, but it also creates an arbitrary number of possible age

vectors at arbitrary point in time and hence is not able to satisfy the second and third criterion. However, different compromises are possible.

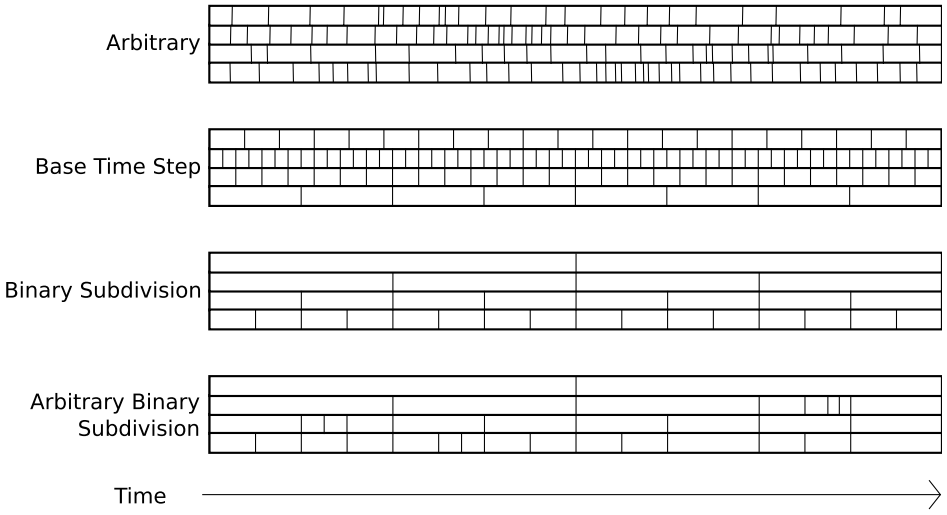


Figure 3.2: Different schemes for dividing the time axis

Some of these compromises are displayed in Figure 3.2. The figure shows the time steps taken for four sequences of proxels in each algorithm. Points in time at which proxels can be merged are those for which the vertical lines are above each other.

Arbitrary Subdivision As explained above, the scheme allowing arbitrary time steps is the most flexible one. However, it is unable to limit the number of possible different age vectors and to create points in time at which proxels can be merged. Thus, it accelerates the state space explosion. These disadvantages alone make it unfeasible to use in an efficient algorithm.

Base Time Step The second approach was first described by Wickborn and Horton [12, 13], and further analyzed by Köberle [6]. They first determine a suitable basic time step and then compute the actual time step size used for each transition as a multiple of the basic time step. Therefore, their time steps are constant for each transition and cannot include dynamic criteria - severely limiting the algorithm's flexibility. On the other hand, this limit guarantees that the age of a transition is always a multiple of its basic time step, and that circumstance effectively limits the number of possible age vectors. Also, according to [6], this algorithm requires each proxel created for time $t + \Delta t$ to check all active time steps between t and $t + \Delta t$ for proxels with identical system states. Hence, the computational complexity of the algorithm grows not only with the number of proxels per time step, but also with

the number of active time steps. And this number can become large if numerous transitions are active and their step sizes are almost prime.

Binary Subdivision The third approach is the binary subdivision of time. Here, the whole simulation time is seen as a single time step. The time step size used can only be reduced by recursively subdividing a time step into two equally-sized halves. Hence, if a transition at time $t = 0$ is to be simulated with $\frac{1}{4}th$ of the simulation time, the whole interval needs to be subdivided in two equal halves, and then the first half has to be divided again, yielding two quarters. Thus, even if no further subdivision of time is necessary for any other transition, the time domain is already split into two quarters and one half of the simulation time. Consequently, after the first transition is simulated with a quarter of the whole simulation time, the remaining simulation cannot be computed in a single time step of $\frac{3}{4}$ the simulation time, it has to be simulated in two steps of size $\frac{1}{4}$ and $\frac{1}{2}$. Furthermore, it is not possible to simulate a model with the time steps of size $\frac{1}{4}$, $\frac{1}{2}$ and $\frac{1}{4}$ in this order, because the first step of the binary subdivision of time would already split the time at $\frac{1}{2}$ and this split cannot be removed afterwards.

The advantages of making these limitations are manifold: First, there exists no basic time step and the subdivisions can be arbitrarily small, albeit always of size $1/2^n$ for positive integers n. While being arbitrarily small, the step sizes still fit the binary subdivision pattern. Consequently, all age values will always be multiples of $1/2^i$ where i is the highest subdivision recursion depth used. This at least somewhat limits the number of possible age vectors.

Second, the subdivision only at fixed points in time has the consequence that many proxels exist and can be merged at these subdivision points. All sequences of proxels will meet at the start and end times of the simulation. Almost all sequences of proxels will also meet at the middle of the time interval, the only exceptions are those that are simulated with the whole simulation time as a single time step. In general, if a point in time t is a multiple of $1/2^i$ for an arbitrary integer i, all proxel sequences simulated with a step size $\Delta t \leq 1/2^i$ will also contain proxels at time t and hence can possibly be merged at that time. Similarly, if two transitions at time t are simulated with step sizes $1/2^i$ and $1/2^j$ with $i \leq j$, then proxels of both sequences will exist at time $t + 1/2^j$ and can potentially be merged. This is in stark contrast to the base time step algorithm, where transitions at time t could be simulated with step sizes of 4 and 5 times the base time step and in that case would not share common points in time until $t + 20\ \Delta t_{base}$.

Finally, a binary subdivision simplifies the determination of the time step sizes. While other algorithms must numerically compute viable time step sizes, the binary subdivision algorithm must only decide whether the currently chosen step size is small enough or if it needs to be split in half. This simplifies the development of

suitable subdivision criteria and can potentially speed up the computation.

Arbitrary Binary Subdivision In Figure 3.2, the binary subdivision was depicted with constant step sizes for each sequence of proxels, i.e. the recursion depth for each sequence has been chosen in advance and was kept during the course of the whole simulation. But this limitation is not inherent to the algorithm. It was merely applied to better convey the basic idea of the binary subdivision. To allow more flexibility, this limitation can be dropped, allowing the algorithm to choose to subdivide any interval at any time.

This added flexibility does not impact the performance of the algorithm. All properties explained in the paragraph *Binary Subdivision* apply to the arbitrary binary subdivision as well.

Choosing a Time-Division Scheme Due to its flexibility and expected high performance, the arbitrary binary subdivision was chosen. Since the simple binary subdivision has more limitations but not more benefits, it cannot be expected to perform equally well. The base time step algorithm, on the other hand, may have a similar performance to the arbitrary binary subdivision, but it has already been tested by other researchers. Also, due to its dependence on statically determining the step size for each transition in advance, it cannot support dynamic subdivision criteria, which are thought to perform better than static ones.

Inherent Weaknesses of the Binary Subdivision Approach

The time-division scheme of arbitrary binary subdivision is the most promising one and is consequently the one selected to be implemented and tested. However, it is important to understand its inherent weaknesses, which can degrade the approach's performance no matter how effective the selected subdivision criterion is:

The first weakness results directly from the binary subdivision. If a desired time step is of size $1/2^i$, this requires i recursive subdivisions of time and at least i more simulation steps, even if no further event can happen. For example, to simulate a model with a single deterministic transition at time 1/16 up to time 1, the deterministic transition will almost certainly (depending on the subdivision criterion) require a time step size of 1/16 for the first time step. Simulating this time step requires a recursive subdivision of the respective first half of each time step at times 1/2, 1/4, 1/8 and 1/16. Hence, even if no further transitions can fire until the end of the simulation, the model must be simulated with consecutive time steps of sizes 1/16, 1/8, 1/4 and 1/2 to reach the end of simulation at time 1. While this example of certain inactivity is an extremely rare situation, the general problem of the subdivision scheme to force subsequent time steps to be smaller than necessary will be observed in nearly every model.

The second weakness stems from the fact that the binary subdivision only allows time steps of sizes $t_{end}/2^i$ and hence possible simulation time steps differ by at least a factor of two. While this allows time steps to be almost arbitrarily small while only requiring a few subdivision steps and hence a shallow recursion depth, this coarse selection of time step sizes can be suboptimal. Two transitions for which the decision factors for subdivision are very similar should also be simulated with the same step size. But if one of these factors is slightly above the subdivision threshold and the other one slightly below it, the two transitions would actually be simulated with steps sizes that differ by a factor of two.

These two weaknesses clearly limit the overall efficiency of the selected approach. It is possible that their impact even undoes the efficiency gained by the variable time steps for at least some simulation models.

3.3 Selection of a "Sphere of Influence" for the Time Steps

After a time division scheme has been selected, it must be chosen what entities the different time steps should be applied to. There are three different approaches. A single time step size could be computed for and applied to:

- all successors of all proxels of a time step
- all successors of a proxel
- each transition separately

These three choices will be explained in detail in the next paragraphs. The examples are based on the simple model in Figure 3.3.

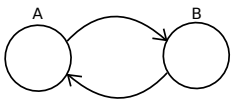


Figure 3.3: A circular reachability graph

One Step Size per Time Step Simulating all proxels of a time step with the same step size (cf. Figure 3.4) closely resembles the standard proxel algorithm. The algorithm could easily be implemented in an iterative fashion, since the successor proxels of all proxels of a given time step exist at the same time and thus can be placed into the same container. However, the downside of this approach is that the actual step size used has to be a compromise depending on all proxels of a time step. For most simulation models, proxels for numerous system states exist at any given

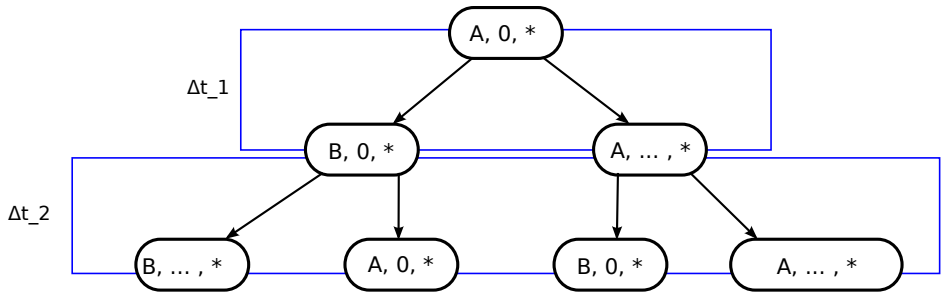


Figure 3.4: One step size per time step

time and thus, the actual step size will tend to be about the same for each time step, effectively going back to constant time steps.

This property makes the application of step sizes to a whole time steps unsuitable. Thus, the approach was not pursued any further.

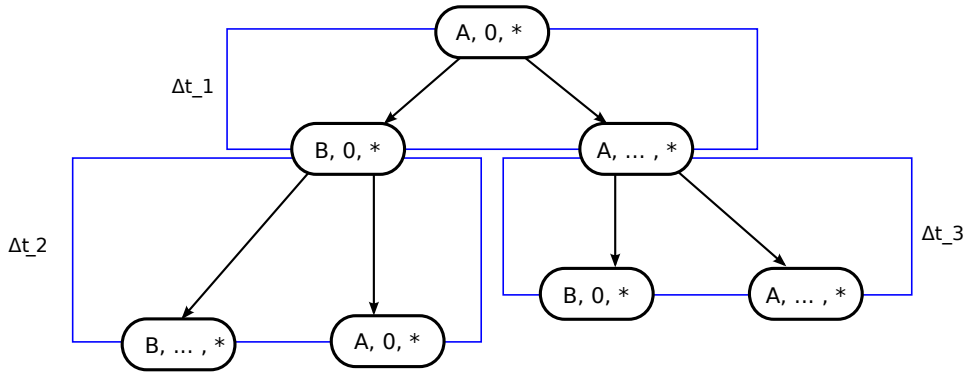


Figure 3.5: One step size per proxel

One Step Size for all Successors of a Proxel Simulating each proxel with its own step size (cf. Figure 3.5) seems like a natural choice. As seen in Section 3.1, the transition probabilities for all active transitions of a system state depend on each other and hence need to be computed in parallel. Also, the probability to remain in a state for a time step depends on the probability to leave the state for all active transitions during that time step. Consequently, all of these transition probabilities must be known for the same time step size.

The downside of this algorithm is that the step size chosen still needs to be a compromise between all active transitions of a proxel's marking. Since the step size will usually depend on the fastest transition (the one with the highest rate), this

approach is very ineffective in situations with multiple active transitions with vastly differing rates and therefore unsuitable for rare-event simulation. Unfortunately, many simulation models that are difficult to simulate with the proxel method and therefore are thought to benefit most from variable time steps have at least some rare-events.

Thus, no further time has been invested in investigating this approach.

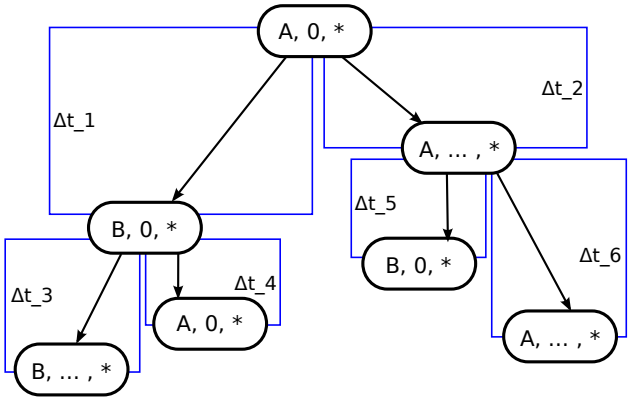


Figure 3.6: One step size per transition

Simulating all Transitions Separately Subdivision criteria are naturally developed for single transitions. A step size could be chosen depending on a transition's rate or the transition probability during a time interval. When computing a single time step for all successors of a proxel or even for all proxels of a time step, the criteria for each transition are simply combined to yield an overall criterion. But this combination process is always a compromise and looses information about the optimal step size for the individual transitions.

So it makes sense to directly apply the individual step sizes to the individual transitions (cf. Figure 3.6). While this approach adopts best to a particular situation and seems to be suitable for the simulation of rare events, it causes some algorithmic difficulties. First, since the transitions leaving a marking can be simulated with different step sizes, it is not clear how to compute the probability to stay in a state, or even how to determine the point in time for which this inactivity proxel is to be created. Second, as Section 3.1 showed, computing the transition probabilities for the transitions individually causes some error. This error is even amplified when the transitions are simulated with different step sizes.

But since these difficulties are not inherent weaknesses and may be solved, simulating all transitions separately is a viable candidate for implementation.

Decision Even though applying the different step sizes to individual transitions causes some difficulties and still poses open questions, it nevertheless is the most promising approach. Hence, it was chosen to be implemented. The solutions for the problems described in the previous paragraph are part of the next section.

3.4 Development of the Simulation Algorithm

Developing the simulation algorithm involves integrating the arbitrary binary subdivision scheme (cf. Section 3.2) with a facility to simulate all transitions with arbitrary time step sizes (cf. Section 3.3), as well as solving the algorithmic difficulties caused by allowing the latter. Since the solution to the algorithmic problems may influence the integration with the subdivision scheme, the algorithmic problems are solved first.

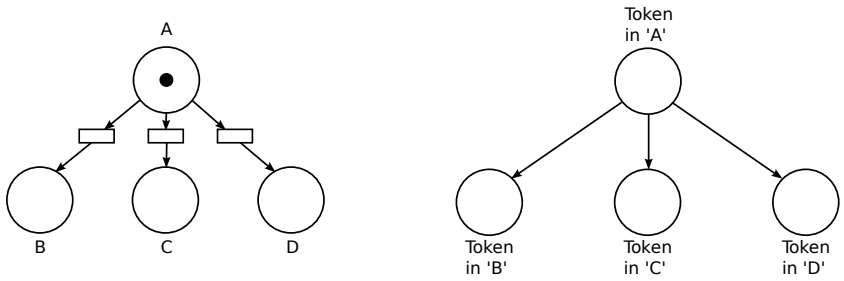


Figure 3.7: A sample model to illustrate the algorithmic difficulties, shown as a Petri net and the reachability graph

Solving the Algorithmic Problems Figure 3.7 shows a simple stochastic model where the transitions leading to the places B, C and D are to be simulated with the step sizes 1, 2 and 4, respectively. The corresponding proxel tree for the naive implementation is shown in the left of Figure 3.8. The proxel probabilities replaced by an asterisk can be computed, but those replaced by a question mark cannot. For example, the probability to still be in state A at time $t = 1$ is the probability to be in state A at time $t = 0$ minus the probability to leave the state A for the states B, C and D at time $t = 1$. But the probabilities to leave for states C and D at time $t = 1$ have not been computed. Simply ignoring them would cause the probability to be in state A at time $t = 1$ to be too high. Using the calculated probabilities of leaving for C at time $t = 2$ and for D at time $t = 4$ would cause it to be too low.

So it seems that the probabilities to leave the state through all active transitions must be computed for all time steps. This approach is illustrated on the right-hand side of Figure 3.8: All transition probabilities for $t = 1$ are calculated and hence the

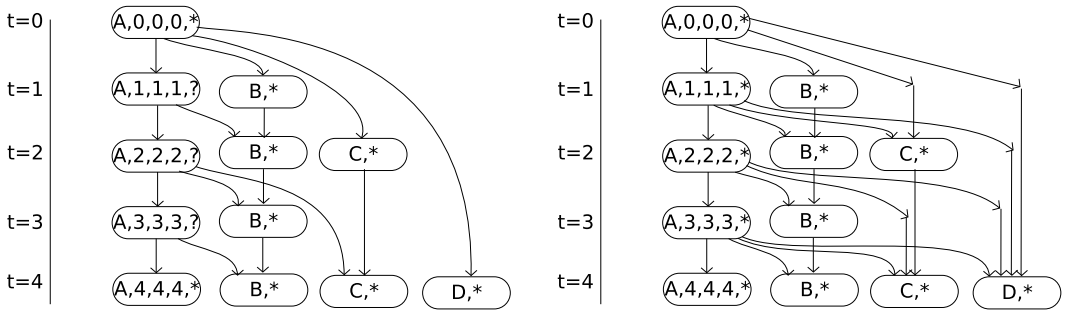


Figure 3.8: The beginning of the proxel tree for the model in Figure 3.7. Shown are the naive approach (left) and a consistent computation (right).

probability to stay in marking A can be computed. However, not all corresponding proxels are created and stored. Instead, the probability to leave for the markings C and D is *redirected* to the proxels at times $t = 2$ and $t = 4$. Thus, until time $t = 4$, only two proxels exist with the marking C and one with the marking D - the same result as with the naive approach. However, a higher number of transition probability computations was necessary to restore the consistency of the algorithm when compared to the naive approach.

Extending the Proxel Definition The idea to redirect probability allows a consistent simulation with variable time steps. But that approach requires a facility to store which transition probability should be redirected to what proxel. One could try to determine whether a proxel exists to which the probability could be redirected whenever a transition probability is computed. But this approach is time-consuming. It is far more efficient to directly store this information inside the proxels themselves. For example, the proxel $(A, 0, 0, 0, *)$ in the right column of Figure 3.8 would directly store that the probability to leave for state D is to be directed to the proxel $(D, *)$ at time $t = 4$ for the next four time steps. It would also hand this information down to the proxel $(A, 1, 1, 1, *)$ and the following inactivity proxels until $t = 4$. This way, the lookup of the proxel the probability is being redirected to only takes constant time.

To allow storing this information, the proxel definition $P(m, \tau, p)$ (cf. Section 2.3) is extended to $P(m, \tau, p, R, D)$, where R and D are both new vectors. R is the vector of probability redirections and stores the target proxels for the redirection of probability for each transition that is active in marking m. D is the corresponding vector of redirector recursion depths. It stores the subdivision recursion depths for which each of the redirectors is valid. If a transition *trans* at time t is to be simulated with recursion depth i, i.e. with step size $t_{end}/2^i$, a target proxel is created at time $t + t_{end}/2^i$, and a redirector to that proxel is added to the current proxel for

transition *trans* and the corresponding redirector recursion depth for that transition is set to i.

Redirectors and redirector recursion depths are cleared for all proxels that are the result of a transition firing and they are generally inherited by proxels created due to inactivity during a time step. Only if an inactivity proxel has reached one or more of its redirector recursion depths, the depths and their corresponding redirectors are cleared.

Finalizing the Algorithm With the probability redirectors ready, the simulation algorithm can be finalized. The complete algorithm in pseudocode is shown in figure 3.9.

When a transition of proxel P is chosen to be simulated with recursion depth i, a proxel is created at $t_{end}/2^i$ time units in the future of P representing the target system state (the state the system would be in after the transition has fired) and having a probability of zero. A redirector to that target proxel is added to P with a redirector recursion depth of i.

If any active transition for the marking $P.m$ is *not* redirected, then at least one transition of P needs a further subdivision of time and therefore a copy of P is added to the list of proxels needing further subdivision. If all transitions are redirected, no further subdivision is necessary and all transitions are computed with the step size of the current recursion depth. A target proxel for inactivity is created and all transition probabilities are redirected to the proxels stored in P's redirector vector.

The output container depth is the recursion depth at which the output container will be read. Since all recursion steps with the same end time share the same output container, they also share the same output container depth. More specifically, if a time step $[t, t + \Delta t]$ is subdivided, a temporary output container is created to hold the results of the first subdivision half $[t, t + \frac{1}{2}\Delta t]$ and hence the output container recursion depth for that recursion step is the same as its recursion depth. The second subdivision half $[t + \frac{1}{2}\Delta t, t + \Delta t]$ uses the output container of its parent recursion step and therefore inherits its output container depth (cf. Figure 3.10). The reason that the output container depth needs to be passed down at all is that an inactivity proxel needs to clear a redirector if the output container depth is less than or equal to the redirector recursion depth.

3.5 Design of the Subdivision Criteria

So far, the development of the subdivision criteria was not considered. It was simply assumed that a valid subdivision criterion exists. This approach was indeed a suitable one, since the development of the simulation algorithm is independent of the actual subdivision criterion used. Furthermore, developing and describing both

Procedure `simulateVTS`(t_{start}, t_{end}, *inputProxels, outputProxels, depth, outputContainerDepth*)

```
Input: t_start, t_end, inputProxels, depth, outputContainerDepth
Output: outputProxels
Data: proxelsSubdivide, proxelsTmp
begin
    Δt = t_end - t_start
    foreach Proxel P ∈ inputProxels do
        foreach Transition trans ∈ P.transitions do
            if P.D[trans] is set then
                continue
            if not P.shouldSubdivideTimeStep(trans, Δt) then
                Proxel pSucc = P.createSuccessor(trans, Δt, outputContainerDepth)
                pSucc.D.clear()
                pSucc.p = 0
                outputProxels.addOrMerge(pSucc)
                P.R[trans] = pSucc
                P.D[trans] = depth
        if P.allTransitionsRedirected() then
            pProxelsOut.addOrMerge( P.createInactivitySuccessor(Δt,
            outputContainerDepth))
            foreach Transition trans ∈ P.transitions do
                P.R[trans].p+= P.createSuccessor(trans, Δt, outputContainerDepth).p
        else
            proxelsSubdivide.add(P)
    if proxelsSubdivide.numProxels > 0 then
        simulateVTS (t_start, t_start + Δt/2, proxelsSubdivide, proxelsTmp, depth +1,
        depth +1)
        simulateVTS (t_start + Δt/2, t_end, proxelsTmp, outputProxels, depth+1,
        outputContainerDepth)
end
```

Figure 3.9: Pseudocode of the VTS algorithm.

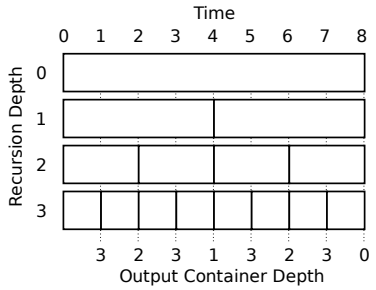


Figure 3.10: Visualization of the relationship between recursion depth and output container depth

parts separately reduces the complexity.

As the simulation algorithm has been finalized, the development of effective and efficient subdivision criteria is the last step to be taken in order to develop a fully functional proxel simulation algorithm with variable time steps. This section discusses five different possible subdivision criteria and explains the rationale behind each criterion.

Limiting the Target Probability

The first criterion, *GLOBAL_PROB*, tries to globally limit the probability mass p of each proxel to a fixed value. This appears to be reasonable considering that the error made by simulating a proxel during a time step is usually proportional to the probability mass of that proxel. So, if a proxel has a probability mass of 0.5, it will be responsible for about half of the simulation error during that time step. Even if the remaining 0.5 probability mass is distributed over thousands of proxels, the single proxel with the high probability will mostly be responsible for the simulation accuracy. So it reasonable to split the time step for that proxel and try to distribute the probability mass equally over all proxels.

The actual criterion of GLOBAL_PROB is simply to compute the transition probability $transProb$ for each transition of each proxel P, and to subdivide the time step for that transition, if $transProb * P.p > k$ for a constant $k \in [0, 1]$. The exact value of k will determine the number of subdivisions necessary and hence will enable the user to choose a trade-off between simulation speed and accuracy.

GLOBAL_PROB will not always be able to effectively limit the probability of the target proxel. A target proxel may be the result of merging multiple proxels, thus the resulting merged proxel may have a probability mass of $P.p > k$. This flaw, however, is only of minor concern, since a proxel with a merged probability mass of more than k will be subdivided by the algorithm when it is simulated itself.

Limiting the Transition Probability

Another approach to increase the simulation accuracy is to directly limit the transition probability allowed per time step. The transition probability determines what fraction of the probability of the current system state will leave that marking during a time step. The larger this fraction is, the bigger the part of the transition's hazard rate function that needs to be integrated in a single time step, and the higher is the integration error and potentially also the ND and BA errors (cf. Section 3.1).

The criterion, called TRANS_PROB, therefore subdivides a time step if $transProb > k$ for a constant $k \in [0, 1]$.

Limiting the Transition Probability Fraction

The third subdivision criterion was developed in order to simulate every transition with about the same number of steps, so that in each step about the same amount of probability leaves the marking. This criterion, called BASE_PROB, is supposed to ensure that each target proxel contains about the same probability mass and hence contributes equally to the overall simulation error. The rationale for this criterion is that each proxel needs the same computation time and therefore should equally contribute to the simulation result.

The way to have the same amount of probability leave a marking in every time step is to limit the transition probability to a fraction of the base probability, i.e. the probability to be in the marking right after the most recent firing of a transition. Thus, this base probability needs to be stored in each proxel. For proxels being the result of a transition firing, the base probability is the same as the proxel's probability. For proxels that are the result of inactivity during a time step, the base probability is inherited from its predecessor. If two proxels are merged, their base probabilities are added.

So the preliminary computation for BASE_PROB is to subdivide a time step, if the probability mass of the child proxel would be higher than a certain fraction of the parent proxel's base probability, i.e.

$$transProb * P.p > k * P.p_{base} \tag{3.3}$$

A small problem in the algorithm needs to be corrected: Time steps sizes are chosen such that in each time step, only a certain fraction of the base probability leaves the state. For a $k = 1/n$, the whole transition will be simulated in n steps and in each time step, the nth fraction of the base probability will leave the time step. But after $n - 1$ time steps, only $1/n$th of the base probability is left and the criterion states that this amount of probability is allowed to leave the state in a single time step. Therefore, the last time step would be arbitrarily large, even if all probability

would have left the marking long before the end of that time step[1]. To prevent this behavior, this criterion is slightly modified: If all of the remaining probability leaves the state during the current time step, it is checked, whether the same is true for only the first half of the current time step. If this is the case, then the time step is subdivided. Otherwise the predicate known from Equation 3.3 is used to determine the need for subdivision.

Following the Transition Means

A criterion that has already been used before is to determine the step sizes based on the mean firing times of the transitions. It was first used by Wickborn and Horton [13] and is base on research [7] suggesting that the simulation results tend to converge to the exact value if the simulation step size for each transition is less than $\frac{1}{4}$th of the transition's mean. So it stands to reason that this observation can be generalized and that all transitions behave equally well if they are all simulated with a step size that corresponds to the same fraction of their mean value. This criterion is simply called MEAN and it subdivides a time step, if

$$trans.mean * k > \Delta t$$

for a $k \in [0, 1]$. This criterion is static in that the resulting step sizes can be determined before the simulation is even started. This has the big advantage that the step sizes need only to be calculated once. Also, since the criterion does not involve any transition probability, the transition probabilities only need to be computed for those time steps that are actually simulated and not for those that are going to be subdivided (as is the case with the other criteria).

On the other hand, a static criterion is far less flexible, since it cannot adapt to a given system state. This may impact its simulation efficiency.

Limiting the Error per Time Step

The purpose of a good subdivision criterion is to reduce the simulation error as far as possible while allowing the computation to be carried out using as few proxels as possible. All aforementioned criteria indirectly tried that by limiting other parameters. But it should be possible to directly estimate the error made per time step and reduce the step size if the error is too high.

As shown in Section 3.1, there are three different sources of error for the proxel approach, and all of them can be estimated. So a suitable criterion is to check

[1] Analytically, a small amount of probability would always remain in the state for distributions that do not have a limited support. In practice, however, all implemented methods for solving ODEs slightly overestimate the leaving probability at the end of the distributions, numerically making the support range finite.

if the error from each source is below a certain threshold and then to require a UNANIMOUS vote to not subdivide the time step. In other words, if the error from any source is higher than its threshold, the time step is subdivided.

This approach should theoretically yield the most accurate simulation results for a given number of proxels, as long as the error estimates are good enough. On the other hand, the computations for the error estimates are quite complex and hence time consuming, meaning that with this criterion fewer proxels can be simulated during the same time as would be possible with other criteria.

A problem to be solved is how to determine the thresholds for the three criteria. All other criteria only had a single threshold and that one could be varied to determine the trade-off between simulation accuracy and speed. But with three thresholds, varying one of them can easily increase the computation time without increasing the accuracy (which might be limited by the other two error sources). Thus, determining the ratio between the three sub-criteria is still an open question.

Determining the Thresholds To determine useful thresholds for UNANIMOUS, the simulation algorithm with constant time steps was used to analyze the criteria's numerical ranges. The "Warranty" model (cf. Section 5.2) was simulated with a step size of 20 and for each transition simulated the three error estimates were computed and analyzed. "Warranty" was chosen, because it is a real-life model, but still computes relatively accurate results for big step sizes[2]. The parallel version of ODE45 (cf. Section 2.2) was used as the integration method, since it provides the most reliable integration error estimates. Figure 3.11 shows the histogram of the distributions of each error source.

The integration error per time step has a range of several orders of magnitude. The biggest error occurred is in the range of about 10^{-7}, but this one occurred only very infrequently.

The BA error is usually in the range of about 10^{-6} with its maximum being about as high as the maximum integration error. It was to be expected that the BA error is otherwise far higher than the integration error obtained when using ODE45, since the latter produces only an error per time step of $O(n^6)$, while the former error per time step is of order $O(n^2)$.

The value of the ND error is not a probability measure, but rather a characteristic number. It therefore cannot be compared directly to the two other error values. But even analyzed by itself, this error estimate proves to be useless, since it returns about the same value in every situation observed. This can mean either one of two things: First, the error estimate may not accurately reflect the actual error incurred. In this case, the error estimate cannot be used. Second, it may show that the ND error only depends on the simulation step size and not on other characteristics. In this

[2]Even for a time step size of $\Delta t = 20$, the log file recording the errors amounts to about 10MB.

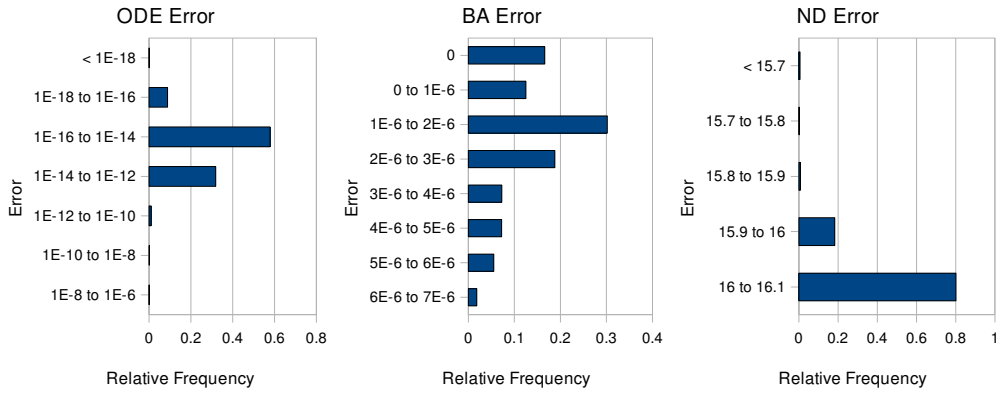


Figure 3.11: The error histogram for the ODE, BA and ND errors (cf. 3.1), measured for the "Warranty" model. The histogram buckets are chosen so that every single error value occurred fits into some bucket. The buckets for the ODE error are not equally sized, but encompasses two orders of magnitude each.

case, the estimate would effectively force an algorithm with variable time steps to use a constant time step size to limit the error per time step.

As a conclusion, the ND error estimate was removed from the UNANIMOUS criterion. This will negatively impact the accuracy of the criterion, since the ND error is not estimated and hence a time step might be chosen not to be subdivided when an optimal ND error estimate would cause a subdivision. However, the remaining criterion might still prove to be accurate enough if the ND error is either very small compared to the other two errors, or if it is of about the same magnitude and strongly correlates with at least one of the two other errors.

The BA and integration errors are consequently chosen to use the same threshold. Since both error estimates represent probability values, using the same threshold means that both error sources are allowed to cause the same error. While this might not be the optimal solution in all cases, it eliminates the need for multiple thresholds and hence the need to separately select those. Using only a single threshold allows a simple decision between simulation accuracy and computation time.

3.6 Significance of Linear Convergence and Richardson Extrapolation

One important property of the CTS proxel method is that for small step sizes (the exact value of "small" depends on the model simulated), the simulation results con-

verge linearly to the analytical solution. This property can be used in two ways: First, when having obtained linearly convergent results, the simulation step size chosen is certain to be sufficiently small for the simulation results to be accurate. For big step sizes, the relationship between the step size and the simulation result can be chaotic and a smaller step size may lead to a less accurate solution. For linearly convergent results, however, the error can be estimated, since with two linearly convergent simulation results for step sizes Δt and $2\Delta t$ the following holds in most cases:

$$error(\Delta t) \approx |res(\Delta t) - res(2\Delta t)|$$

A formal proof of that linear convergence (i.e. of the Proxel method to exhibit a first order error) was given in [7]. While the proof seems to be flawed[3], the linear convergence has been observed in every single model simulated to date and thus the assumption of linear convergence itself seems to be correct.

The second and equally important application of that property is that when the error has been established as a first order one, the Richardson extrapolation (cf. Section 2.5) can be applied to two simulation results obtained for different step sizes to extrapolate the theoretical value for $\Delta t = 0$. This extrapolation is not usually identical to the analytical solution, but it is much more accurate than the two results that were used to compute it. Thus, a more accurate solution could be obtained without increasing the computational effort.

Both applications of the linear convergence are important when using the proxel method for practical applications. Thus, in order for the VTS approach to superseed the currently used CTS one, the convergence of the simulation results needs to be shown for VTS as well.

For CTS, the results are linearly convergent when plotted against the time step size Δt. For VTS, no global time step exists, so the convergence has to be shown subject to the threshold k, since it is the only user-selectable parameter. As k has different meanings for each subdivision criterion, the convergence has to be shown for each criterion separately.

A formal proof of convergence is extremely difficult to obtain and is therefore beyond the scope of this thesis. Instead, experiments concerning the applicability of the Richardson extrapolation will be conducted in Section 5.7 that are suited to either support or refute this hypothesis.

[3] In [7], the formula $\int_{\tau}^{\tau+\Delta t} \mu(x)dx$ is used to analytically calculate the transition probability. The correct formula, however, should be $e^{\int_{\tau}^{\tau+\Delta t} \mu(x)dx}$ (cf. [11], p244.4), and even that does only hold for a single active transition.

3.7 Conclusion

This chapter described the theoretical backgrounds of variable time steps for the proxel approach. An in-depth analysis of the errors occurring in the proxel simulation was conducted along with noting ways to reduce and estimate the individual errors. Afterwards, a time division scheme was chosen and it was decided to apply different time steps to all transitions individually. With this knowledge, the final simulation algorithm was developed.

Afterwards, multiple criteria were designed that are able to decide whether a given time step should be divided. These criteria attempt to reduce the simulation error by limiting the transition probability per time step, the absolute probability of a target proxel, the fraction of transition probability to base probability, the actual errors incurred (or rather their estimates) per time step, and the fraction of time step size to transition mean, respectively.

Finally, the importance of being able to apply Richardson's method of extrapolation to the VTS proxel method was emphasized. It was decided to collect empiricial evidence rather than to give a formal proof about the applicability of Richardson's method to the simulation results obtained from the proxel algorithm with variable time steps.

Chapter 4

Implementation

All previous chapters where concerned with the theory of variable time steps for the proxel approach. In order to be of any value, these theoretical constructs need to be tested experimentally and thus must be implemented.

The process of application development and implementation is not the topic of this thesis and is therefore not described. What this chapter does, however, is to describe and justify implementation decisions that either differ from previous implementations of the proxel algorithm, are thought to be of general interest for future implementations, or that describe facilities necessary for the experiments conducted in the next chapter.

Section 4.1 lists basic decisions made that influenced the overall development process. Section 4.2 describes facilities that were put in place to allow for a more exact and rapid experimentation process. Finally, improvements that increase the robustness and speed of the simulator application are the topic of Section 4.3, while the process of ensuring equal quality of the implementation of the VTS and CTS algorithms is explained in 4.4.

4.1 Basic Implementation Decisions

Programming Language To test the algorithms and criteria proposed, it was necessary to implement a proxel simulator. C++ was chosen as the programming language. Using C++ instead of C may slightly reduce the speed of the simulator application, but it does not impact the validity of the test results, since the CTS and VTS algorithms are both implemented in C++ using the same programming style and even share most of the code.

Class Structure C++ allows object-oriented programming and this implementation makes use of that paradigm by implementing the simulation models and the components of their reachability graphs as classes. Markings (i.e. elements of the reachability graph) are a class containing an identifying string for debugging purposes and a list of active transitions. The general interface for distributions is implemented as an abstract base class, with different kinds of distributions (Normal,

Weibull, ...) being subclasses and instances of a distributions with an actual set of parameters are instances of those subclasses. Transitions are modelled as classes as well. Each transition contains a reference to the associated distribution and one to the target state. It also contains an array of changes to the activation state for each transition of the system, to determine whether the transitions get activated, deactivated or stay active or inactive when the current transition fires. The `Transition` class can be subclassed to implement different impulse rewards.

Complete models, finally, are represented as an abstract `Model` base class. It assembles the model with the help of the classes described before, can return a list of initial proxels for the model as well as the models t_{end}. Finally, it evaluates the result proxel set of a simulation run to retrieve the desired simulation results. Individual models are implemented as subclasses of that `Model` class.

As a consequence, models need to be compiled into the application and cannot be changed at runtime or loaded dynamically. This reduced flexibility would make the application unfeasible for industrial applications. However, this is not an issue for academic purposes, where the research on the algorithms requires constant change and thus recompilation of the application anyway. Having the model selected at compile time allows the determination of the exact size of the proxels' age vectors at compile time. Hence, the age vector can directly be embedded into the `Proxel` class and does not require an additional dynamic memory allocation for each proxel, simplifying the application code and making it slightly faster.

4.2 Facilities to Simplify Experimentation

Application Output The application outputs various information including the actual simulation results as determined by the model, computation time, statistical information about the simulation run and various sets of debugging information. For test series, it is desirable to have a machine-readable version of the simulation results. To achieve this, the significant simulation output is reported twice: a human-readable version is written to the standard output and a single line of comma-separated values suitable for further processing by a spreadsheet application is written to the standard error output. These two outputs can easily be separated. For example, the standard output can simply be omitted, or the machine-readable standard error output can be appended to a text file to collect results of multiple simulation runs.

Automated Testing The application has many parameters that can be changed:

- The model can be simulated using the CTS or the VTS algorithm.
- One out of eight integration methods can be chosen.

- For VTS, different subdivision criteria can be selected
- For the VTS subdivision criteria, different thresholds can be chosen.
- For CTS, the simulation step size can be selected.
- The firing of transitions can be set to happen at the middle or the end of a time step.
- Proxels with a very small probability (in this implementation $< 10^{-15}$) can be omitted to reduce simulation time without noticably changing the simulation results.

Selecting a different step size for CTS or a different criterion threshold for VTS is the parameter that is most frequently varied for testing and can consequently be chosen as an application parameter on the command line without requiring recompilation. All other settings require changes to the algorithms and necessarily require recompilation. To change one of these parameters, one would need to manually comment out all code sections corresponding to the previous setting and removing the code comments from all sections for the new setting. This process is time-consuming and error-prone. Instead, the application relies on conditional compilation through the `#ifdef` and `#if` preprocessor directives. For each possible parameter, a preprocessor constant is chosen and all code sections for that parameter are enabled or disabled based on whether the constant is set. This way, all code sections responsible for a parameter are enabled and disabled at the same time.

The preprocessor constants are defined and selected globally in a separate header file so that all changes can be managed from a single location. This header file also checks the set of defines for validity, e.g. whether exactly one integration method and exactly one model is selected. Furthermore, the file assembles a comma-separated string containing all selected options. This string is written to the standard error output along with the simulation results to allow for determining the exact simulation environment for each result. The `#define` directives are implemented in a way to allow them to be overridden by compiler parameters. This has the advantage that shell scripts can be created that automatically compile the application with many parameter sets and conduct multiple tests with each.

Measuring the Computation Time For the experiments, it is important to accurately measure the computation time for each simulation run. The simulator measures this time itself (instead of using an external tool like the UNIX program `time`), so that the value does not include building the simulation model and loading the application itself, and to be able to output the computation time along with the other simulation results.

It was chosen to use the UNIX functions returning application ressources usage in order to determine the computation time. A simple watch-like measurement of the system time before and after the simulation run would only return the *duration* of the simulation run, which can externally be influenced by other applications running in parallel, some of which cannot be controlled for (schedules tasks, network traffic, ...). In contrast, the ressource-usage functions return the cumulated CPU time used by the application, which is barely influenced by concurrently running applications and therefore yields nearly deterministic results.

4.3 Increase of Robustness and Speed

Proxel Container Both, the CTS and all VTS algorithms require a container to store a set of proxels and allow finding mergeable proxels. C++ supports the generic programming paradigm and consequently allows writing the algorithm to work with any container that conforms to an implicit interface. Three different containers were implemented for this simulation application:

- A generic binary search tree.
- An AVL tree [4], a self-organizing binary search tree.
- A dynamic array.

The generic binary search tree allows insertions and finding duplicates in $O(log(n))$ in the average case, but needs $O(n)$ in the worst case. The AVL tree guarantees $O(log(n))$ even for the worst case of both operations, but has a slightly bigger constant overhead. The dynamic array needs $O(n)$ to find duplicates, but has an amortized $O(1)$ for insertions. Theoretically, the AVL tree should be superior to the regular binary search tree in almost all situations. However, since the opposite has been reported in [9], both trees were implemented to be able to compare them.

The dynamic array is not suitable for finding identical proxels, since it requires $O(n)$ for each search. But there are situations in the VTS algorithms where that operation is not necessary: When deciding to subdivide the time step for a proxel, that proxel is copied to the subdivision container. Since the source container already merged all possible proxels, the subset of proxels that is copied to the subdivision container does not contain any mergeable proxels any more. In this case, the dynamic array is much faster than any binary tree. As the VTS algorithm is implemented as a template function, it can be called with any input proxel container type without the need to duplicate code.

Robustness of the Numerical ODE Solutions For distributions with limited support, all probability will have left the state when the end of the support is reached. When numerically solving ODEs, however, some probability may be left in the state after the end of the support has been reached. Since the hazard rate function (cf. Section 2.1) is either zero or undefined for all points in time after the end of the support, the remaining probability would never leave the state. To prevent this, it was chosen to define the hazard rate function value of all distributions with limited support as ∞ for all times after the end of the support. The rationale is, that all probability should have left the state at the end of the support and, if not, should at least leave the state as early as possible after the end of the support. A value of ∞ ensures that all probability leaves the state with the next evaluation of the hazard rate function.

Furthermore, all algorithms implemented to compute the transition probabilities are algorithms to solve general ODEs. But since they are used only to compute transition probabilities, some additional constraints can be enforced to improve the accuracy of the results.

First, all transition probabilities have to be in the range $[0, 1]$ and can be clamped to that range. For the Runge-Kutta multi-step methods (cf. Section 2.2), this also applies to all intermediate approximations of the remaining probability.

Additionally, care must be taken when using multi-step methods with distribution functions with limited support either naturally (deterministic, uniform distributions) or through computational inaccuracies (normal distribution). The hazard rate function of those distributions is ∞ at the end of the support and beyond that. When the hazard rate function evaluates to ∞, the next estimate for the remaining probability is $-\infty$ and is consequently set to zero[1]. Part of the computation for the next slope estimate is the product of the remaining probability (*zero*) and the hazard rate function at the next evaluation point (∞). This product is undefined and evaluates to a floating point value of NaN (*Not a Number*). All operations involving a NaN value will evaluate to NaN themselves, invalidation all simulation results. To prevent this behavior, all slope estimates are checked and the computation is cancelled when the first non-finite value is detected. Since an infinite slope indicates that all probability will have left the state at that time, continuing the computations is not necessary.

4.4 Equal Quality of the Algorithm Implementations

In order for the experimental results to be significant, it needs to be ensured that different results of the CTS and VTS algorithms stem from the differences between

[1] Since zero is the lowest-possible probability value.

the algorithms themselves, and not just from the different levels of quality of their implementations.

In order to ensure this non-discrimanatory implementation (cf. Section 1.4) of both algorithms, they are implemented in a way to share as much code as possible. The more code is shared, the more is the quality of both implementations likely to be comparable.

The classes that represent the simulation models along with their state transitions and underlying distributions are independent of the algorithm that uses them and can therefore easily be shared. The same is true for the computation of transition probabilities, which is moved to a separate function and shared between both implementations.

The algorithms, however, need different representations for the proxels themselves, since the VTS algorithm requires additional elements in the proxels. To allow code-sharing, the `VTSProxel` class is directly derived from the `Proxel` class used by the CTS algorithm.

All container classes are implemented as template classes and can thus be shared by the implementations even though the containers store different types (`Proxel` or `VTSProxel`) for each algorithm.

Therefore, the only code that is not shared is the actual simulation algorithm. But with all the low-level code shared, the simulation algorithms amount to no more than 100 lines of code per algorithm, as opposed to the several thousand lines of code of the whole application.

4.5 Summary

This section gave an overview over implementation details that are thought to be of interest. First, the selection of C++ as the programming language was motivated and the class structure created was described.

Then, the application facilities for human-readable and machine-readable output formats were explained along with the preprocessor macros used to automate testing and a short introduction to exact computation time measurement on UNIX-like operating systems.

Afterwards, multiple containers to store and lookup proxels were presented, and it was shown how to improve the robustness of the numerical integration methods.

Finally, is was explained that the equal quality of the CTS and VTS implementations was ensured by sharing almost all code between the two.

Chapter 5

Experimental Verification

5.1 Expectations and Goals

So far, multiple hypotheses were stated (either implicitly or explicitly) in this thesis:

- The simulation accuracy can be increased by using higher-order ODE solvers (cf. Section 2.2).
- The simulation accuracy can be increased when adjusting the firing times (cf. Section 3.1).
- The simulation performance can be increased by using an AVL tree as the proxel container instead of a regular binary search tree (cf. Section 4.3).
- An efficiency increase can be expected when using variable time steps (cf. Section 1.2).

While all of them were stated in the hope that they would show to be correct, every single one of them needs to be tested experimentally to be either supported or refuted. In addition to verifying these hypotheses, the following open questions should be answered through experiments to form a sound opinion on the viability of variable time steps for the proxel method:

- Which subdivision criterion (cf. Section 3.5) for the VTS algorithm is the most efficient one?
- How high is the computational overhead of the VTS algorithm?
- For which VTS subdivision criteria is the Richardson extrapolation (cf. Section 2.5) applicable?

The remaining chapter covers the experiments conducted in order to answer the aforementioned questions. The next section introduces the simulation models used in the experiments. Since some of these models are used in multiple experiments, they are introduced beforehand. All other sections of the chapter cover experiments to test the individual hypotheses and answer the open questions.

5.2 Simulation Models

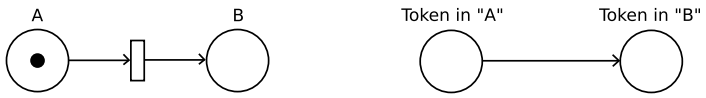


Figure 5.1: The "Leaving" model, displayed as a Petri net and reachability graph.

"Leaving" One of the most trivial simulation models possible is the "Leaving" model (cf. Figure 5.1). It simulates only a single transition that is only active once. Hence, the only system behavior is that of probability *leaving* state A for state B over time. After a certain amount of time, all probability will have entered the second state, so measuring the remaining probability is not a viable parameter to be observed. What can be measured and observed, however, is the average firing time of the transition (measured through impulse rewards, cf. Section 2.6). This is especially useful since that mean firing time can also be determined analytically for most transitions, allowing a comparison of the exact results with the measured ones. The simulation error exhibited in this model should almost exclusively be the integration error, since the *basic assumption* cannot be violated by a single transition and the error through non-deterministicness should barely influence the computation of the result (which is an average).

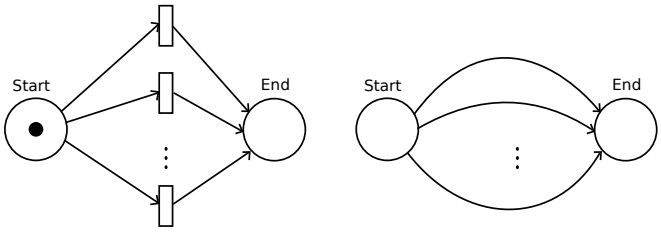


Figure 5.2: The "Leaving_n" model, displayed as a Petri net and reachability graph

A variation of the "Leaving" model is the "Leaving_n" model (cf. Figure 5.2). Here, probability can leave state A for state B through multiple transitions. This model, too, should cause no error but an integration error. Since numerous transitions are active at the same time, parallel integration methods should perform significantly better than those computing the transition probabilities for each transition separately.

For most probability distributions, the resulting mean firing time cannot easily be computed from the analytical mean firing times of the individual transitions. Hence, to retrieve the exact solution to which the experimental results could be compared, the model needs to be simulated once with a large number of time steps.

"Chain" The chain model (cf. Figure 5.3) is a linear chain of places, each connected to its successor by a single transition. The number and type of transitions can be varied.

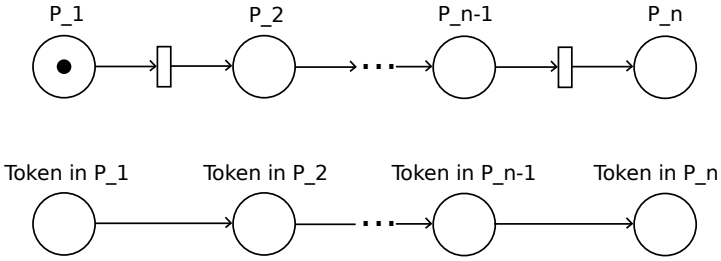


Figure 5.3: The "Chain" model, shown as a Petri net and reachability graph

The model is more complicated than the simple "leaving" model and a simulation run may exhibit all three types of errors. Since it does not contain cycles nor active race age transitions, the simulation state space is relatively small and can be analyzed manually.

The average time of the final transition firing is recorded as the simulation result. The analytical solution is simply the sum of all transitions' mean values, provided that $cdf(0) \approx 0$ for all transitions (which is not necessarily true for the normal distribution).

"Circular" Circular is a recurrent model, meaning that a marking may occur multiple times during a simulation run. As a consequence, the error made by simulating a transition will be repeated over and over again and may accumulate to influence the simulation accuracy far more than a single transition normally would.

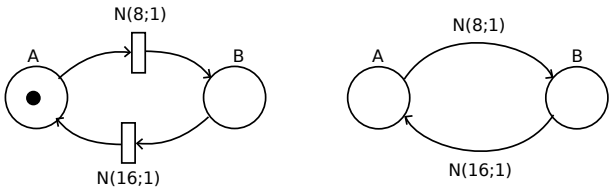


Figure 5.4: The "Circular" model, shown as a Petri net and reachability graph

Many real-life simulation models contain circular elements, usually to represent a subsystem that can be in either one of two states.

Since the probability mass moves back and forth between the two markings indefinitely, all performance measures determined by impulse rewards would increase with the simulation time and hence are not good candidates to determine simulation

accuracy. Instead, the model is simulated until the stationary solution is reached, i.e. until the flow of probability leaving each state per time step is the same as the probability mass entering that state and hence the probability to be in each state no longer changes over time.

For CTS simulation runs, the point in time at which the stationary solution is reached can be determined dynamically by comparing the set of proxels of consecutive time steps. If they are very similar, the stationary solution has been reached. For VTS simulations, however, this approach is not viable, because for the binary subdivision (cf. Section 3.2), the simulation end time has to be known in advance in order to establish a simulation time interval that can be subdivided.

Therefore, the point in time at which the stationary solution is reached is determined before starting the actual simulation runs by simulating the model with very small constant time steps and computing the change in probability to be in each state during each time step. The time at which the change in probability is acceptably small is then noted and doubled to be used as the simulation end time to allow less accurate simulation methods to reach their stationary solution. The probability to be in one of the two states is then recorded as the simulation result.

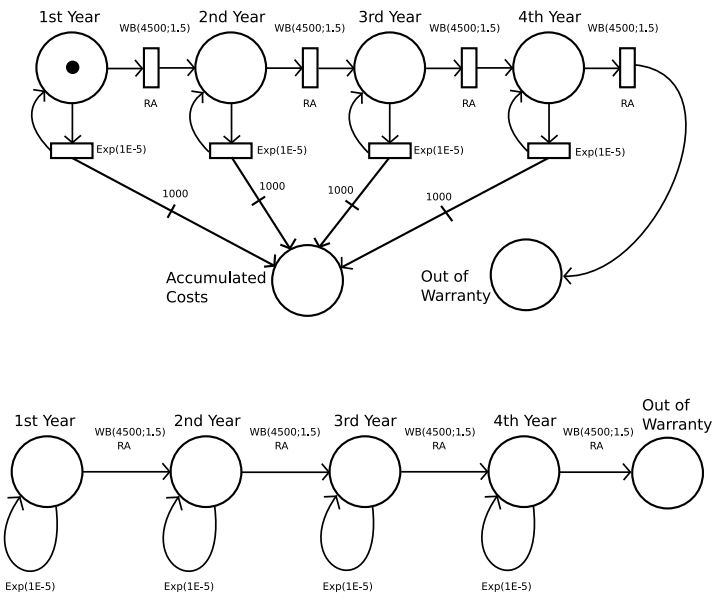


Figure 5.5: The "Warranty" model, displayed as a Petri net and reachability graph

"Warranty" The machine warranty model (cf. Figure 5.5) is a simulation model taken from [8]. The structure of the Petri net has been modified from the version in [8] for the sake of clarity. The state space of the model, and hence the simulation

results are, however, the same. The model is an actual simulation model from the Daimler AG[1]. It simulates the occurrences of damages (given by the exponential distributions) to a single piece of equipment over time and counts the cost incurred (1000$ per defect). The model simulates the lifetime of the item over the course of a warranty period of 4 years or 20.000km, whichever happens first. The simulation time unit is *km* instead of an actual time measurement. To reduce the state space size, the counting of the costs incurred has been removed from the model for the reachability graph, and has been established as impulse rewards. The simulation result measured is the average costs incurred during a warranty period.

This model was chosen for the experiments because it is actually used in the industry, and because it is a model where the CTS proxel algorithm already performs exceptionally well compared to the Monte-Carlo approach. Also, this model can be simulated with a wide range of time step sizes (multiple orders of magnitude) and still returns useful results[2].

"Machine Defect" The machine defect model (cf. Figure 5.6) was taken from the Ph.D. thesis of Claudia Krull [5] and the accompanying defense presentation. It simulates the defect of a machine in a factory in order to determine the average time to breakdown of the machine. The machine can only fail while people are working on it, i.e. during a shift. The factory is active for about 16 hours per day (two shifts) and is closed for about 8 hours, both normal-distributed with a standard deviation of 1h. The breakdown-interval of the machine is far longer (about 422 hours), and hence in the state "On Shift", two transitions with highly different firing times "compete" for the probability to leave the "On Shift" state. This property make the simulation model *stiff* and therefore difficult to simulate, meaning that it requires relatively long computation times to retrieve results with even modest accuracy.

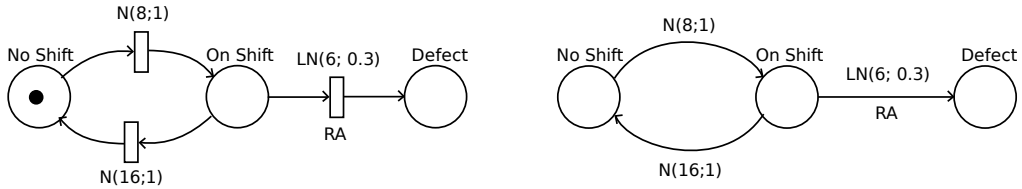


Figure 5.6: The Machine Defect" model, displayed as a Petri net and reachability graph

[1] At the time of writing of [8], the company was still called *DaimlerChrysler*

[2] The simulation time step size *could* be reduced almost indefinitely for all models, making the range of time step sizes practically unlimited. However, since computation time increases sharply with decreasing step size, there exists a practical lower bound to the step sizes.

5.3 Reduction of the Individual Errors

The proxel method exhibits at least three different sources of errors, which are all caused by the discretization of time. These errors are the integration (ODE) error, the error made through violation of the basic assumption (BA) of the proxel method (cf. 3.1) and the error through the non-deterministic behavior of the system being simulated as if it where deterministic (ND error, cf. 3.1).

There is no known way to reduce the BA error, but the ODE error can be reduced significantly by using higher-order integration schemes, and the ND error can be reduced by assuming the firing time of a transition to be in the middle of a time interval instead of at the end (cf. Section 3.1).

Reducing any one of these errors should increase the accuracy of the result when the step size is kept the same. An exception is the case where two errors have about the same magnitude but differ in sign. In this case, reducing only one of the errors can actually reduce the accuracy of the result.

Both ways of improvement are tested here on multiple simulation models. Since the size of the individual errors cannot be calculated and the errors may or may not cancel each other out, all integration methods are tested with and without adjusting the firing times.

The figures in the following paragraphs plot the simulation step size used against the absolute error of the simulation result computed with that step size. This is done for each integration method mentioned in Section 2.2. When applicable, the test has been conducted with the normal integration method as well as the variant modified to allow parallel integration (cf. Section 3.1) of the transition probabilities. In this case, the integration method names are suffixed with `_Separate` or `_Parallel`, respectively.

Leaving_N The "Leaving_n" model is a very simple model, containing only two reachable markings. Its only non-trivial element is the presence of multiple concurrently active transitions. Thus, it highly depends on an accurate computation of parallely active transitions. For this experiment, it was simulated with five active transitions with identical Weibull distributions ($\lambda = 45, k = 1.5$). The results are shown in the two diagrams of Figure 5.7.

With standard firing times, the influence of the integration methods is rather erratic. The Euler method ($O(n)$) causes the worst results, the Trapezoid methods ($O(n^2)$ for quadrature) the best ones. In between are all other integration methods. The parallel versions of ODE12, RK4 and ODE45 have almost exactly the same results, as do their separate counterparts. But the separate methods are more accurate than the parallel ones by about the factor 2.5.

With modified firing times (transitions fire in the middle of a time interval, cf. Section 3.1), the picture is very different: All parallel integration methods exhibit

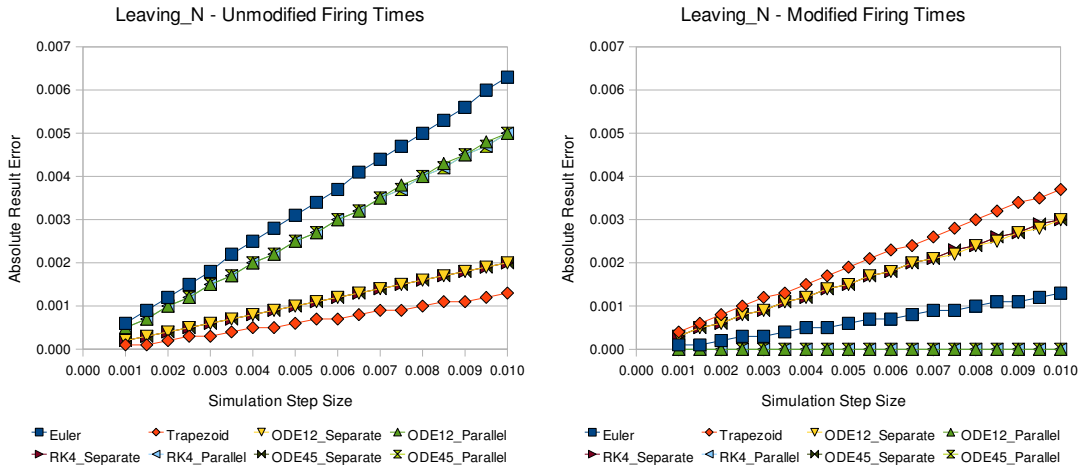


Figure 5.7: Discrepancy between the correct solution and the computed one for the "Leaving_n" model with and without modifying the firing time.

no measurable error at all for the step sizes chosen (i.e. the output of the simulation program is correct to at least six digits). The Euler method is second, the separate integration methods have all about the same results and are the third most accurate, while the trapezoid integration delivers the worst results.

Circular The "Circular" model is simulated to test the influence of cycles in a model's reachability graph to the simulation accuracy. It does not have any concurrently active transitions and so the parallel integration methods compute exactly the same results as the separate ones. Thus, the result diagrams (Figure 5.8) do not distinguish between the separate and parallel versions.

The result is very similar to that of the "Leaving_N" model. Without modifying the firing times, trapezoid integration performs best, Euler's method worst, and the three higher-level methods perform very similar to each other and are in-between. With modified firing times, the order stays the same with the one exception that the trapezoid method now performs worse than all other methods.

Overall, the result accuracy is improved dramatically and the best method for unmodified firing times (trapezoid) is less accurate than the best one for modified firing times (RK4 and ODE45) by a factor of ten.

Warranty The "Warranty" model is simulated to evaluate the effects of modified firing times and better integration methods on a small real-life model.

As with the other tests, the three parallel methods perform very similarly, as

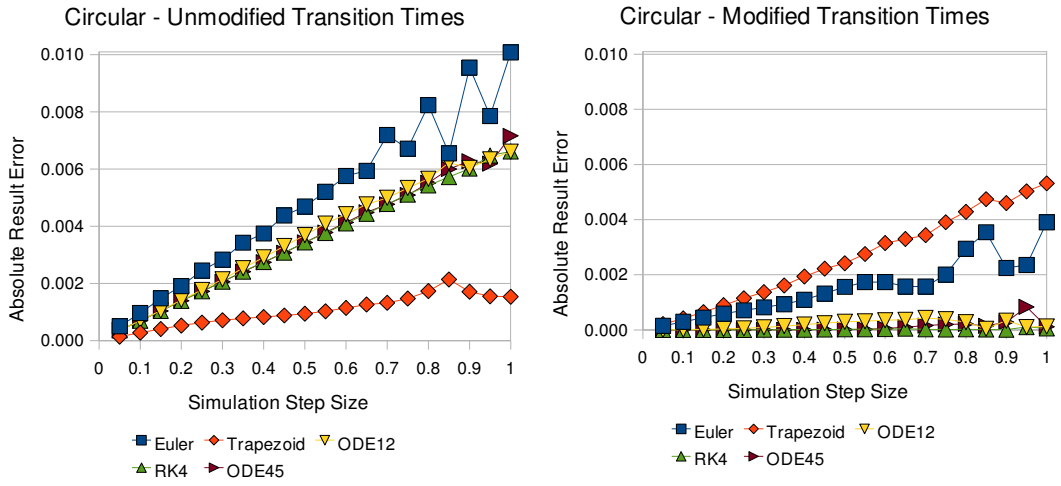


Figure 5.8: Discrepancy between the correct solution and the computed one for the "Circular" model with and without modifying the firing time.

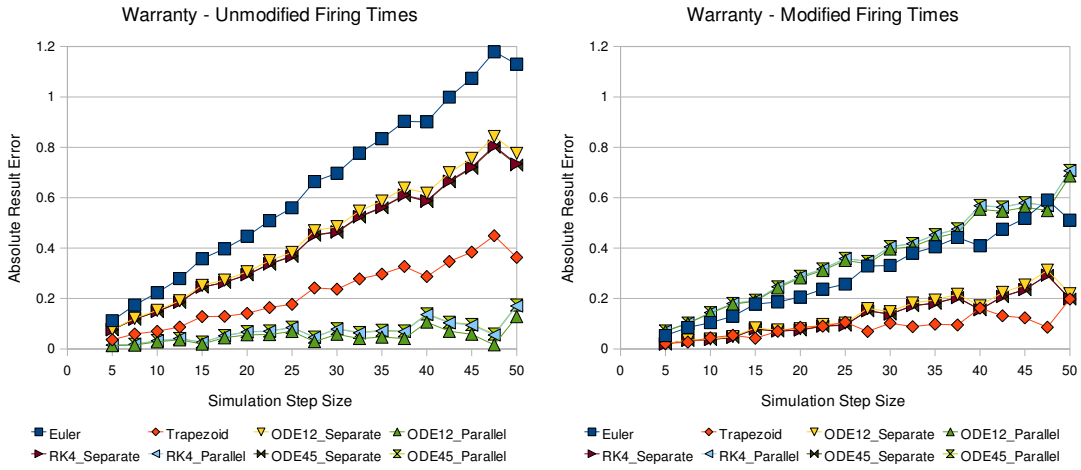


Figure 5.9: Discrepancy between the correct solution and the computed one for the "Warranty" model with and without modifying the firing time.

do the three separate methods. But other than for all other models, the parallel integration methods perform best without modifying the firing times, and worst with modification. The accuracy of all other integration methods is increased by modifying the firing times.

Still, the most accurate solution is the one obtained with RK4 and ODE45 with

unmodified firing times, being about twice as accurate at the best one computed with modifying the firing times.

Machine Defect The final model used is "Machine Defect". This model is rather stiff and it is therefore computationally expensive to get accurate solutions.

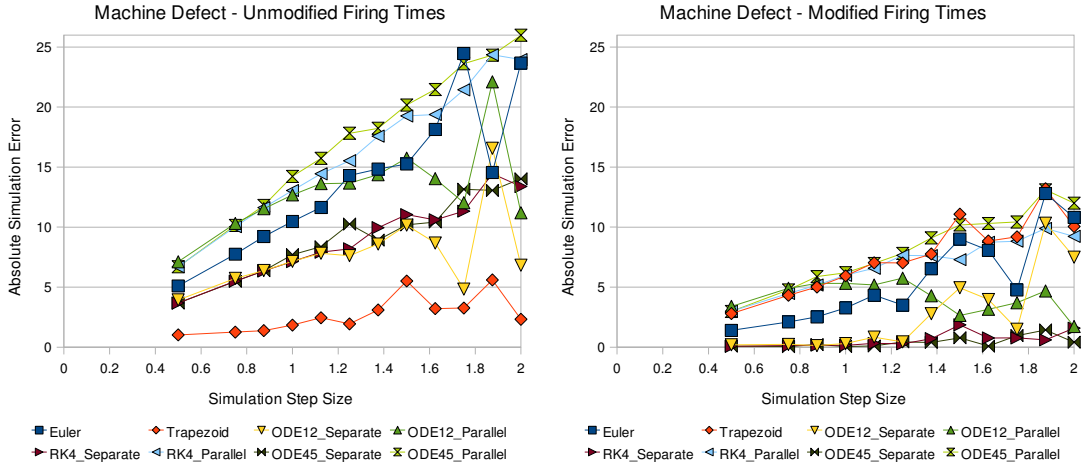


Figure 5.10: Discrepancy between the correct solution and the computed one for the "Machine Defect" model with and without modifying the firing time.

Overall, the results with modified firing times are more accurate (cf. Figure 5.10) for all integration methods but the trapezoid integration. For this model, the separate integration methods always perform better than the corresponding parallel ones. For small step sizes, the most accurate method using modified firing times is about ten times more accurate that the best one with traditional firing times.

Discussion of Results Increasing the simulation result accuracy by separately reducing the integration and ND errors is a promising approach with a flaw. The experiments have shown dramatic increases of accuracy for some combinations of integration methods and choice of traditional or modified firing times. However, the actual combination that computes the most accurate results is different for every model, so that no general recommendation on the use of integration methods and modification of the firing times can be given.

For the more complex models "Warranty" and "Machine Defect", the separate integration methods performed significantly better than the parallel ones. This is contrary to the predictions and no explanation could yet be found.

One behavior that could be observed for all models is that once the step sizes

are small enough so that the simulation results no longer change erratically when reducing the step size, then the differences in accuracy between the ODE12, RK4 and ODE45 methods becomes extremely small compared to their differences to the results of the other integration methods. Since ODE12 is the fastest method of the three, it is the method of choice if an advanced integration method is required.

Fortunately, the graphs in the result plots usually do not overlap for step sizes that are small enough for the results to not behave erratically and thus a method (combination of integration method and modification of the firing time) that is more accurate for a bigger step size is usually also more accurate for a smaller one. Hence, a suitable approach is to test all methods for comparatively large time steps, determine the best method for the model at hand, and then to computed the accurate simulation results with much smaller step sizes only with the most accurate method.

This approach can be very beneficial even though the initial simulations have to be conducted with up to 16 methods (8 integration methods, each with traditional and modified firing times) instead of just one. But these can be computed with a much higher step size and consequently take only a small fraction of the time necessary to calculate the final simulation result. Also, the most efficient method has been shown in the experiments to usually be two to 50 times more accurate than the traditional one (trapezoid integration, unmodified firing times). Since increasing the accuracy by a factor of two would otherwise require to simulate with half the step size and would therefore at least double the computation time, a substantial amount of time can be saved by choosing the right method.

Concluding, no single best combination of integration method and firing time could be identified in the experiments. But determining the optimal combination for a given model by testing all combinations with rather large step sizes is a viable strategy to save overall computation time.

In most tests, the results for the Heun method (ODE12) were comparable to those for RK4 and ODE45. Since Heun's method is less computationally expensive, it should be preferred over RK4 and ODE45.

5.4 Performance Increase Through AVL Trees

Most of the computation time of a proxel simulation run is spent on finding proxels representing the same system state so that these can be merged (cf. Section 2.3). Simply not merging these proxels would significantly reduce the computation time per proxel, but would also increase the rate of growth of the number of proxels and thereby actually increase the overall computation time. Therefore, it is of essence to find a suitable proxel container that allows for the efficient identification of these proxels with the same state S. In the reference implementation of the proxel method,

a generic binary search tree is used to efficiently store and compare the proxels. But since such a tree can easily degenerate depending on the order of proxel insertions, the best-case performance of $O(log(n))$ for finding a proxel in the tree cannot be guaranteed. Self-balancing trees like AVL trees have a slight overhead when adding elements, but the trees created are always well-balanced and guarantee a $O(log(n))$ time complexity for insertions and lookups.

Hence, using AVL trees as proxel containers should be much faster than using generic binary search trees. However, this assumption has been tested experimentally in [9] and has been refuted, as the generic binary tree was usually faster by a factor of two to three. Since this result conflicts with the theoretical predictions and no satisfactory explanation was given, it was chosen to repeat the experiments for this thesis.

Machine Defect

	Computation Time (s)		
Step Size	AVL Tree	Binary Tree	Factor
8	0.01	0.01	1.00x
4	0.1	0.1	1.00x
2	7.86	10.34	1.31x
1	55.3	96.04	1.73x

Warranty

	Computation Time (s)		
Step Size	AVL Tree	Binary Tree	Factor
78.13	0.32	0.94	2.96x
39.06	1.26	6.73	5.33x
19.53	5.23	52.49	10.0x
9.77	21.48	409.09	19.0x

Figure 5.11: The computation times for the "Warranty" and "Machine Defect" models using AVL trees and regular binary search trees as proxel containers.

To determine the speed increase, the "Warranty" and "Machine Defect" models are simulated with constant step sizes. For both, the regular binary search tree and the AVL tree, the number of proxels per simulation run and the actual simulation results were identical, indicating that both containers work correctly.

Figure 5.11 shows the simulation results: The AVL tree is never slower than the binary search tree, and can be significantly faster.

Discussion of Results Even though AVL trees have a slight computational overhead when adding new proxels, this is compensated by the speedup when searching for proxels with identical system states. During the experiments, this overhead never translated to higher overall computation times. Additionally, the more proxels have to be stored in a single container, the higher was the speed gain from using AVL trees. So it seems, that AVL trees are not only faster by a constant factor, but actually have a more favorable time complexity than regular binary search trees.

Thus, AVL trees are recommended to be used to store proxels whenever it is necessary to find and merge proxels. In cases where only inserting and iteration over all proxels is necessary, a dynamic array or a linear list may be more appropriate.

Unfortunately, the discrepancy between the results presented here and those in [9] cannot be explained. The original source code used for the older experiments is no longer available. One reason for the difference is surely that in [9] a unique hash was computed for each proxel and the proxels where added to the binary search tree based on that hash and not on their actual system state. Since the degeneracy of a binary search tree highly depends on the order in which the elements are inserted, and since applying a hash to elements usually randomizes their order, the regular binary trees in [9] could have been far more balanced than in the experiments conducted for this thesis. But even a perfectly randomized list of proxel would cause the binary tree to degenerate somewhat compared to a self-balancing AVL tree. Hence, randomization through hashes cannot completely explain the discrepancies.

Other possible explanations for the discrepancy include implementation bugs and testing errors, but these cannot be validated without access to the original source code.

Finally, it should be noted that the method of calculating unique hashes for each possible system state requires the state space to be finite and relatively small to eliminate the possibility of *hash collisions*. Most probability distributions have infinite support which is only made finite by inaccurate numerical integration. And when applying variable time steps, even distributions with limited support can have a huge set of possible age vectors, since the time steps and consequently the age differences can be almost arbitrarily small. Thus, the hashing approach is not directly applicable for variable time steps and AVL trees are the better choice for VTS.

5.5 Computational Overhead of the Variable Time Step Algorithm

An important issue is to determine the computational overhead of the VTS algorithm, i.e. how much more time the VTS algorithm needs if it conducts exactly the same computation as the CTS one.

The algorithm for variable time steps (cf. Section 3.4) is far more complex. It needs to maintain multiple proxel containers, requires proxels to store additional data and has to transfer probability mass to proxels existing at different points in simulation time. Additionally, most VTS criteria require at least the transition probabilities to be computed for all recursion steps even if the time step will be subdivided, while the CTS algorithm only computes these for the time steps that are actually simulated. The UNANIMOUS criterion additionally requires the computation of a BA violation estimate for all recursion steps.

Thus, the VTS algorithm is likely to need more computation time to carry out the same computations. This added complexity needs to be compensated by the increased efficiency of well-selected variable time steps. To determine the computa-

tional overhead of the VTS algorithm, the "Warranty" and "Machine Defect" models are simulated with a CONSTANT criterion for VTS. This criterion subdivides a time step if the step size is above a constant value, effectively allowing constant time steps to be used in the VTS algorithm. The computation times are then compared to those of the corresponding CTS simulation runs. Both algorithms use the separate ODE12 integration method for this test, since it is the least complex method that still computes integration error estimates, which are needed for some VTS criteria.

Machine Defect

	Time (s)		
Step Size	CTS	VTS	Factor
8	0.01	0.02	1.66x
4	0.1	0.26	2.70x
2	7.86	19.62	2.49x
1	55.3	122.94	2.22x

Warranty

	Time (s)		
Step Size	CTS	VTS	Factor
39.0625	1.26	2.54	2.01x
19.53125	5.23	10.5	2.00x
9.765625	21.48	43.86	2.04x
4.8828125	88.13	183.06	2.07x

Figure 5.12: The computation times for CTS and VTS CONSTANT algorithms with the same step sizes.

The simulation results for the CTS and VTS algorithms were identical each time, indicating that both algorithms are implemented correctly. Figure 5.12 shows the measured computation times. Time step sizes were chosen such that the computation time is high enough to be accurate, low enough to be completed in a reasonable amount of time, and fit the constraint for VTS time steps, i.e. $\Delta t = t_{end}/2^i$. For both models, the VTS algorithm needed about twice as much time as the CTS algorithm when doing the same computations and delivering the same results.

While this demonstrates that the VTS algorithm does indeed have a computational overhead over the VTS one perproxel simulated, it also shows that the overhead seems to be a constant factor and not dependent on the step size. This means that the time complexity required to find a proxel to which to redirect a transition probability is successfully hidden by adding the probability redirections directly to the proxels and does not increase the overall time complexity (although it does increase the overall computation time).

5.6 Quality of the Subdivision Criteria

In Section 3.5, multiple subdivision criteria were developed. As no single criterion was clearly superior, it was decided to implement and test all five of them and to compare their simulation results to each other as well as to the CTS results.

All criteria have a single parameter k that can be tuned to adjust the simulation run for speed or accuracy. But these k have different meanings for each criterion, so comparing the simulation results computed by using the same k for each criterion would be meaningless.

Instead, a meaningful comparison can be conducted by assessing the simulation efficiency, i.e. the accuracy of the simulation result with respect to the computation time. This is done by simulating multiple models with all VTS criteria and with the CTS algorithm as a baseline to compare the VTS results against. All combinations of models and criteria are simulated with various settings for the respective k (the k for the CTS algorithm being the step size). The values of k are chosen such that the simulation accuracy is high enough to compute non-chaotic results and such that the computation time is not too high, depending on the model no higher than 30-3000 seconds for a single simulation run.

To determine the simulation accuracy, the exact simulation result must be known. This can be computed analytically for some of the models. For the other ones, an approximation of the analytical result was calculated by conducting expensive simulations with the CTS algorithm. Since this result x' is likely to be closer to the analytical solution x than all test run results x_i, the absolute error of the simulation result $|x - x_i|$ can be reasonably approximated by $|x' - x_i|$.

To assess the quality of the criteria, the absolute error of each simulation run is plotted against the computation time for that run, and the diagram points of each criterion are connected. The resulting graphs are usually monotonously decreasing, meaning that with increasing computation time, the simulation result accuracy increases. If the complete graph of a criterion A lies below the graph of a criterion B, then criterion A is the more efficient one, since it requires less computational effort to achieve a level of accuracy. If the graphs of two criteria overlap, each is more efficient than the other one in some areas and less efficient in others.

Test Setup Since no clearly superior integration method or treatment of the firing times could be found during the experiments in Section 5.3, all simulation runs are performed with the well-known options of the standard proxel algorithm:

- trapezoid integration
- initialization of age vector elements with zero after a transition fires (as opposed to initializing it with $\frac{1}{2}\Delta t$)
- omission of proxels with a probability mass of less than 10^{-15}

The only exception is the integration method for the UNANIMOUS criterion. It must obviously include an integration error estimate for the criterion to work correctly. Only ODE12 and ODE45 compute these estimates. For the computation of

the transition probabilities themselves, ODE12 is usually sufficient, and it is much faster than ODE45. But its error estimate depends on Euler's method and therefore can be too unreliable to base a subdivision decision on it. In these cases, ODE45 is preferable. Thus, since both integration methods have their specific advantages, the UNANIMOUS criterion will be tested with both methods.

Additionally, the integration method for UNANIMOUS must be a parallel one (cf. Section 3.1), because the separate integration methods only compute the correct integration error estimates for the cases of independent ordinary differential equations, i.e. for the cases where only a single transition is active. Using them would often calculate too small integration error estimates and hence would not cause a time step subdivision when one is necessary.

Chain Model The chain model (cf. Figure 5.3) is simulated with five transitions distributed with $N(1, 0.25)$, $N(4, 0.5)$, $N(9, 0.75)$, $N(16, 1)$ and $N(25, 1.25)$, respectively. The vastly differing rates and non-uniform behavior of the transitions test the ability of the criteria to adopt to different situations.

The simulation result recorded is the average time at which the fifth transition fires. Its analytical solution is the sum of the mean values of all five distributions. Figure 5.13 gives the simulation results for all successful criteria, i.e. all criteria for which the simulation runs finished and computed a sensible result. The criteria GLOBAL_PROB, UNANIMOUS_ODE12 and UNANIMOUS_ODE45 are omitted from the diagram, because their simulation results did not converge to the real value independent of the computation time. The graph for the BASE_PROB criterion has actually a similar shape as those of the other criteria, but since it is up to 100 times more accurate than the CTS algorithm, it looks like a straight line.

For the "Chain" model, the BASE_PROB criterion clearly performs best and is the only VTS criterion to perform better than the CTS algorithm. The MEAN criterion's performance is close to than of CTS, TRANS_PROB requires many times more computation time to achieve the same accuracy.

Circular In the circular model (cf. Figure 5.4), each transition fires numerous times, and thus the simulation errors can easily build up. This test is significant in assessing the quality of the VTS criteria, because most simulation models contain circular elements. A well-designed VTS criterion should still be able to limit the overall error.

For this test, the two transitions are set to fire according to a $N(8, 1)$ and a $N(16, 1)$ distribution. The result measured is the probability to be in the first state in the stationary solution. Its analytical solution should be $\frac{2}{3}$, based on the ratio of the two distributions' mean values.

Figure 5.14 shows the simulation results. The results for the GLOBAL_PROB

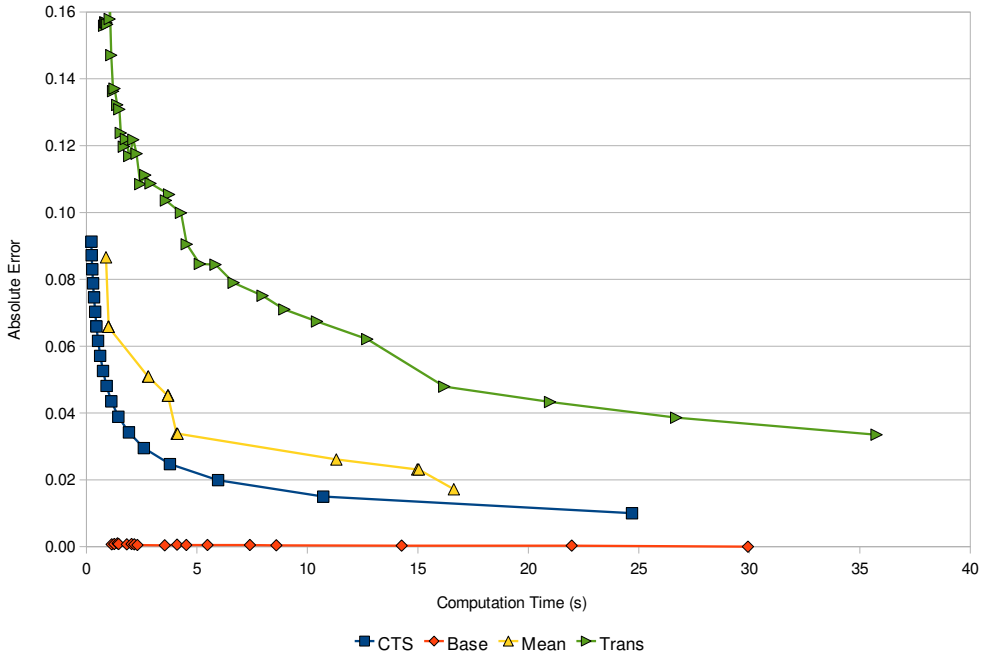


Figure 5.13: Simulation results for the "Chain" model

criterion did not converge to the actual result and are therefore omitted. The BASE_PROB results behave somewhat erratic for very short simulation runs, but converge when the simulation takes longer than 3 seconds and are then more accurate than the CTS results for the same computation time.

The results for TRANS_PROB show the same behavior, but converge far later and far slower. Also, it was not possible to retrieve results for TRANS_PROB in under 6 seconds. The maximum threshold for TRANS_PROB is $k = 1$, for which the simulation is completed in a single time step with nonsensical results. TRANS_PROB computes useful results for value below one, but even for $k = 0.99999999$, the computation time is about 6 seconds.

There are very few simulation results for the MEAN criterion, because in situations where the rates of all transitions differ only by powers of two, the value of k must be divided by two to cause a different subdivision of time. The result for MEAN with the next smaller k did further increase the accuracy, but took 70 seconds to compute and is consequently not displayed in the diagram.

The result for UNANIMOUS-ODE45 seems to converge to the actual result very slowly, the result for UNANIMOUS-ODE12 apparently converges to a totally different value.

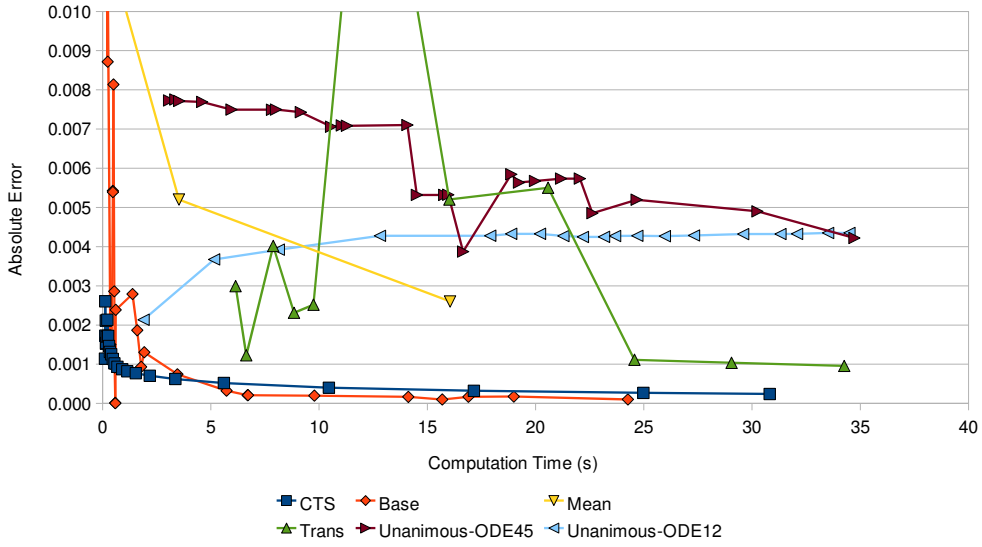


Figure 5.14: Simulation results for the "Circular" model

Warranty The warranty model (cf. Figure 5.5) is a real-life model that is actually in use in the industry. Thus, the results of this experiment have direct practical relevance. Yet, the model is a very simple one, since all Weibull-distributed transitions fire only once and the exponentially distributed transitions fire only a few times owing to their low rates.

The results for the "Warranty" model are show in Figure 5.15. The results for the GLOBAL_PROB criterion converged to a wholly different value, and hence are useless and consequently omitted. The BASE_PROB criterion performs best in this experiment as well, but strangely does not increase the result accuracy when the computation time increases. MEAN also performs better than CTS, but inexplicably computes two simulation results with different accuracies in the same computation time of about 86 seconds. TRANS_PROB and UNANIMOUS-ODE45 perform about as well as CTS, while UNANIMOUS-ODE12 is slightly worse.

Machine Defect The "Machine Defect" model (cf. Figure 5.6) resembles the "Circular" model, but this one has a stationary solution. Because it has multiple active transitions with different firing times, it is stiff and thus needs far more computational effort to compute accurate results than is necessary for "Circular".

The simulation results (cf. Figure 5.16) are somewhat unexpected. The result for GLOBAL_PROB are off by a factor of four and not even close to the correct result for all $k > 10^{-6}$. For $k = 10^{-6}$ the simulation run consumed more than one

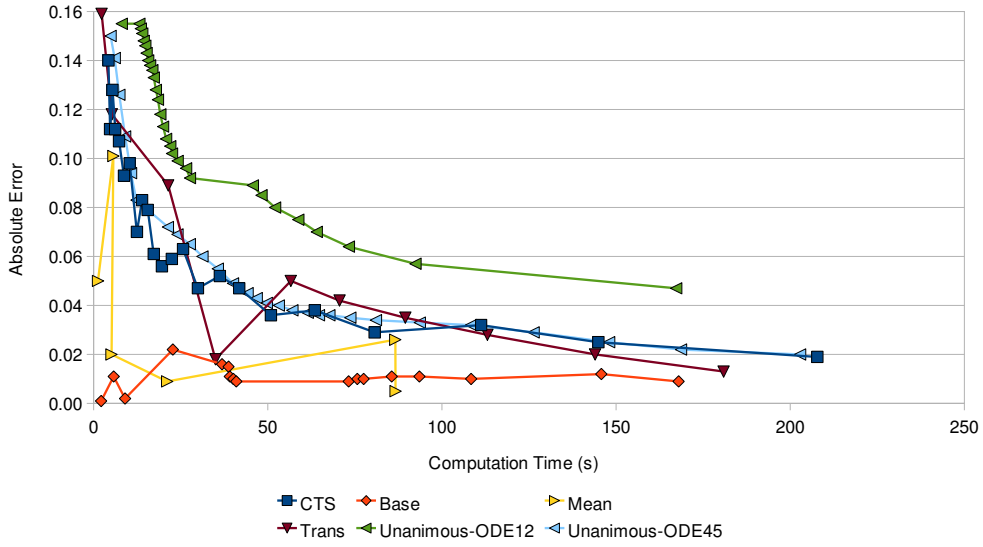


Figure 5.15: Simulation results for the "Warranty" model

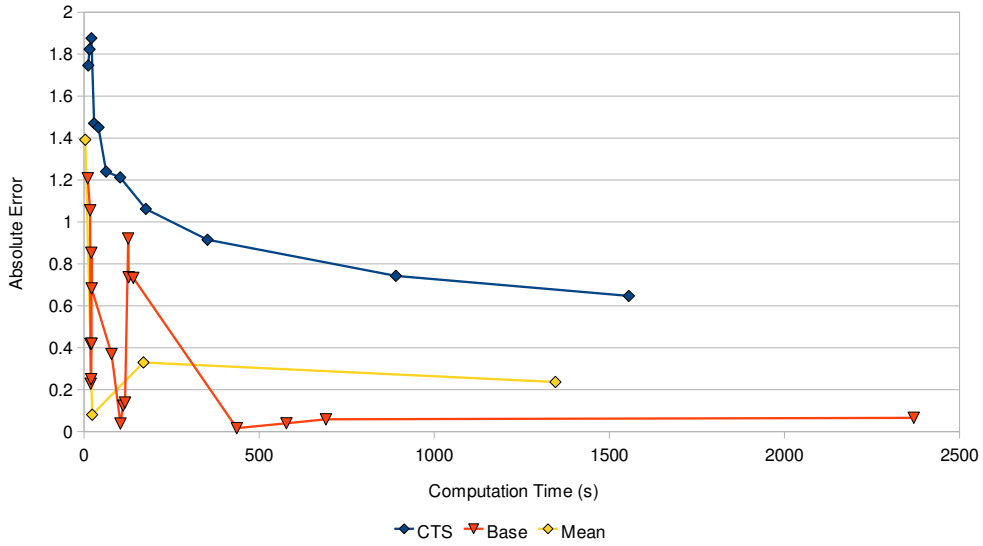


Figure 5.16: Simulation results for the "Machine Defect" model

gigabyte of RAM and had to be aborted. UNANIMOUS-ODE12 descended into an infinite recursion for all tested values of k and consequently crashed. TRANS_PROB

finished for some parameter values, but deadlocked in an infinite recursion for others, including all tested simulation runs for $k < 0.87$. UNANIMOUS-ODE45 did finish in all runs, but the results for UNANIMOUS-ODE45 and TRANS_PROB were still less accurate than CTS by a factor of ten to 50.

So only the MEAN and BASE_PROB criteria computed acceptable results. Both were more accurate than CTS, with BASE_PROB usually having the higher accuracy of the two.

Discussion of Results

Limiting the Target Probability (GLOBAL_PROB) GLOBAL_PROB was created as a simple and yet effective criterion, but in the experiments, it rarely computed meaningful results. The reason is that it has a theoretical flaw that renders it useless: In proxel simulations, successor proxels always have a probability mass that is less than or equal to that of the parent proxel (the only exception being proxels that are the result of merging). So once a proxel's probability mass is below the threshold selected for the simulation run, its successor's probability will always be below that threshold as well, no matter how big the next time step is. Consequently, the next time step size selected by the criterion will be as big as permitted by the binary subdivision - potentially reaching the end of the simulation in a single simulation step. This behavior causes a huge error and renders the simulation results useless. For most simulation models, the results using GLOBAL_PROB do not converge to the actual simulation result, no matter how small k and how high the computational effort is.

Thus, GLOBAL_PROB is not a viable VTS criterion.

Limiting the Transition Probability (TRANS_PROB) TRANS_PROB limits the fraction of the remaining probability that is going to leave the current marking during a time step. Its results were always less accurate than those of the CTS algorithm and it did crash in some situations. This is the result of two grave problems of the approach.

First, since only a certain fraction of the remaining probability mass will leave a marking during a time step, the remaining probability mass in the state after n time steps will have the shape of an exponential decay[3]. Hence, the largest absolute probability will always leave the state in the first time step. For example, for a TRANS_PROB threshold of $k = 0.5$, half of the probability will leave the state in the first time step and the remaining half will be simulated with potentially

[3]This is only to say, that the remaining probability plotted against the *number of time steps* passed since entering that state follows the exponential decay function. Since the size of the time steps will vary, the remaining probability plotted against the time *passed* will follow the actual probability distribution of that transition.

thousands of time steps. So simulating the transition to create the first proxel will cause about the same error as simulating all the other thousands of transition steps for that transition, clearly a miss-distribution of computational effort which explains that the TRANS_PROB results where always less accurate than the CTS ones.

The second problem is even more severe. It stems from the fact that for transitions with limited support, their hazard rate function will be infinite at the end of the support interval and hence all remaining probability will leave the state at once. The uniform distribution naturally has limited support and normal distributions tend to have numerically limited support, because the numerical approximation of their hazard rate function will go up to infinity as well[4]. Analytically, all probability should have left the marking at the time that the hrf becomes infinite. But since the proxel method numerically approximates the transitional behavior, some probability can remain in the marking up to that point. In this case, all probability will leave the state (i.e. the transition probability is one) in the next time step, no matter how small the time step is. But at the same time, TRANS_PROB will try to subdivide all time steps were the transition probability is larger than k. This conflict causes an infinite recursion and results in the simulation application to crash.

These two problems are the reason for TRANS_PROB to not be a useful criterion.

Limiting the Transition Probability Fraction (BASE_PROB) BASE_PROB performed well in all tested situations. It always terminated and returned meaningful results. In all tested situations and models, BASE_PROB delivered more accurate results than the CTS algorithm with the same computational effort.

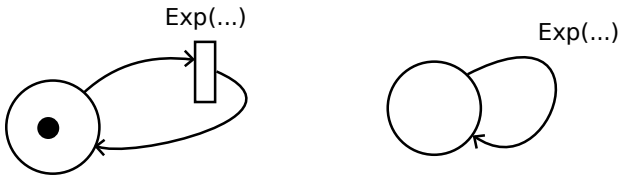


Figure 5.17: Petri net and reachability graph for the smallest model that causes problems with BASE_PROB

It should be noted, however, that BASE_PROB has a small theoretical flaw for a minor class of simulation models: In models where an exponential transition connects a marking with itself (i.e. the marking before the transition fires is the same as after it fires, cf. Figure 5.17), the proxel representing the case that the transition just fired has exactly the same system state as the proxel representing inactivity. Therefore, both proxels will be merged and their base proabilities will be added, meaning that

[4]The *limit* of the hazard rate function is infinity for almost all distributions, but for the two distribution types mentioned, the hrf actually *reaches* infinity at a certain point.

the base probability for that state increases and can increase to even become bigger than one. This behavior is obviously inconsistent and can lead to badly-chosen time step sizes. Fortunately, very few models fall into this problematic class.

Limiting the Error per Time Step (UNANIMOUS) The basic idea behind the UNANIMOUS criterion was to directly estimate the simulation error per time step and subdivide time steps to limit the overall error. While the approach itself may be worth to be developed further, the current implementation has some shortcomings which renders it useless.

First, no suitable error estimate for the ND error could be found. By simply ignoring it, the simulation error per time step can be huge when the BA and integration errors are small and the ND error is big, but is not noticed. This can lead the simulation result to converge to a completely different value than the actual solution.

Second, the computation of the error estimates is very time-consuming. So even if a UNANIMOUS simulation run could achieve a certain level of accuracy using far fewer proxels than other criteria, the added computational overhead of the error estimates can still cause it to be more time-consuming and thus less efficient.

Third, all error estimates are just that, estimates. If the error estimates are smaller than the actual error, a big simulation error can be caused. If the estimates are higher than the actual error, the simulation is conducted with more time steps then necessary, reducing efficiency.

An integration error estimate can even be as bad as to never go below a certain threshold, no matter how small a time step is. Among others, this can happen at the end of a distribution's support and will lead to an infinite subdivision of time and thus an infinite recursion.

This problem is especially prominent when using ODE12 integration where the quality of the error estimates is very low.

Following the Transition Means (MEAN) The MEAN criterion is a very reliable one. It completed all tests and always computed meaningful results. It performed better than CTS in some tests and worse in others. This is likely to be the result of its static nature, which makes it impossible to adapt to varying situations.

Also, since MEAN maintains a constant step size for each transition, and since the possible step sizes all differ by powers of two, there are often very few values of k that cause different simulation results. The most obvious example is a model with two otherwise identical transitions whose rates differ by a factor of $\sqrt{2}$. Hence, these transitions should also be simulated with time steps that differ by that same factor. However, due to the limitation on the time step sizes, the two transitions will either be simulated with the same step size, or with step sizes that differ by a factor of 2.

In any case, one of the transitions will be simulated with $\sqrt{2}$ more time steps than necessary.

Conclusion Of the subdivision criteria tested, BASE_PROB is clearly the most competitive. It is robust and outperformed the CTS algorithm in every test. It is therefore recommended to be used in productive simulation environments that implement a VTS proxel approach. MEAN is the only other criterion that performed at least almost as well as CTS and its results are more predictable than those of BASE_PROB.

5.7 Applicability of the Richardson Extrapolation

As explained in Section 3.6, the applicability of the Richardson extrapolation is an important property of the CTS proxel algorithm. Applying Richardson's method allows for the computation of more exact simulation results without having to reduce the time step size. And the convergence of the simulation results when reducing the simulation step size - a prerequisite for the application of Richardson's method - allows for the estimation of the simulation result accuracy. Thus, it is important to assess the applicability of this method to the VTS algorithm.

Since a formal proof of linear convergence (or rather of the VTS proxel algorithm to exhibit a first-order error depending on the criterion threshold k) could not be obtained in Section 3.6, empirical evidence will be collected in this section to determine the applicability of Richardson's method.

The threshold k has a different meaning for every criterion developed. Consequently, convergence would have to be shown for all criteria separately. However, only the MEAN and BASE_PROB criteria delivered useful results in all experiments in Section 5.6. Thus, the convergence only needs to be investigated for these two criteria. All other criteria already showed serious flaw that would not be compensated by successfully showing the convergence.

BASE_PROB To test for linear convergence, the models "Chain", "Circular", "Warranty" and "Machine Defect" were simulated with the BASE_PROB criterion with a wide range of values for the threshold k (about 1000 per model in the interval $[0, 1]$).

The results are shown in Figure 5.18. Since the details of the graphs for small values of k are hard to recognize, the graphs in the right column show a magnification of this area of the one on the left. Consequently, the graphs on the right have a different scale from the left ones for both axes.

The results are somewhat surprising. As expected, the simulation results do indeed converge to the analytical solution of the model. However, they converged chaotically rather than linearly.

For "Chain" and "Circular", the simulation result first approaches the correct value from above for decreasing values of k, but do not do so linearly, but rather by alternating intervals of no change with sudden jumps. Once the values come close to the correct one, they alternate chaotically around the it. With further decreasing k, the amplitude of that alternation decreases, but the behavior itself stays chaotic.

For the "Warranty" model, the simulation result approaches the correct value almost linearly for decreasing values of k, but this linear descent is interrupted by sudden jumps in the result. For values of k that are smaller than 0.005 the linear descent and the results seem to change chaotically.

The results for "Machine Defect" show alternating intervals of smooth descent, sudden jumps and intervals of constant results for varying k. For $k < 0.2$, the results behave chaotically just as for the other models.

Figure 5.20 shows the results of the Richardson extrapolation for various pairs of values for k. These values for A and B are chosen to differ by a factor of two, an approach that somewhat mitigates the effects of single outliers. Still, the results are devastating: Only in very few cases were the extrapolated results more accurate than both initial ones. Often, the extrapolated result was even less accurate than both simulated ones.

MEAN The special property of MEAN to be a static criterion in conjunction with the binary subdivision of time makes it tricky to compute meaningful result graphs, because the simulation results do not necessarily change when the value of k is varied.

A simple example illustrating this effect is a single uniform transition U(0, 1) that is simulated in the time interval [0,1]. For a k of one, the transition is simulated with a single time step. For a k even slightly smaller than one, the transition needs to be simulated with two time steps of size 0.5 each. The value of k can be chosen arbitrarily from the interval $[0.5, 1[$ without changing the time step sizes and hence the simulation result. The same effect occurs for all transitions independent of the chosen distribution and distribution parameters. For models with multiple transitions, the simulation results may change when the simulation step size of any transition changes.

Fortunately, the k for which the simulation result changes can easily be computed. Thus, to retrieve all possible simulation results, one only has to simulate the model with ks that are slightly above and slightly below these points of step size change. For all other values of k, the simulation result will be exactly same as for the two adjacent points of change.

Figure 5.19 shows the result graphs for these values of k. For "Chain" and "Circular", the results converge monotonously to the correct value, albeit not uniformly, but with alternating jumps and intervals of constant results. The set of all data

points before a jump, however, *do* lie almost on a straight line, as do all data points of simulation results after such a jump.

For the "Warranty" model, the results oscillate around the correct value with a decreasing amplitude for decreasing k. Thus, there is no linear relationship between the results for arbitrary k. For those k that differ by powers of two, however, a linear relationship does exists.

For the "Machine Defect" model, finally, no predictable behavior for the simulation result in relation to the values of k seems to exist. For large k, the result seems to decrease with decreasing k. For $k \leq 0.2$, however, the results seem to oscillate, but this observation is little more than a guess, since too few results exist to support or refute this theory. It was not possible to obtain more accurate results, since the computation of the next more accurate result would already have taken about 30 hours. It should be noted that for the "Machine Defect" model, some simulation results are identical, even though some of their transitions have different simulation step sizes. This happens for simulation results where only the lognormal transitions differ in simulation step size. Since the simulation result is measured by impulse rewards at that transition, and the rewards are triggered whenever probability is transferred to proxels in that state, the simulation result is independent of the actual simulation step size of that transition.

The observations of the convergence behavior are reflected in the numerical attempts at Richardson extrapolation as well (cf. Figure 5.21). For the "Chain", "Circular" and "Warranty" models, the extrapolated results are better than both initial ones for all k that are smaller than a certain k_0. They are often more accurate by an order of magnitude. For the "Machine Defect" results, however, the extrapolated results are sometimes more accurate and sometimes less accurate than the direct simulation results and no clear pattern of that behavior is visible.

Discussion of Results

In section 5.7, the Richardson extrapolation has been shown to not be generally applicable to the simulation results of BASE_PROB and MEAN, the two subdivision criteria with otherwise acceptable results for the simulation efficiency.

BASE_PROB For BASE_PROB, the extrapolation results were usually worse than the actual simulation results that were used for the extrapolation. This is likely to be not a direct result of the criterion itself, but of the interaction of the criterion with the binary subdivision algorithm. Since the binary subdivision does not allow arbitrary step sizes, parts of transitions may be simulated with step sizes that differ by a factor of two for different simulation runs, even if their values for k are barely different.

Regardless of that explanation, the BASE_PROB criterion should be used very carefully to simulate stochastic models in productive environments, since through the lack of linear convergence, the accuracy of the simulation results cannot be assessed reliably.

MEAN The Richardson extrapolation cannot directly be applied to arbitrary simulation results obtained with the MEAN criterion. As an effect of the binary subdivision, the transitions only change their step sizes when the value of k passes certain values. Thus the simulation results only change if the value of k passes such a point. Applying the Richardson extrapolation to two results obtained for values of k just before and after such a point would result in an extrapolation with a large positive or negative value[5] that bears no resemblance to the actual simulation result. To a lesser degree, this also applies to other arbitrarily chosen ks. Since the MEAN results do not vary for big intervals of k, extrapolation of two values inside such an interval would yield the same result as both simulation results from that interval. And choosing values of k from different constant result intervals returns varying extrapolation results, depending on the actual values of k that were used for the extrapolation (cf. Figure 5.22). This means that the extrapolation can yield arbitrary results depending on which values for k are chosen. So the Richardson extrapolation is not applicable to the MEAN results in the general case.

Fortunately, no matter what step sizes a model's transitions are simulated with for a given k, if the model is simulated with $\frac{1}{2}k$, the step sizes of all transitions are cut in half as well. Hence, the simulation results when reducing k to $\frac{1}{2}$ of its former value for each new simulation run are usually linearly convergent. Why this is not always the case for the results of the "Machine Defect" model is still unclear. Possible explanations are that the values of k for which the model was simulated were still too big to exhibit linear convergence behavior, or that the linear convergence does not occur for more complex models. Unfortunately, simulating the "Machine Defect" model with even smaller values of k that those used for the graphs in Figure 5.19 is not computationally feasible. If the value of k for the "Machine Defect" model with the MEAN criterion is cut in half, the computation time increases by about the factor ten, and thus the computation for the next smaller value of k that would show a different simulation result would take about 30 hours.

[5] The reason is that the Richardson extrapolation for first-order errors is just a linear extrapolation. Since the two values vastly differ by their results $f(k)$, and are almost identical in their k values, the extrapolation slope $\frac{\Delta f(k)}{\Delta k}$ is huge and the extrapolation for $f(0)$ differs from the actual value by a large value.

5.8 Summary

The "Experiments" chapter presented experiments and experimentation results suitable to answer the questions posed in Section 5.1, including those relevant to answer the questions posed as the goal of this thesis in Section 1.3.

The experiments about reducing individual errors showed that reducing some errors can positively impact the simulation result accuracy, even though the parameter combination necessary to achieve the accuracy increase can vary. Using AVL trees as the primary proxel container, on the other hand, has shown to be beneficial in every case tested.

The current VTS implementation has been measured to be about half as fast as the CTS implementation for the same time step sizes. Nevertheless, some VTS subdivision criteria managed to compute more accurate results in less time than the CTS approach. Most notably, the BASE_PROB criterion was superior to the CTS implementation in every single test, and the efficiency of MEAN was still comparable to CTS. All other criteria failed at least one of the tests by either crashing the simulation program or computing nonsensical results.

Finally, the experiments concerning the applicability of Richardson's method of extrapolation showed that this method is not applicable to simulation results computed with the BASE_PROB criterion, and applicable only within limits to the MEAN results.

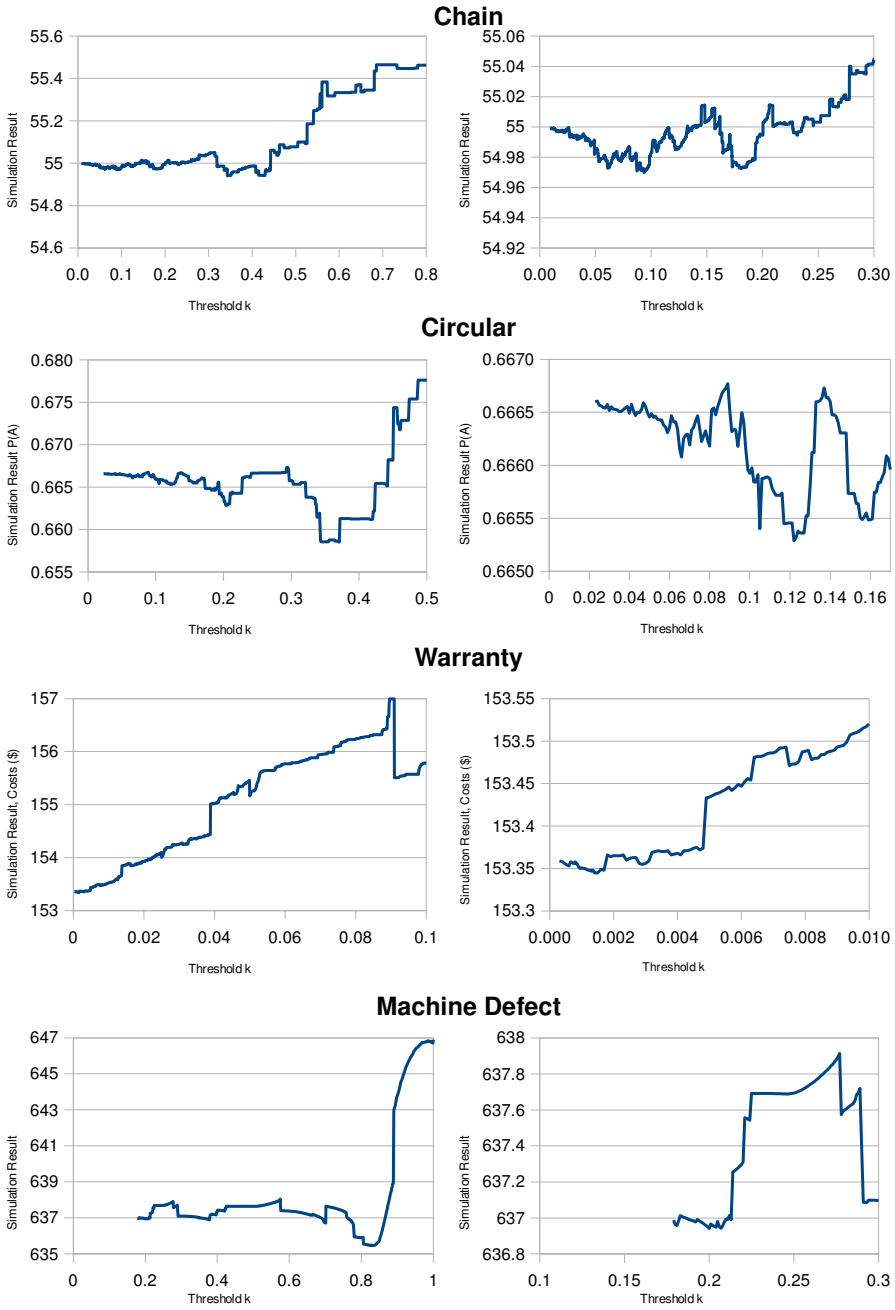


Figure 5.18: Plots of the BASE_PROB results for four models. The plots on the right-hand side show the details of the leftmost part of the corresponding plots on the left side.

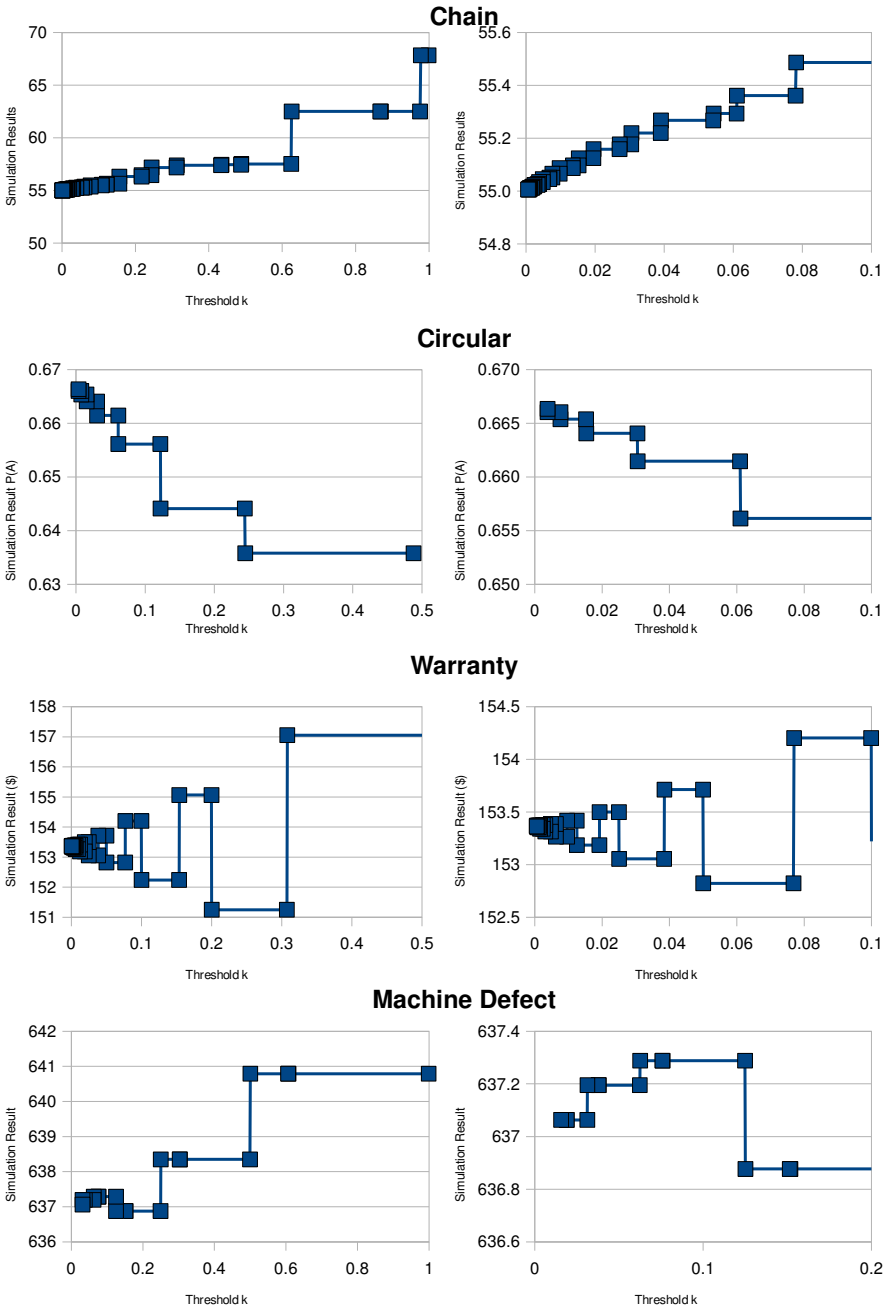


Figure 5.19: Plots of the MEAN results for four models. The plots on the right-hand side show the details of the leftmost part of the corresponding plots on the left side.

Chain

A	B	f(A)	f(B)	Richardson	Err(f(A))	Err(f(B))	Err(R)
0.5000	0.2500	55.0775	55.0032	54.9289	0.0775	**0.0032**	0.0711
0.2500	0.1250	55.0032	54.9862	54.9691	**0.0032**	0.0138	0.0309
0.1250	0.0625	54.9862	54.9733	54.9605	**0.0138**	0.0267	0.0395
0.0625	0.0321	54.9733	54.9934	55.0135	0.0267	**0.0066**	0.0135
0.0321	0.0160	54.9934	54.9974	55.0015	0.0066	0.0026	**0.0015**

Circular

A	B	f(A)	f(B)	Richardson	Err(f(A))	Err(f(B))	Err(R)
0.800	0.400	0.66328	0.66124	0.65921	**0.00339**	0.00542	0.00746
0.400	0.200	0.66124	0.66388	0.66651	0.00542	0.00279	**0.00016**
0.200	0.100	0.66388	0.66592	0.66797	0.00279	**0.00074**	0.00131
0.100	0.050	0.66592	0.66646	0.66699	0.00074	**0.00021**	0.00032
0.050	0.025	0.66646	0.66657	0.66667	0.00021	0.00010	**0.00001**

Warranty

A	B	f(A)	f(B)	Richardson	Err(f(A))	Err(f(B))	Err(R)
0.064	0.032	155.8	154.26	152.717	2.434	0.892	**0.650**
0.032	0.016	154.26	153.88	153.509	0.892	0.517	**0.142**
0.016	0.008	153.88	153.49	153.092	0.517	**0.121**	0.275
0.008	0.004	153.49	153.37	153.248	0.121	**0.001**	0.119
0.004	0.002	153.37	153.37	153.362	**0.001**	0.002	0.005
0.002	0.001	153.37	153.35	153.337	**0.002**	0.016	0.030

Machine Defect

A	B	f(A)	f(B)	Richardson	Err(f(A))	Err(f(B))	Err(R)
0.8	0.4	635.9	637.33	638.753	1.093	**0.332**	1.757
0.4	0.2	637.33	636.94	636.554	0.332	**0.055**	0.442

Figure 5.20: Numerical results for various attempts at Richardson extrapolation for the four models simulated with BASE_PROB. The entries with the lowest error are marked in bold. A and B are the two different values chosen for k. The symbol $f(X)$ represents the result for the threshold X and $Err(X)$ is the error of the result X. $Err(R)$, finally, is the error of the Richardson extrapolation.

Chain

A	B	f(A)	f(B)	Richardson	err(A)	err(B)	err(R)
0.978	0.489	67.834	**57.514**	47.194	12.834	**2.514**	7.806
0.489	0.244	57.514	57.180	**56.845**	2.514	2.180	**1.845**
0.244	0.122	57.180	**55.641**	54.103	2.180	**0.641**	0.897
0.122	0.061	55.641	55.361	**55.081**	0.641	0.361	**0.081**
0.061	0.031	55.361	55.220	**55.078**	0.361	0.220	**0.078**
0.031	0.015	55.220	55.124	**55.029**	0.220	0.124	**0.029**
0.015	0.008	55.124	55.066	**55.007**	0.124	0.066	**0.007**

Circular

A	B	f(A)	f(B)	Richardson	err(A)	err(B)	err(R)
0.488	0.244	0.63580	0.64411	**0.65242**	0.03086	0.02256	**0.01425**
0.244	0.122	0.64411	0.65614	**0.66816**	0.02256	0.01053	**0.00149**
0.122	0.061	0.65614	0.66147	**0.66680**	0.01053	0.00520	**0.00013**
0.061	0.030	0.66147	0.66407	**0.66667**	0.00520	0.00260	**0.00000**
0.030	0.015	0.66407	0.66538	**0.66670**	0.00260	0.00128	**0.00003**
0.015	0.008	0.66538	0.66603	**0.66668**	0.00128	0.00063	**0.00002**
0.008	0.004	0.66603	0.66635	**0.66668**	0.00063	0.00031	**0.00001**

Warranty

A	B	f(A)	f(B)	Richardson	err(A)	err(B)	err(R)
0.61	0.31	157.06	**151.25**	145.447	3.688	**2.116**	7.920
0.31	0.15	151.25	152.24	**153.227**	2.116	1.128	**0.140**
0.15	0.077	152.24	152.82	**153.407**	1.128	0.544	**0.040**
0.077	0.038	152.82	153.06	**153.287**	0.544	0.312	**0.080**
0.038	0.019	153.06	153.19	**153.317**	0.312	0.181	**0.050**
0.019	0.010	153.19	153.27	**153.346**	0.181	0.101	**0.021**
0.010	0.005	153.27	153.31	**153.362**	0.101	0.053	**0.005**
0.005	0.002	153.31	153.34	**153.368**	0.053	0.026	**0.001**
0.002	0.001	153.34	153.36	**153.369**	0.026	0.012	**0.002**
0.001	0.001	153.36	153.36	**153.369**	0.012	0.005	**0.002**

Machine Defect

A	B	f(A)	f(B)	Richardson	err(A)	err(B)	err(R)
1.00	0.50	640.789	638.350	**635.911**	3.793	1.354	**1.085**
0.50	0.25	638.350	**636.877**	635.404	1.354	**0.119**	1.592
0.25	0.12	**636.877**	637.288	637.699	**0.119**	0.292	0.703
0.12	0.06	637.288	637.195	**637.102**	0.292	0.199	**0.106**

Figure 5.21: Numerical results for various attempts at Richardson extrapolation for the four models simulated with MEAN. The entries with the lowest error are marked in bold. A and B are the two different values chosen for k. The symbol $f(X)$ represents the result for the threshold X and $Err(X)$ is the error of the result X. $Err(R)$, finally, is the error of the Richardson extrapolation.

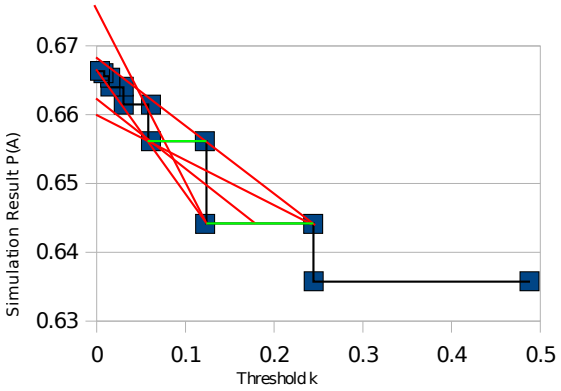


Figure 5.22: Possible and valid Richardson extrapolations for a typical model simulated with the MEAN criterion. The green line segments are the ones from which the extrapolation candidates are chosen. The red rays are possible extrapolation paths, their intersection with the vertical axis are the resulting extrapolations.

Chapter 6

Conclusion

This chapter constitutes the conclusion of this thesis. Section 6.1 summarizes the experimental results from Chapter 5, while Section 6.2 interprets these results with respect to the goals set forth in Section 1.3.

Section 6.3 is concerned with the limitations of the generalizability of the experimental results and the sections 6.4 and 6.5 deal with possible extensions to the proxel method and the benefits of this thesis' findings to practical applications, respectively.

6.1 Summary of Results

Results Concerning the VTS Approach To estimate the computational overhead of the VTS approach, models were simulated with a special VTS criterion that emulates the CTS behavior. The results showed that the VTS algorithm takes about twice as much time to compute the same simulation results as the VTS algorithm. Thus, a VTS criterion would need to reduce the number of proxels in a simulation run by 50% without loosing accuracy to even be on par with the CTS algorithm.

As a side effect, this experiment showed that the simulation results for CTS and VTS are identical if the same step sizes are used for both (meaning that the VTS algorithm emulates the CTS behavior). This implies that both implementations are equally correct. As these results are also very similar to either the analytical solution or results obtained from other sources (e.g. [5, 7]), the CTS and VTS implementations are very likely to produce correct results.

Testing the VTS criteria themselves revealed that only one of them is clearly superior to the CTS algorithm, while another on is usually on par with CTS. All other criteria either produced far less accurate results or failed to complete certain simulation runs.

A closer analysis of the two most efficient criteria showed that the Richardson extrapolation is not sensibly applicable to the best one and applicable only within limits for the other one. Thus, it is far more difficult to assess the accuracy of VTS results than for the CTS ones.

Overall, a VTS criterion has been found that can be used to simulated stochastic models more efficiently that the older CTS approach. Thus, it has been shown that

the approach of using variable time steps is a viable way of improving the simulation accuracy of the proxel method.

Results Concerning the Proxel Approach in General While developing the VTS algorithm and the corresponding criteria, some advances were made that are not only applicable to the VTS approach, but also to the proxel method in general.

It was theorized that two new approaches can improve the simulation accuracy of the proxel method: First, higher-order integration methods can be used to approximate the differential equations underlying the transition probability computations. And second, transitions can be simulated as firing in the middle of a time step instead of at the end of it.

Experiments showed that higher order integration and modified firing times can indeed increase the simulation accuracy for some models. For other models the accuracy is only increased if only one of the two modifications is used. However, it was explained that a feasible approach is to test various combinations of integration methods and firing times with relatively large time steps for a model to determine the best combination for that model. The simulation result with the desired accuracy is then obtained with that selected combination. It was illustrated that this approach can significantly reduce the computation time necessary to obtain results with a certain level of accuracy, even though many short simulation runs must me conducted in advance to determine the most suitable combination.

Finally, it was shown that AVL trees are far more time-efficient in storing and finding proxels than the previously-used regular binary search trees. For some models, accelerating the simulation by factors of ten and more is possible.

6.2 Interpretation of Results

The goal of this thesis was to answer four questions concerning the viability of using variable time steps in the proxel method for the simulation of stochastic systems. Now that all experimental data has been evaluated, these questions can be answered.

1. Under which conditions are variable time steps superior to constant ones?

The answer to this question depends on the actual algorithm used for VTS and the step size criteria. For the binary subdivision algorithm implemented, one of the criteria tested has been shown to be superior to CTS in all test cases.

Another criterion still produces results that are usually similar or better than those of the CTS algorithm.

2. What are the quantitative benefits of using variable time steps?

This question cannot be answered with exact numbers, because the benefits are model-dependent. The speed gain (i.e. the reduction of computation time necessary for the same accuracy) of the VTS algorithm can be made arbitrarily large for artificial models. For the models tested, the VTS algorithm was between two and 100 times[1] faster than the CTS one.

3. What is the best way to determine the time step size under which conditions?

The experiments have shown that the time step size is best determined by the BASE_PROB criterion in all tested cases. This is independent of whether the model contains cycles, transitions with and without limited support, or whether the simulation result measured are part of a stationary solution or are the results of impulse rewards.

4. Is Richardson's extrapolation [10] applicable to the results of variable time steps?

Unfortunately, this important question has to be answered negatively for both competitive VTS criteria. The results for the most efficient criterion exhibit rather chaotic behavior and thus the Richardson extrapolation rarely computes more accurate results than the simulation results used for extrapolation. For second-best criterion, the extrapolation only computes accurate results if the thresholds for the two simulation runs are chosen to differ by powers of two. Both negative results are, however, likely not to be inherent to the idea of variable time steps, but only to the current implementation of binary subdivision.

In addition to the four original questions, additional ideas were developed and tested, showing that the usage of AVL trees as well as higher-order integration methods for differential equations and a modification of the firing times can all significantly increase the simulation accuracy.

Concluding, all four initial questions posed in Section 1.3 have been answered, the fourth question, however, negatively. Thus, the goal of this thesis has been accomplished.

6.3 Limitations

This thesis showed that modifying the proxel method to use variable time steps can significantly improve the simulation efficiency. While the experiments supporting

[1] That speed gain of a factor of 100 was, however, achieved for an artificial model with little practical relevance.

that conclusion were carefully selected to represent typical simulation tasks, the conclusion is still subject to some limitations.

First, the models used for the experiments are only a very small subset of all possible models. They were chosen in order for their results to be generalizable. Thus, tests were conducted on models with and without cycles, with and without rare events, and different model properties (e.g. the result of impulse or parts of the stationary solution) were measured as the simulation result. However, the experimental results cannot be generalized to all simulation models. It might well be that models exist for which the simulation efficiency of the standard CTS algorithm is superior to that of the VTS algorithm using the BASE_PROB criterion.

Second, all experiments were conducted with a single time subdivision scheme - the binary subdivision. The existence of a single subdivision scheme for which the simulation is more efficient than for the CTS algorithms demonstrates the general applicability of variable time steps to the proxel method. The validity of all other conclusions concerning the VTS approach, however, is limited to VTS algorithms using the binary subdivision scheme. Other subdivision schemes are possible (for example the base time step approach in [12, 13]) and those may prove to be even more efficient or the Richardson extrapolation might easily be applicable to their simulation results.

Likewise, better time domain subdivision criteria may exist that allow for even more efficient simulation runs or a general applicability of the Richardson method to their results.

6.4 Possible Extensions and Future Research

While developing the different facets of the VTS algorithm and criteria, many possible improvements to the proxel algorithms have been discovered. Most of them were either too complicated or unrelated to the thesis. Thus, they have neither been implemented nor tested. This section therefore summarizes the ideas that have not yet been described and may be the basis for future research.

Consider the Rates of Successive Transitions in VTS Criteria All but one VTS criterion implemented base the decision about the subdivision of a time step exclusively on the currently active transitions. However, the BA error actually depends on the succeeding transitions. If a transition with a low rate is simulated with a big step size and followed by a transition with a high rate, it is highly likely that the two state changes could have taken place during the single time step that the first transition is simulated with, and thus the BA error is extremely high in these situations.

Hence, for models containing such a situation, the simulation results may be very

inaccurate when using variable time steps. Developing subdivision criteria that consider the rates of successive transitions in addition to the rates of the currently active ones may prove to be vital to efficiently simulate some classes of models.

Develop an Estimate of the ND Error The error caused by the non-deterministic behavior of the system simulated discretely has been observed and described. However, its consequences could not be fully explained and thus, no reliable error estimate has been developed. Finding such an estimate would not only allow for a better understanding of the proxel method, but would enable the UNANIMOUS criterion to work much more reliably, possibly superseding the efficiency of the BASE_PROB criterion.

Structural Analysis of the Model During the experiments conducted to develop, debug and evaluate subdivision criteria and the VTS algorithm, it became clear that it is very hard to determine which changes to the algorithm would affect the simulation result. Since simulation results can take many different forms, such as transient or stationary solutions, as well as the results of impulse rewards, it would be hard to find a generic explanation for the sensitivity of a simulation model.

To determine which transitions need to be simulated with smaller step sizes, the structure of the model's reachability graph could be analyzed. Additionally, statistical data (e.g. the average number of times a transition fires) could be collected during a preceding simulation run.

As a simple example, a model's reachability graph may have two sinks from which probability cannot leave. If only the first sink is of interest to the simulation result, the path to the second sink may be neglected as long as events happening there cannot influence the first sink. For this example, other solutions exist (such as simply removing the path leading to the second sink from the model). However, models may have more subtle situations in which the simulation accuracy of one transitions may not be required to be as high as the accuracy of another transition that follows the same distribution and distribution parameters.

Finding these situations would enable a VTS algorithm to use fewer proxels and hence be faster while maintaining the same level of accuracy.

Merging Proxels with Similar Age Vectors The efficiency of the proxel approach relies heavily on merging proxels with identical system states. In the VTS algorithm, situations may occur where two proxels have the same marking at the same time and their age vector elements are not equal, but at least very similar. In VTS simulation runs, the individual age vector elements of two proxels may differ by far less than the currently simulated time step. As the age values are just approximations, it might be beneficial to merge those proxels with similar age vectors.

As a result, the simulation run may be completed faster and with fewer proxels without impacting the accuracy. Through merging, the state-space explosion might be minimized and hence simulation with smaller time steps may be possible while creating the same overall number of proxels.

This idea, however, leaves some open questions. First, a distance measure needs to be defined on the proxels' age vectors to determine which age vectors are similar enough to be merged. Second, an algorithm must be developed to efficiently find similar proxels. Comparing the age vector of each proxel with each other proxel would be too inefficient and would probably even slow down the simulation.

Self-Correcting Behavior of the Integration All integration methods used are only numerical approximations and are subject to errors. The order of the error is known for all implemented integration methods. This order is valid for the general case, but ODEs with hazard rate functions are a special class: The probability to still stay in a marking after some time has passed will always be monotonously decreasing, and its lower bound and limit are always zero.

The first property is guaranteed even for the numerical approximations by the fact that neither $hrf(t) = 0$ nor the remaining probability can be negative, no matter how crude the integration method is. The second property is ensured by limiting the leaving probability to at most one. Thus, no matter how crude an approximation is, after some time it will reach zero, the same value as for the analytical solution. So the integration's local truncation error (error per time step) does not indefinitely sum up for the global truncation error and the global error is far smaller than the sum of local errors suggests. It could even be possible that the order of error for integrating transition probabilities is higher (i.e. the order's exponent is bigger) than is true for the general case.

If this idea proves to be true, it would have consequences for using the local truncation error as a measure to limit the global error, and for the integration method necessary to achieve a certain level of accuracy.

Global Knowledge to Determine Simulation Step Size The basic idea of variable time steps is to balance computational effort and accuracy. Since each proxel needs the same amount of time to be processed, they all should optimally also hold the same probability mass. Some of the developed criteria tried to achieve this by equally distributing the probability to all proxels of a time step. However, the small differences in probability for one time step can accumulate over the simulation time.

A remedy can be to incorporate global knowledge. If n proxels exist at a given time, each should hold a probability mass of about $1/n$. The subdivision criteria could be modified so than proxels having less probability than that are simulated with bigger time steps than those having a probability mass of more than $1/n$. The

exact changes to the subdivision criteria, however, are still unknown and untested.

Accuracy Bounds Currently, the only way to evaluate VTS algorithms and subdivision criteria is to compare them to CTS algorithm. This allows to determine which algorithm is more efficient, and by what factor.

Another interesting measure would be to know how much better the algorithm could still become. To answer this question, it would be necessary to develop accuracy bounds that can determine how many proxels would be necessary to simulate a given model with a required level of accuracy. Having such bounds would allow to better understand the potential for improvement (or lack thereof) to the VTS algorithm.

Dynamically Switching the Integration Method The basic idea of the UNANIMOUS criterion is to estimate the individual errors caused by simulating a proxel with a certain time step size and to limit the error occurring per time step.

Hitherto, limiting the error is done by subdividing the time step. But in the case of the integration error, it is also possible to redo the calculations with a more accurate integration method (cf. Section 2.2) to reduce the integration error.

So, if the integration error is the highest error for a transition, and if it is the only error that would cause a subdivision of time, recomputing the transition probabilities with a more accurate integration method would reduce the number of proxels needed while maintaining the same level of accuracy.

This would however require at least reliable error estimates, preferably accurate error bounds.

Other Integration Methods The integration methods that were implemented and are suitable for UNANIMOUS are ODE12 and ODE45. ODE45 often is more accurate than required and thus wastes time, while at least the error estimates of ODE12 are too inaccurate to reliably base a subdivision decision on them.

It might be worthwhile to implement additional integration methods. Dormand [1] notes severable other Runge-Kutta integration methods, among those at least one embedded method using second and third order methods.

Classifying Proxels For markings in which no race-age transition and no memoryless transition is active, the active transitions have an age value of zero for proxels that were created as a result of a transition firing, and non-zero for proxels that are the result of inactivity.

Treating these two kinds of proxels differently can significantly speed up a simulation run: Under the conditions mentioned above, two inactivity proxels can never be merged with each other, because if their system states were equal, they would

also have been equal one time step before (and one before that and so on, until at least one of the two has been the result of a transition firing) and would already have been merged by then.

Likewise, if the conditions hold, no "inactivity proxel" can be merged with a proxel that is the result of a transition firing, since the latter does have age values of zero for its active transitions while the former does not.

So the only proxels that can potentially be merged in this situation are two proxels that are both the result of transitions firing. As a consequence, all "inactivity proxels" could be stored in a simple linear list or a dynamic array, since their container does never need to be searched to find duplicate proxels. Proxels that are the result of a transition firing would still be stored in a binary search tree, but this tree would be much smaller and thus less expensive to traverse without the "inactivity proxels"

Investigate the Effect of Various Integration Methods on the VTS Efficiency Experiments have shown that using higher-order integration methods does not generally increase the simulation accuracy for CTS algorithms. For VTS, however, this might be different. Since in a VTS simulation, transitions are simulated with widely varying step sizes, the integration accuracy can be the major error source in simulating the bigger time steps.

6.5 Applications and Benefits

The techniques developed for this thesis are suitable to significantly speed up the simulation of certain classes of stochastic models. Thereby, it is made possible to obtain more accurate results in the same amount of time, reduce the amount of time necessary to retrieve results with a given level of accuracy, or to simulate bigger models in an acceptable time period.

Some of the improvements are applicable to the CTS approach as well as the VTS one and should increase the simulation efficiency for the majority of simulation models.

The efficiency increase gained by introducing variable time steps may significantly speed up simulations in the future. But for now, only one of the implemented criteria computes more accurate results than the standard CTS approach. Furthermore, estimating the result accuracy for this criterion is still an unsolved problem. Consequently, the criterion and thus the entire VTS algorithm are not yet fit for practical applications.

VTS and some of the developed criteria have the potential of significantly increasing the efficiency of the proxel method, if their remaining problems can be solved. Thus, they should be the subject of further research.

Glossary

BA Error Error caused by violating the basic assumption of the proxel method (only one state change can happen during a given time step., 21

CTS Constant Time Steps, 3

ND Error Error caused by the non-deterministic behavior of the system during a time step to be simulated deterministically., 27

ODE Ordinary Differential Equation, 8

ODE Error Error caused by integrating ordinary differential equations, 22

ODE12 Embedded Runge-Kutta integration scheme using first and second order integration methods, 11

ODE45 Embedded Runge-Kutta integration scheme by Dormand and Prince, using fourth and fifth order integration methods, 11

RK4 The fourth-order Runge-Kutta method, 10

SPN Stochastic Petri net, a way of modelling stochastic systems, 17

VTS Variable Time Steps, 3

Bibliography

[1] John R. Dormand. *Numerical Methods for Differential Equations - A Computational Approach*. CRC Press, 1996.

[2] Peter J. Haas. *Stochastic Petri Nets: Modelling, Stability, Simulation*. Springer Series in Operations Research. Springer, 2002.

[3] Graham Horton. A new paradigm for the numerical simulation of stochastic petri nets with general firing times. *European Simulation Symposium*, 2002.

[4] Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 1998.

[5] Claudia Krull. *Discrete-Time Markov Chains: Advanced Applications in Simulation*. PhD thesis, Otto-von-Guericke Universität Magdeburg, 2008.

[6] Hannes Köberle. Analyse und implementierung von strategien zur bestimmung geeigneter schrittweiten für die proxelbasierte simulation von stochastischen modellen. Master's thesis, Otto-von-Guericke-Universität Magdeburg, 2006.

[7] Sanja Lazarova-Molnar. *The Proxel-Based Method: Formalisation, Analysis and Applications*. PhD thesis, Otto-von-Guericke Universität Magdeburg, 2005.

[8] Sanja Lazarova-Molnar and Graham Horton. Proxel-based simulation of a warranty model. In *European Simulation multiconference 2004*. SCS European Publishing House 2004.

[9] Sanja Lazarova-Molnar and Graham Horton. An experimental study of the behaviour of the proxel-based simulation algorithm. *Simulation und Visualisierung 2003, SCS European Publishing House*, 2003.

[10] Lewis F. Richardson. The deferred approach to the limit. part i. *Philosophical Transactions of the Royal Society*, (226):299–361, 1927.

[11] David Stirzaker. *Elementary Probability*. Cambridge University Press, 1994.

[12] Fabian Wickborn and Graham Horton. Feasible state space simulation: Variable time steps for the proxel method. *2nd Balkan Conference in Informatics, 17th-19th November 2005, Ohrid, Institute of Informatics, Faculty of Natural Sciences and Mathematics, Skopje, Macedonia*, 2005.

[13] Fabian Wickborn and Graham Horton. Reducing the effect of stiffness for a state space-based simulation method using adaptive time steps. *6th International Workshop on Rare Event Simulation, 8th-10th October 2006, Bamberg, Germany*, 2006.

Certificate of Originality

I hereby certify that this work has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that this thesis has been written by me. Any help that I have received in my research work and the preparation of the submission itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated.

Magdeburg, September 25, 2008

Robert Buchholz