Script for Advanced Discrete Modelling 2006

Robert Buchholz, Sebastian Schönfeld, Jens Müller

June 20, 2007

Contents

1 Stochastic Processes

Random Variables

A random variable $X(t)$ is a variable that takes on values at random, i.e. it has (potentially) different values for given t's.
If $t \in \mathbb{N}$, X is a discrete time random variable and can also be denoted as X_t. The set of all $X(t)$ is *countable*. An example of discrete time random variables are stock market closing prices, because their values change in discrete intervals (once per trading day).
With $t \in \mathbb{R}$, X is a continous time random variable, i.e. its value can change from one point in time to the next. The common example for continous time random variables is the queue length in a bank, because the queue length may change at any given time.
A random variable X_t is *memoryless*, if its future behaviour only depends on its current state and not on any state before that. The property of memorylessness can also be written as:

$$P(X_{n+1} = X|X_n, X_{n-1}, ..., X_1, X_0) = P(X_{n+1} = X|X_n) \tag{1}$$

State Space

The state space of a random variable $X(t)$ is its domain, i.e. the set of all possible values $X(t)$ can take on.

2 DTMCs - Discrete Time Markov Chains

A discrete time Markov chain or DTMC can be used to describe a system with a *finite* or *infinite countable* number of states. The system may only be in one state at any given time and may only change its state at discrete points in time. For a system to be representable by a DTMC the system has to fulfill the *Markov Property* and therefore must be memoryless. This means that the selection of the next state of the system *must only* depend on the previous state if we look on first order Markov chains. Making this assumption is a simplification of real systems most of the time and often leads to invalid or inaccurate results.
The Markov property is almost true for (among others):

- roulette results
- playing the lottery
- coin tossing

2.1 Representations

2.1.1 Graphical Representation

A DTMC can be represented as a *directed graph* where the nodes represent states of the Markov chain and edges represent transitions between those states. The edge weights denote the probabilities of the respective transitions.
An invariant of all DTMCs is that the sum of all edge weights of edges leaving one node has to be exactly 1.

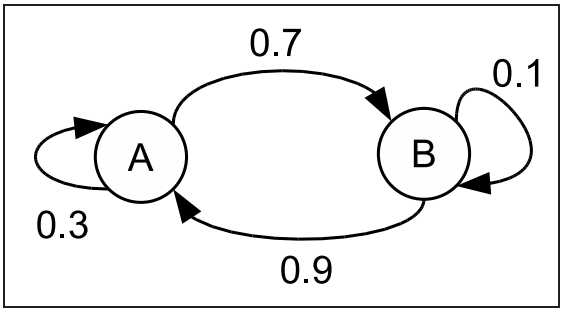

Figure 1: Example of a DTMC

2.1.2 Mathematical Representation

Stochastic Matrices

A square matrix $P_{n\times n}$ is said to be a *stochastic matrix*, if the sum of the values of every single row is one:

$$\sum_j P_{i,j} = 1 \tag{2}$$

So the matrix $A = \begin{pmatrix} 0.3 & 0.7 \\ 0.9 & 0.1 \end{pmatrix}$ is a stochastic matrix (and in addition also a transition probability matrix) and the matrix $B = \begin{pmatrix} 0.9 & -0.4 \\ 0.2 & 1.3 \end{pmatrix}$ is not.

Transition Probability Matrix

If a matrix fullfils the property (2) and additionally the property

$$0 \leq P_{i,j} \leq 1 \ \forall i, j$$

the matrix is said to be a *transition probability matrix*. A transition probability matrix has the form:

$$P = \begin{pmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,n} \\ P_{2,1} & P_{2,2} & \cdots & P_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{n,1} & P_{n,2} & \cdots & P_{n,n} \end{pmatrix}$$

Here, the matrix element $P_{i,j}$ is the probability for the system to go from state i to state j in one step and $P_{i,i}$ is the probability to stay in state i.
Similarly, ${P_{i,j}}^{(n)}$ is the probability for the system to go from state i to state j in exactly n steps.

Probability Vector

A vector containing the probabilities of the system to be in each state at a given point in time is called *probability vector*. It is of the form

$$\Pi = \begin{pmatrix} P(S_1) \\ P(S_2) \\ \vdots \\ P(S_n) \end{pmatrix}$$

where $P(S_j)$ denotes the probability of the system to be in state S_j at the given point in time and will be abbreviated as P_j in the following.

This requires the vector $\Pi^T(P_1, P_2, ..., P_n)$ to satisfy the constraint

$$(\sum_{i=1}^{n} P_i) = 1$$

An *initial state vector* Π_0 is a probability vector describing the system state at time 0 (initial time).
The transition from one DTMC state to the next one can be computed with the *balance equation* (here Π_i denotes the ith element of Π):

$$\Pi_i = \sum_j \Pi_j P_{j,i}$$

This formula can be rewritten to a vector equation and then be used to calculate a state vector Π_{n+1} from a known Π_n:

$$\Pi_{n+1}^T = \Pi_n^T P \tag{3}$$

Steady State

A steady state is a system state (represented by a state vector) that does not change over time. If a DTMC has a steady state, there exists a steady state vector Π_n with:

$$\Pi_n = \Pi_{n+1}$$

By using equation (3) this can also be written as:

$$(\Pi_n)^T = {\Pi_n}^T P$$

However not all DTMCs do have a steady state. Let's take for the example the DTMC desribed by $P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. Solving the DTMC with the initial probabiltiy vector $(\pi_0)^T = (1, 0)$ here will never lead to a steady state solution but to a infinite altering between the values of (1,0) and (0,1). This behavior is called periodic. With the theorem of Person-Froebenius it can be determined if a transient solution will lead to a periodic solution or not.

2.1.3 Introduction of some property definitions

- time homogeneous:
  $P(X_2 = x_2|X_1 = x_1) = P(X_{n+1} = x_2|X_n = x_1)$
  $P_{ij}(n) = p_{ij} = P(X_{n+1} = x_i|X_n = x_i)$
- reachable states:
  $p_{ij}^{(n)} > 0$
- communicating states:
  $p_{ij}^{(n)} > 0$ *and* $p_{ji}^{(n)} > 0$
- irreducible states
  $\forall i, j : i \leftrightarrow j$, in words: all i,j are communicating
- periodic states:
- recurrent states:
  $P_{ii}^{(n)} = 1$, the probability of coming back to this state is one
- transient states:
  $\exists p_{ii}^{(n)} > 0$, the probability to come back to this state is not zero

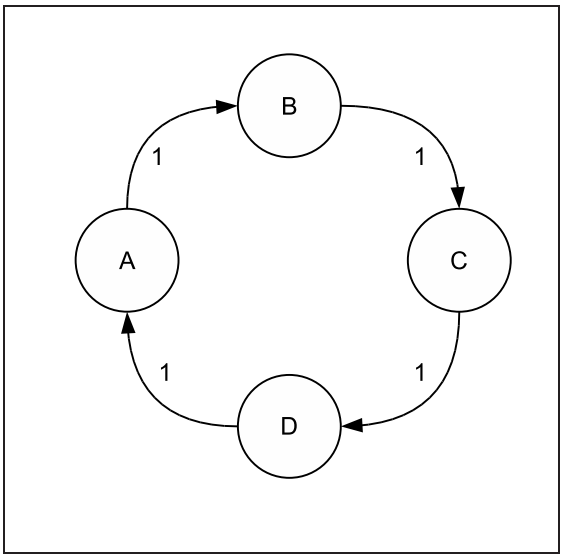

Figure 2: e.g. A is a periodic with d = 4 (d= depth)

- absorbing states:
  $p_{ii} = 1$, the probability to stay in this state is one (there is no possibility to leave this state)
- stationary distribution:
  $\exists \;\; if \;\; \exists z : z \cdot P = z$
- limiting distribution:
  $\exists \pi : \pi = \lim_{n \to \infty} \Pi_n$
- ergodic:
  irreducable, aperiodic and finite ???

If a marcov chain is irreducable and aperiodic then there always exists a π independently from the choosen initial state vector π_0.

2.1.4 Nearly Decomposable DTMCs

If we have given a DTMC as shown in the figure below and we assume that ϵ is very small the computation of a steady state would be expensive. Especially when we use ${\Pi_0}^T = (1, 0, 0, 0)$ as the inistial state vector. To handle this situation we divide the DTMC into two parts and compute several iterations. After that we reconstruct the old DTMC and use the computed values as our new initial state vector. This proceeding makes the computation of the steady state faster but it doesnt solve the problem. How good or bad this method works depends on the first initial state vector, if this one is a bad choise this method doesn't help computing the steady state of the DTMC much faster than before.
It is very important to know how to handle such DTMCs because this problem emerges quite often in practice.

2.2 Solving DTMCs

System of Linear Equations

The naive approach to solve a DTMC - i.e. to calculate its steady state vector - is to solve the system of linear equations given by the steady state equation:

$$\Pi^T = \Pi^T P$$

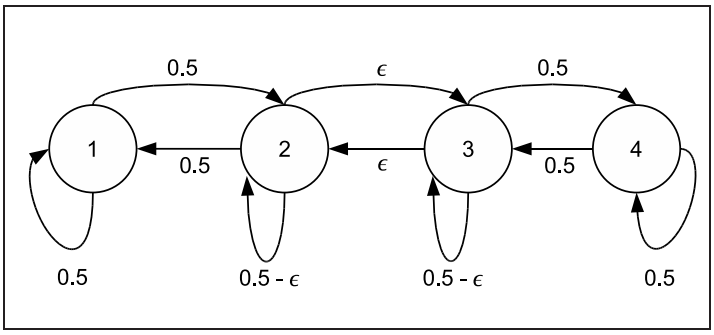

Figure 3: Example of an nearly decomposable DTMC

which can be reorganized to

$$0 = \Pi^T (P - I)$$

where I is the *identity matrix*. (All main diagonal elements are one and all the others zero)
However, multiplying this equation by $(P - I)^{-1}$ would only yield

$$\Pi = 0$$

which is a correct solution for the system of linear equations, but is useless for solving the DTMC as it does not satisfy the constraint for probability vectors ($\sum \Pi_i = 1$). This is because we solved a homogenous system of linear equations, which always has either an infinite number of solutions or only the solution $(0, 0, ..., 0)^T$, but $(0, 0, ..., 0)^T$ is always part of the set of solutions.
To avoid this problem, one can either solve a system of linear equations that incorporates the constraint given above. But that would yield a non-square matrix which cannot be inverted easily. Another approach is to start with a solution that satisfies the constraint given above (e.g. the vector $(1, 0, ..., 0)^T$) and use an iterative algorithm to get better results that are closer to the steady state solution in each step:

2.2.1 Power Method

The simplest iterative algorithm is to compute

$$\Pi_{n+1}^T = \Pi_n^T P$$

over and over again until the result converges to the steady-state solution.
We also can get Π_{n+1}^T by doing the following computation

$$\Pi_{n+1}^T = \Pi_0^T P^{n+1}$$

where only Π_0 and P are needed. This is called the **power method**. It can easily be implemented, but is computationally expensive.

2.2.2 Method of Jacobi

Another iterative approach is to start with the final equation of the system of linear equations

$$0 = \Pi^T \cdot (P - I)$$

or by merging $(P - I)$ to A

$$0 = \Pi^T \cdot A$$

Now, instead of solving this equation, one decomposes A to an *upper triangle* (U), a *lower triangle* (L) and a *diagonal matrix* (D), so that

$$A = D - (L + U)$$

This yields

$$0 = \Pi^T(D - (L + U))$$

rearranged

$$\Pi^T(L + U) = \Pi^T D$$

which in turn can be used as a iterative algorithm:

$$\Pi_k^T(L + U) = \Pi_{k+1}^T D$$

as soon as it is rearranged to

$$\Pi_{k+1}^T = \Pi_k^T \cdot D^{-1} \cdot (L + U)$$

2.2.3 Gauss-Seidel Method

Here nearly the same approach as for the Jacobi method is used. We start again with $0 = \Pi^T \cdot A$ But use annother composition for A:

$$A = U + L + D$$

This yields

$$0 = \Pi^T(U + L + D)$$

rearranged

$$\Pi^T(D + L) = -\Pi^T U$$

which in turn can be used as a iterative algorithm:

$$\Pi_{k+1}^T(D + L) = -\Pi_k^T U$$

as soon as it is rearranged to

$$\Pi_{k+1}^T = -\Pi_k^T \cdot U \cdot (D + L)^{-1}$$

It has been proven that this algorithm converges, always calculates the correct solution with respect to the constraint for the steady-state solution and is in most cases faster than the power method (in pracise often twice as fast as the power method) and never slower. The proof for correctness itself however would go far beyond the scope of this script and is therefore omitted.

2.3 Example(s)

Which meal for lunch?

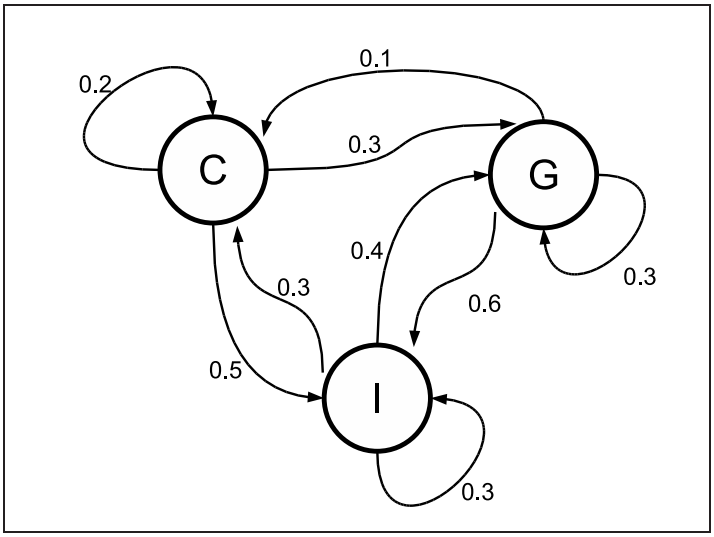

Figure 4: Chinese, Greek or Italian food for lunch?

- task definition:
  - Every noon Mr. M. Infidel goes for lunch to a {Chinese, Greek or Italian} restaurant. A private investigator hired by his wife wants know which restaurant he should observate in the next days and got the following data from Mrs. Bond:
  - probability vector (also initial state vector) $\pi_0 = \begin{pmatrix} \pi_C \\ \pi_G \\ \pi_I \end{pmatrix}_{3x1} = \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \end{pmatrix}$
  - transition probability matrix (selected from given picture) $\mathrm{P} = \begin{pmatrix} 0.2 & 0.3 & 0.5 \\ 0.1 & 0.3 & 0.3 \\ 0.3 & 0.4 & 0.3 \end{pmatrix}_{3x3}$
- solution:
  $\pi_{k+1} = \pi_k \cdot P$

  auxiliary calculation: $\pi_1 = \pi_0 \cdot P$ $\quad \downarrow \quad \begin{pmatrix} 0.2 & 0.3 & 0.5 \\ 0.1 & 0.3 & 0.3 \\ 0.3 & 0.4 & 0.3 \end{pmatrix}_{\underline{3}x3}$

  $\rightarrow$

  ${\pi_0}^T = (1.0\ 0.0\ 0.0)_{1x\underline{3}}$ $\qquad {\pi_1}^T = (0.2\ 0.3\ 0.5)$

  ${\pi_1}^T = (0.2000\ 0.3000\ 0.5000)$
  ${\pi_2}^T = (0.2200\ 0.3500\ 0.4300)$
  ${\pi_3}^T = (0.2080\ 0.3430\ 0.4490)$
  ${\pi_4}^T = (0.2106\ 0.3449\ 0.4445)$
  ${\pi_5}^T = (0.2100\ 0.3445\ 0.4456)$
  ${\pi_6}^T = (0.2101\ 0.3446\ 0.4453)$
  ${\pi_7}^T = (0.2101\ 0.3445\ 0.4454)$
  ${\pi_8}^T = (0.2101\ 0.3445\ 0.4454)$

Our private investigator is lucky because the DTMC has a steady state. He knows now that the probability to meet the husband on the next day at the Italian restaurant is 44.56 %, the probability to meet him at the Greek one is 34.45 % and the probability to meet him at the Chinese restaurant is 21 %.

NOTE:
If we want to have the probability to meet the husband on the next day at a certain restaurant depending on the last two days we have to introduce several new states.
The new states are II,IC,IG,CC,CI,CG,GG,GI and GC.

This shows that that the use of a DTMC is limited to cases where the history of the system has no effect on the current system behaviour. In practice this almost never the case and if we want to use a DTMC to simulate the current behaviour in dependence on the history of the system our DTMC states will increase exponential.

3 CTMCs - Continuous Time Markov Chains

DTMCs are a useful tool in simulation but somewhat limited, because their state changes only can happen at discrete points in time (which accounts for the name *discrete time* Markov chains). To overcome this limitation, we introduce *continuous time* Markov chains. Here, the system changes continuously and is therefore described by a set of *rates of change* rather than probabilities. But to better handle the system mathematically CTMCs limit those rates to be constant over time, hence the underlying *distribution function* has to be an *exponential distribution*.

3.1 Representations

Graphical Representation

A CTMC can be represented graphically much like a DTMC as a directed graph with *nodes* and *edges*. Nodes still represent states in which the system can be at a given time. However, for a CTMC, the edge weights do not represent probabilities, but the change rates. For that reason, the invariants that were true for DTMCs no longer apply: **no** invariant exists for the sum of the *outflows* of a given state. Additionally in a CTMC graph, there can be no edges from a node to itself.

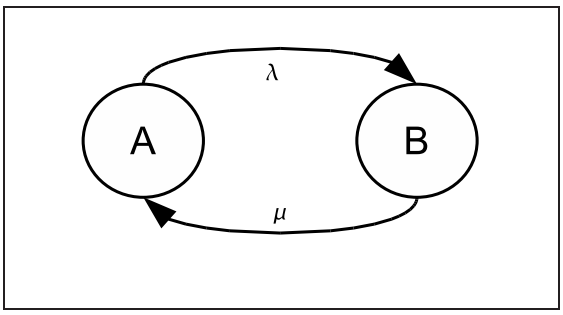

Figure 5: Example of a CTMC (note: λ and μ are exponential distributed)

Mathematical representation

The central component of the mathematical representation of a CTMC is the *rate of change* λ, which represents the parameter of the corresponding exponential *probability density function*

$f(t) = \lambda e^{-\lambda t}$ and *cumulative distribution function* $F(t) = 1 - e^{-\lambda t}$. These functions in turn are describing the flow of probability from one state to another over time.
These change rates are put together in a square matrix Q, where $q_{i,j}$ denotes the change rate from state i to state j. As there can be no change rate to stay in a state i, $q_{i,i}$ is used to represent the alltogether *outflow* from state i, i.e. the rate at which probability leaves state i. Therefore $q_{i,i}$ is computed as

$$q_{i,i} = -\sum_{j,j\neq i} q_{i,j} \tag{4}$$

so that

$$\sum_{i} q_{x,i} = 0 \ \forall x \tag{5}$$

Hence the matrix $Q_{n\times n}$ has the form:

$$Q = \begin{pmatrix} -(\sum_{j\neq 1} q_{1,j}) & q_{1,2} & \cdots & q_{1,n} \\ q_{2,1} & -(\sum_{j\neq 2} q_{2,j}) & \cdots & q_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{n,1} & q_{n,2} & \cdots & -(\sum_{j\neq n} q_{n,j}) \end{pmatrix}$$

A matrix Q having these properties is called an *(infinitesimal) generator matrix*. For our simple example CTMC in section 3.1 it looks so: $Q = \begin{pmatrix} -\pi & \pi \\ \mu & -\mu \end{pmatrix}$.

The representation of a system state is identical to that of a DTMC: a state is represented by a *probability vector* $\Pi^T = (\Pi_1, \Pi_2, ..., \Pi_n)$, where Π_i is the probability of the system to be in state i and $\sum_i \Pi_i = 1$

3.2 Transient Behavior and Steady State

Given a probability vector $\Pi^T = (\Pi_1, \Pi_2, ..., \Pi_n)$ describing the state of the system at a given time, the change of probability of the system to be in a certain state can be described as the difference between the inflow and outflow of probability from this state, that is:

$$\frac{d\Pi_i}{dt} = In_i - Out_i = (\sum_{j,j\neq i} \Pi_j q_{j,i}) - (\sum_{j,j\neq i} \Pi_i q_{i,j})$$

Considering equation (4), this can also be written as

$$\frac{d\Pi_i}{dt} = In_i - Out_i = (\sum_{j,j\neq i} \Pi_j q_{j,i}) + \Pi_i q_{i,i}$$

and putting the last summand into the sum:

$$\frac{d\Pi_i}{dt} = In_i - Out_i = \sum_{j} \Pi_j q_{j,i}$$

This equation can be generalized for the whole vector Π:

$$\frac{d\Pi}{dt} = Q^T \cdot \Pi \tag{6}$$

The steady state can be described with the help of equation (6) as

$$\frac{d\Pi}{dt} = Q^T \Pi = 0 \tag{7}$$

which is true if and only if $Q^T \Pi = 0$

3.3 Calculating the steady-state solution

System of Linear Equations

The system of linear equations needed to calculate a steady state for CTMCs ($Q^T\Pi = 0$) has the same shortcomings as the system used in (2.2) to compute steady-state solutions for DTMCs, i.e. it has the trivial (and useless) solution $\Pi = (0, 0, ..., 0)$ coming from the fact that the constraint for probability vectors is not incorporated in it. Therefore, solving this *homogenous* system of linear equations is not a suitable approach.

Iterative Approach

Similar to DTMCs, the iterative approach to calculate the steady-state solution is promising. Here the matrix Q is split into two matrices M and N, so that

$$Q = M - N \tag{8}$$

inserting this in equation (7):

$$(M - N)^T\Pi = 0$$

This formula can be rearranged to

$$M^T\Pi = N^T\Pi$$

and made into an iterative algorithm:

$$M^T\Pi_{k+1} = N^T\Pi_k$$

which can be rearranged again to yield the final equation

$$\Pi_{k+1} = (M^T)^{-1}N^T\Pi_k$$

This algorithm can be used with any initial probability vector Π to compute better and better results. To do this, one has to compute the matrix $H = (M^T)^{-1}N^T$ only once and then use it to compute $\Pi_{k+1} = H\Pi_k$ over and over again.
The correctness of this approach has been proven for at least two splittings of Q into M and N:

- Jacobi method: set M to be the diagonal matrix part of Q and N to be the negative sum of the upper and lower triangle parts of Q - both **not** including the diagonal.

$$\pi^{k+1} = D^{-1}(L + U)\pi^(k)$$

- Gauss-Seidel method: set M to be the sum of the diagonal and the upper triangle parts of Q and N to be the negative lower triangle part of Q. Same as with the Jacobi method, the upper and lower triangle parts both **do not** include the diagonal part, so that equation (8) becomes $M - N = (D + U) - (-L) = Q$

$$\pi^{k+1} = (D - L)^{-1}U\pi^(k)$$

- SOR (succesive overelaxation)
  - "tuned" of Gauss-Seidel
  - relaxation parameter $\omega = 1/\alpha$
  - converges only if $0 < \omega < 2$
  - $\pi^{k+1} = (D - \omega L)^{-1}[(1 - \omega U)I + \omega]$
  - clever choice of ω can lead to dramatic speedup - usually aroun 1.6

Even so the Jacobi method may seem faster, because its M is a diagonal matrix and can therefore easily be inverted (${m_{i,j}}^{-1} = \frac{1}{m_{i,j}}$), generally the Gauss-Seidel method converges twice as fast as the Jacobi method rendering the matrix inversion step negligible.

3.4 Calculating transient solutions

Depending on the given task, one may need to calculate the system state of a CTMC at a given point in time. For DTMCs, we could just iteratively calculate the next steps starting from the initial state vector and iterate until we arrive at the requested point in time. However, CTMCs don't have fixed time steps by which we could advance through time. To compute $\pi(t)$ the following three approaches can be used.

3.4.1 Solving of the initial value problem

The behavior of a CTMC with generator matrix Q can be described by (6):

$$\frac{d\Pi}{dt} = Q^T \Pi$$

In case the *initial state vector* $\Pi(0)$ is known (or can be set to be any probability vector), we have to solve an ordinary *initial value problem*. This class of problems can be solved for any t by means of analytical or numerical integration. However, all limitations of these methods apply: analytical integration in most cases cannot be done by a computer and numerical integration needs advanced algorithms to control accuracy (e.g. Runge-Kutta method) and are computationally expensive.

3.4.2 Converting CTMCs to DTMCs

An alternative approach is to transform a CTMC into a DTMC and use the known mechanisms for DTMCs.
In order to discretize the time domain to convert the CTMC to a DTMC, one has to choose the time difference Δt that should elapse between two DTMC states. Because in a DTMC the system can only change its state once per time interval, we must choose Δt such that in this interval a state change doesn't happen more than once (at least most of the time).
Considering that all non-diagonal elements λ of a *generator matrix* represent an exponential distribution function and the *expected value* of an exponential distribution is $\frac{1}{\lambda}$, we should choose Δt to be smaller than the smallest expected value of any distribution function:

$$\Delta t < \frac{1}{q_{i,j}} \; \forall (i,j), i \neq j$$

Now, to create a DTMC we have to remember, that each non-diagonal value λ of Q represents a rate of change. That is, a λ of $2s^{-1}$ means that the corresponding event will occur twice per second. To create a transition probability from that value, we just have to multiply it with the choosen duration of time step Δt (i.e. an event with a rate of $0.3s^{-1}$ will occur during the time $\Delta t = 0.4s$ with a probability of $0.3s^{-1} * 0.4s = 0.12$).
The last things to consider are the diagonal elements of Q. For a generator matrix Q, the row sum of all elements is 0 (see equation (5)) and this stays true when multiplying the matrix with Δt. In order to increase the row sum to 1 - as it has to be for DTMCs, see equation (2) - we have to add 1 somewhere in each line. As the non-diagonal elements already contain a probability which must not be changed, that leaves us with diagonal elements only.
Another view on this is, that the diagonal elements of a *generator matrix* Q contain the rates at which probability leaves the corresponding state, but all with a negative sign. Multiplying the whole matrix with Δt to transform all rates to transition probabilities, the diagonal elements contain the probability $-p$ to leave the corresponding state in one step - still with a negative sign. To convert this probability to the probability to **stay** in the state, we just need to calculate the counter-event to p which has the probability of $1 - p$. To facilitate that, we just add 1 to all diagonal elements, i.e. adding an identity matrix to $\Delta t\ Q$.
Summing it all up, a DTMC can be created from a CTMC by the means of the following formula:

$$P = I + \Delta t\ Q$$

However it is noteworthy, that this conversion only approximates the behavior of the CTMC. That is because in a DTMC only one state change can occur for each transition. So we made it uncertain, but not impossible that during that time multiple state changes would have happened in the system described by the CTMC.
Furthermore, we can only calculate the transient states of the DTMC at times $n * \Delta t$ with $n \in \mathbb{N}$, but may wish to know the CTMC's state for a different point in time.

3.4.3 Uniformization (also called Jensen's Method or Method of Randomization)

Another clever approach that let us control the result's accuracy is the uniformization method. The problem at hand is given by the equations

$$\frac{d\Pi}{dt} = Q\ \Pi \tag{9}$$
$$\Pi(0) = \Pi_0 \tag{10}$$

Which is an *initial value problem* of the form

$$\frac{dy}{dt} = c_1 \cdot t$$
$$y(0) = c_2$$

and has the solution

$$y(t) = c_2\ e^{c_1\ t}$$

This equation is valid not only for scalar constants c_1 and c_2, but also for vectors and matrices. Therefore our problem (9/10) can be written as

$$\Pi(t) = \Pi_0\ e^{Qt} \tag{11}$$

Now the previous section tells us that a *generator matrix* Q can be converted into a *transition probability matrix* P by the means of

$$P = I + \Delta t\ Q$$

By substituting Δt by $\frac{1}{q}$, multiplying the equation by q and rearranging we obtain

$$Q = qP - qI$$

which can be inserted in (11):

$$\Pi(t) = \Pi_0 \cdot e^{(qP-qI)t}$$

and rearranged by resolving the brackets, writing the power as two separate powers and ommiting the identity matrix:

$$\Pi(t) = \Pi_0 \cdot e^{qtP} \cdot e^{-qt} \tag{12}$$

Now remembering that every function including e^{qtP} can be approximated by a *Taylor series*

$$e^{qtP} = \sum_{k=0}^{\infty} \frac{(qtP)^k}{k!}$$

we can insert this approximation into (12)

$$\Pi(t) = \Pi_0 \cdot e^{-qt} \cdot \sum_{k=0}^{\infty} \frac{(qtP)^k}{k!}$$

separate the scalar (qt) and matrix (P) powers and move Π_0 and e^{-qt} into the sum:

$$\Pi(t) = \sum_{k=0}^{\infty} e^{-qt} \cdot \frac{(qt)^k}{k!} P^k \Pi_0$$

Finally, we use equation (3) as a shortcut to not to have to compute P^K over and over again:

$$\Pi(t) = \sum_{k=0}^{\infty} e^{-qt} \cdot \frac{(qt)^k}{k!} \Pi_k \tag{13}$$

Now this equation can be used to calculate $\Pi(t)$ with any required accuracy. This is because in the equation, Π_k is a probability vector and therefore all elements we be in the range $[0, 1]$, and the term $\sum_{k=0}^{\infty} e^{-qt} \cdot \frac{(qt)^k}{k!}$ represents *Poisson coefficients* which get very small after a certain k so that later summands contribute less and less to the overall sum.
Note: Throughout this derivation, the transposition of matrices Q and P has been omitted for better readability. However to obtain currect results, these have to be reinserted ($Q \rightarrow Q^T$ and $P \rightarrow P^T$).

Dual View on Jensen's Method

We showed the derivation of Jensens's Method and the obvious way of looking at it is as a way of directly calculating $\Pi(t)$. However another view is possible:
In each iteration for k in (13), Π_k is the k*th* state of the corresponding DTMC and $e^{-qt} \cdot \frac{(qt)^k}{k!}$ is the probability that this state will actually occur. So another way of looking at the equation is, that it transforms the CTMC to a DTMC, calculates the sequence of all possible states (Π_k with $k \in 0, 1, ...$) of that DTMC and computes the probability for each k, that the DTMC would do exactly k steps in time t. The result of the CTMC is the sum of all those products of system states and probabilites.

3.5 Example(s)

Converting CTMC to DTMC

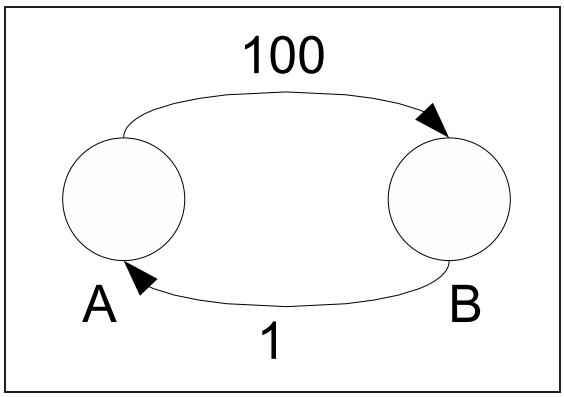

Figure 6: CTMC which will be converted into a DTMC

We want to convert the above CTMC in a DTMC. The generator matrix of our CTMC is $\begin{pmatrix} -100 & 100 \\ 1 & -1 \end{pmatrix}$ and our probabilty vector $\pi^T = (1, 0)$. With the help of the formula $\Delta t < \frac{1}{max(q_{i,j})}$ we calculate $\Delta t = 0.01$. We can now compute the transition probability matrix P with the formula $P = I + \Delta t * Q = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} -0.1 & 0.1 \\ 0.01 & -0.01 \end{pmatrix} = \begin{pmatrix} 0.9 & 0.1 \\ 0.01 & 0.99 \end{pmatrix}$.

4 GSPNs - Generalized Stochatisc Petri Nets

Stochastic petri nets are an easy way of creating models for discrete event simulations (*DES*). Measueres for GSPN can be:

- P(place is empty)
- average number of tokens in a place
- throughput of a transition
- P(Transition was active)

Unfortunately, they have the huge drawback that to calculate the steady-state or a transient solution, one has to do replications of the model over and over again - potentially for millions of times. On the other hand algorithms exist to efficiently solve discrete and continous time Markov chains, but the underlying models can only be build by experts.
So an ideal blend of these two approaches would be to create a model as a petri net and simulate it using Markov chains. Fortunately this is indeed possible - with some restrictions.

4.1 Restrictions of GSPNs

Continous time Markov chains have the limitation of allowing transitions only to be *exponential distributed*. The consequence is that for petri nets to be transformed to Markov chains, those petri nets' *timed transitions* must follow exponential distributions as well. And that's exactly what *generalized stochastic petri nets* (GSPNs) are: stochastic petri nets with their timed transitions being limited to exponential distribution functions and with the additional support of immediate transitions.

4.2 Converting GSPNs to CTMCs

In (G)SPNs a *place* represents a physical or virtual location in the model. On the other hand, a state of a CTMC represents a state of the whole system.
For example, consider a petri net of a network buffer with only one place and one input and one output transition: The place actually represents the buffer and the tokens that travel through it may represent network packets. In CTMCs, each state of a CTMC represents a state of the whole system. Looking at the network buffer example, one state of the CTMC might represent the situation where exactly one token is inside the buffer. For every other number of tokens at the place an additional CTMC state is necessary. If a GSPN model is larger than just one place, a CTMC state has to be created for all possible combinations of place occupations by token.
The set of all those states of a petri net that can occur is called its *state space*.
To convert a GSPN to a CTMC, the first step is to create the full state space of the petri net. One does that by putting the (known) initial state of the GSPN as the first element in the state space graph and then recursively adding all states that this state may lead to. Simultaneously one connects the left and the reached state by a directed edge and notes the probability of reaching the state as the edge weight: for timed transitions with exponential distribution this is the functions rate λ and for immediate transitions, this is the weighted probability, i.e. a probability so that the overall probability of all active timed transitions leaving this state is **1**.
One pitfall is, that in a situation of the GSPN where *timed* and *immediate transitions* are active, the immediate transitions **always** fire first, i.e. a state can **never** be left by a timed **and** an immediate transition.
The resulting graph is called a *reachability graph*, as it contains all reachable system states and the transitions between those. Looking at the network buffer example, it is obvious that the state space can get quite large and for this example even infinite. In order to handle and solve the resulting CTMC, one has to at least make that reachability graph finite. This is done by preventing petri net places to potentially contain an infinite number of tokens, generally by setting *inhibitor arcs*.

This procedure will modify the systems behaviour and may lead to CTMC results that do not accurately reflect the GSPN's behaviour.

4.2.1 Reduced reachability graph

The completed reachability graph contains timed and immediate transitions. While the purpose of CTMCs is to handle a state change over time (i.e. by rates of change), it in no way supports immediate state changes. So to make the reachability graph compatible with CTMCs, we need to eliminate all immediate transitions. The first step to do this is to classify all those states as *vanishing states* that have leaving arcs with immediate transitions and all others as *tangible states*. Afterwards all vanishing states are removed from the graph and the weighted leaving probabilities are integrated as factors into the entering distribution function (see example).
Now that we have reduced the reachability graph to only contain tangible states and transitions between those states that are expontially distributed, we can directly interpret this graph as a CTMC.

4.3 Example(s)

We are observing a pedestrian crossing which is controlled by a traffic light (Fußgängerampel). The pedestrian light can be either red or green. Pedestrians arrive at random intervals I. When the traffic light is green, they immediately cross the street. When the light is red, they either wait for the light to change (with probability p) or leave. The duration of the green phase is described by the random variable G, and the duration of the red phase by R. Let's assume that in the beginning there is no pedestrian waiting at the traffic light, and that the traffic light is red. We can now draw a stochastic Petri net model of this system like the following:

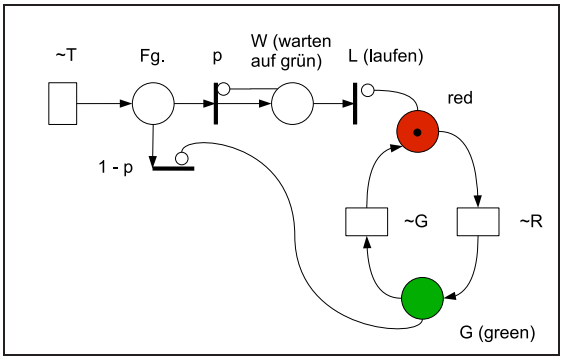

Figure 7: GSPN for the traffic light example

To make the state space of the model finite, we limit the number of people that can wait at a traffic light to one. We code the four places of our GSPN in a Triple (Place Fg., Place W, Place G). We can omit place R because if there is no token in Place G there must be a token in Place R according to our GSPN. We than start with the initial state (0,0,0), which means that there is no pedestrian currently on the street and our traffic light is in his red phase. From this point two different events can happen. On the one hand a pedestrian could arrive (timed transition I fires) and lead us to state (1,0,0). On the other hand the traffic light could switch to green (timed transition R fires) and lead us to state (0,0,1). Drawing this and all the following event possibilities leads us to the following reachability graph:
If we assume p = 0.5 and the rate 0.5/min for I and 2/min for G and 1/min for R. we can derive the continous-time Markov chain shown on the right side of the underneath grahic. On the left side the reachability graph after deleting all vanishing states can be seen. Note that the transition 1-p which goes out from state (1,0,0) doesn't have influence on state (0,1,0) and is therefor not considered. So if we would use p = 0.25 the transition from 0/R to 1/R would be p*I=0.125.

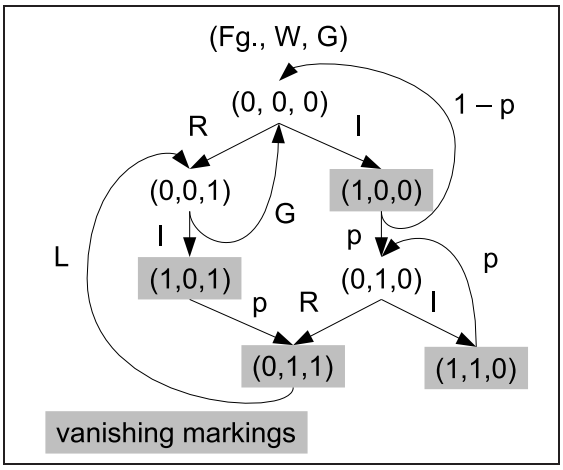

Figure 8: Reachability Graph

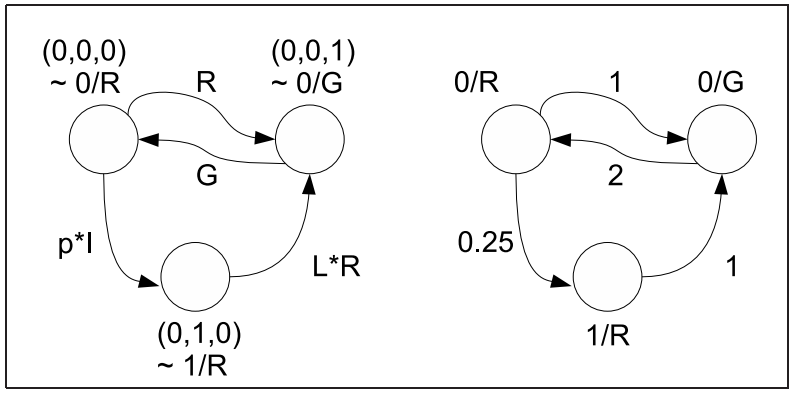

Figure 9: CTMC derived from reachability graph

To get the generator matrix Q we first fill in in the probability rates to go from one state to another in our matrix where the horizontal and vertical order is 0/R, 1/R and 0/G. After that we can calculate the missing value per line by adding all the other values of the line und multiplying it with -1. This leads to:

$$Q = \begin{pmatrix} -1.25 & 0.25 & 1 \\ 0 & -1 & 1 \\ 2 & 0 & -2 \end{pmatrix}$$

The corresponding probability vector for our initial state is $\pi^T = (1, 0, 0)$.

5 Proxels

So far, we can create and simulate both, Markov chains and Petri nets and can even transform GSPNs to CTMCs to combine rapid model development with fast CTMC solvers. However, there still exist a number of cases where CTMCs are insufficient:

- CTMCs only support exponential distributions. Most processes of systems of interest for simulation are not exponentially distributed. Depending on the distinct system to be simulated, this may reduce the results correctness to their mean value (i.e. the *standard deviation* of the models results does not correspond to that of the system) or even make the results completely useless.
- during the conversion from petri nets to CTMCs, one has to limit the number of possible

states. This can only be done by limiting the capacity of the places without being able to calculate the error that is being introduced by that action. A better approach would be to limit the state space based on the importance of the states.

Proxels (*probability elements*) are an attempt to overcome these limitations. The basic idea of proxel simulation is to simulate the system in discrete (*and currently constant)* time steps (like DTMCs) and to compute only those transitions that can actually occur from the particular previous point in time (unlike DTMCs). As proxels also record the time being spent in the current state, proxel simulations are capable of using general distribution functions, not just memoryless ones.

5.1 Hazard rate function

To simulate a system with discrete time intervals, one needs to be able to calculate the probability of events to occur during the next interval. The cumulative distribution function $F(t)$ of a transition calculates the probability of an event to occur before t and $F(t_2) - F(t_1)$ gives us the probability of the event to occur during the time interval $[t_1, t_2]$. This computation can be used for the first interval. But if the event did not happen during the first interval, we do not just need the probability of the event to occur in the second interval, but the probability of the event to occur in the second interval **under the condition** that it has not happened before:

$$P(X < x_0 + \Delta x | X \geq x_0)$$

To accommodate for this, we introduce the *hazard rate function* or *instantaneous rate function* which is defined as

$$\mu(x) = \frac{f(x)}{1 - F(x)}$$

That is, $\mu(x)$ is the rate of change under the condition that the event has not occured yet.
Now, the probability of an event to happen between the times t_1 and t_2 if it has not happened before t_1 is

$$P(X < t_1 + \Delta t | X \geq t_1) = \int_{t_1}^{t_1+\Delta t} \mu(t)dt$$

and can be approximated using rectangular approximation as

$$\mu(t_1) * \Delta t$$

5.2 Definition

A proxel is defined as a tupel $P_x = (S, t, R, P_r)$ and $S = (m, \vec{\tau})$. It has the following meaning:

S	state of the system
t	system time
R	the complete history of the proxel, i.e. the tree of all states leading to this one
P_r	the probability of the system to be in the state desribed by the proxel at time t
m	marking - an unique identification of the state of the system
τ	age vector

Some elements of P_x may require further explanation: m is used to describe the discrete state of the system. If a proxel is seen as a description of a certain state of a corresponding petri net, m would describe the number of tokens in each location. However, m does not need to include those numbers, there just needs to exist a bijective mapping to m.
Another "strange" element of proxels is the age vector τ. It is used to record the age of all active non-exponential transitions and the age of all inactive race age transitions that have been active after their last firing. Here, *age* means the time the transition has been active since the last time

it fired.
In practice, most of the time it is sufficient for a proxel to just contain the tupel (m, τ, P_r). The components of S have been integrated directly into P_x, the history R is often not necessary and at least partially recorded by the structure of the proxel tree, and the system time t is recorded for the list of all proxels at time t and therefore does not need to be recorded for each proxel seperately.

5.3 Implementation

A proxel simulation is started by creating an initial Proxel $P_0(m_0, (0, 0, ..., 0), 1)$ at time $t = 0$. The proxel has an initial marking, all elements of the age vector are zero (because no transition has been active yet) and the probability is one (because it is the only proxel at time $t = 0$).
The next step is to create all proxels at time $t = 1\ \Delta t$ that can be created from the initial proxel: The marking of each proxel is set to reflect the situation made by the event leading to its creation. Events can be firings of transitions or no event at all. The latter happens if it is possible for the system to make no change for a whole Δt. The age vector of the newly created proxels is an updated version of their parent's age vector: the age of the transition that fired (if any) is set to 0, the age of all other active transitions increases by Δt. The probability of the newly created proxel depends on the transition that had fired: The probability P'_r of a proxel created when transition x fired and the probability of the parent state was P_r is:

$$P'_r = \mu_x(t)\ \Delta t\ P_r$$

And the probability P'_r of the proxel that was created when no transition fired at all, the probabilities of all other proxels at that time are given by P'_{r_x} and the probability of the parent proxel is P_r is:

$$P'_r = P_r - \sum_x P'_{r_x}$$

As a result, the combined probability of all proxels created from a common parent proxel has the same value as the parent proxel's probability. As this is true for all proxels and their respective parents it follows that the combined probability of all proxels of time t is exactly one.

Pitfalls

Some pitfalls in proxel simulations result from the rectangular integration instead of a "proper" analytical one. This rough approximation is good enough for most situations but two of those require special treatment:
First, in case that $\mu(0) = 0$, the transition will **never** fire before it has not aged at least one Δt, because its approximated probability is $\mu(0) * \Delta t = 0$, even if the real probability $\int_0^{\Delta t} \mu(t)dt \neq 0$. This can be solved by using *trapezoid integration* as an approximation:

$$P(X < t_1 + \Delta x | X \geq x_1) \approx \frac{\mu(t_1) + \mu(t_1 + \Delta t)}{2} * \Delta t$$

Second, it is possible for $\mu(t)$ to increase beyond 1 because of numerical integration, discrete time steps and poles/singularities in the instantaneous rate function. This would result in a probability of a child proxels that is bigger than the one of the parent or it would cause the proxel that is the result of total inaction during this time step to have a negative probability (see equation (5.3)). To solve this problem, one completely ommits the proxels that represents inaction and normalizes the probabilities of all other transitions leaving the parent proxel to sum up to 1.

Merging proxels

When creating proxels for a new point in time, it happens that one is about to create a proxel $P'(m, \tau, p_2)$ and a proxel with the same marking and age vector $P(m, \tau, p_1)$ already exists. In

this case, instead of creating the new proxel, one has the option of merging the two proxels to a new proxel ($P_{new}(m, \tau, p_1 + p_2)$), preferably by just increasing the probability of the first created proxel. The merger not only has the advantage of saving one proxel: We also save the memory and computational power of creating the children of two proxels, the grandchildren of those and so on and so forth. In practice, the number of proxels per time step increases exponentially with the number of time steps when not merging proxels. With merging however, the number only increases linearly and most of the time even converges to a constant value, which is the reason that makes calculations of a longer simulation time possible.
The merging approach has the following disadvantage: in order to merge proxels, one has to find the candidates for merging. So for each newly created proxel, one has to find the one other proxel that exists at the same point in time and has the same values for m and τ. In order to do so, one would have to scan through all elements of the list or array for that point in time resulting in a O(n) complexity for inserting each proxel and a complexity of about $O(n^2)$ for inserting all proxels for the nth point in time. To lessen this problem, one may use binary (or n-ary) search trees to reduce this complexity to $O(n\ log(n))$ or even use self-balancing search trees (AVL trees, red-black trees, 2-4 trees) to guarantee the complexity of $O(log(n))$ for the insertion of a proxel.

Inaccuracies

The results of a proxel simulation run are not completely the same as the results of the simulated system. A small error always remains. This error is dependant of Δt for the following reasons:
First, the choosen method for numerical integration is a very rough approximation (however a better approximation might take longer than to reduce the size of Δt and make more replications). The error for this approximation is reduced with decreasing Δt.
Second, proxel simulation is based on the assumption that during each time step, only one transition fires. To clarify: We create different proxels for each transition that **may** fire, but for each proxel only one transition actually **has** fired. This assumption becomes better and better with reduced Δt because the smaller the time step is, the smaller is the chance that two transitions would be able to fire during it.
These two errors lead to the obvious conclusion that the accuracy of the results can be increased by reducing the time step size. However by doing so, the computational complexity is increased on at least two levels:

- We need more memory to store the proxels, because more proxels exist for each time step. The reason for this is that the more time steps exist between the activation of a transition and its firing, the more proxels exist with different age vectors that cannot be merged. For multiple memoryless active transitions, this can even lead to a *state space explosion* - an exponential grow of the number of proxels in each step.

- We need more computation time, especially to calculate transition probabilities and to find merging candidates.

Fortunately, there exists another way of increasing the accuracy of the results: The errors in our error categories (integration errors and discretization errors) are almost completely *linearly dependent* of Δt, i.e. reducing Δt by 50% will also reduce the error by about 50%. As this also at least doubles the computational complexity, this cannot be done often before the time needed for a computation reaches days and more. However what can be done is the so-called *Richardson-Extrapolation*: When plotting a value of the simulation results (e.g. the probability of the system to be in state m) against its time step size Δt, all those points lie close to a straight line. This line can be found by mathematical means (e.g. *linear regression*) or just by using a ruler, and the line's value for $\Delta t = 0$ can be computed. This value is closer to the result of the real system than all simulation results needed to calculate it and it did not increase the computation time at all!.

5.4 Example

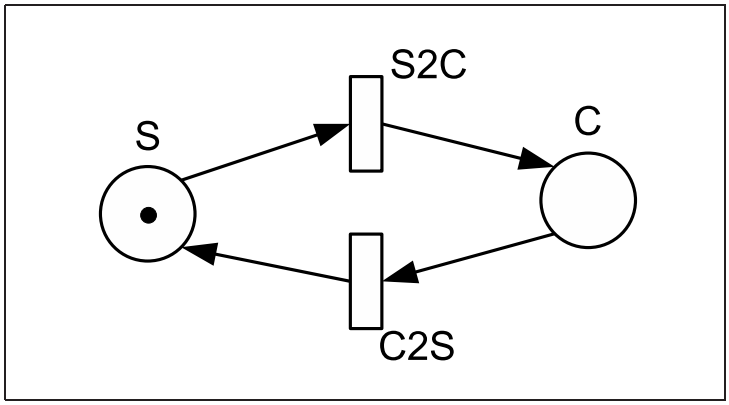

Figure 10: SPN describing a simple weather model

The figure above shows a picture of a SPN, which describes a very simple weather model. The transitions S2C (Sunny to Cloudy) and C2C (Cloudy to Sunny) are not exponentially distributed. We have decided to use Proxel Simulation and begin immediately to draw the tree of generated proxels for the discrete time steps t=0, 1δ, 2δ. We start with the initial proxel (m=S, τ= 0, $P_T = 1$) and than look which new proxels can be generated till the next time step and write them down. Leading to the following proxeltree:

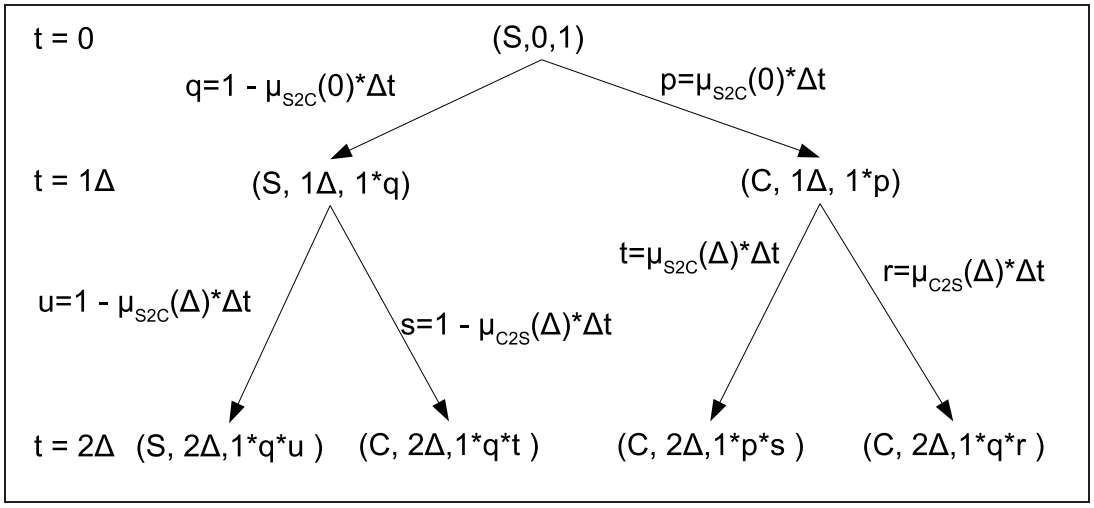

Figure 11: Proxeltree until t=2Δ

6 Phase Distributions

As seen in the section about proxels, one huge performance reduction occurs in proxel simulations when the actual state space gets very big. This happens - among others - when there are many active race age transitions and their *expected value* is much bigger than the time step size of the simulation. In this case, we potentially need proxels for each marking m and for each combination of ages of these transitions, making the simulation increasingly inefficient.
Markov chains do not have this problem, because with memoryless transitions there is no difference between race age and race enable transitions. Therefore research was conducted on whether it is

possible to simulate arbitrary distribution functions through the use of exponential distributions. Research showed that this is indeed possible.
Consider this simple petri net:

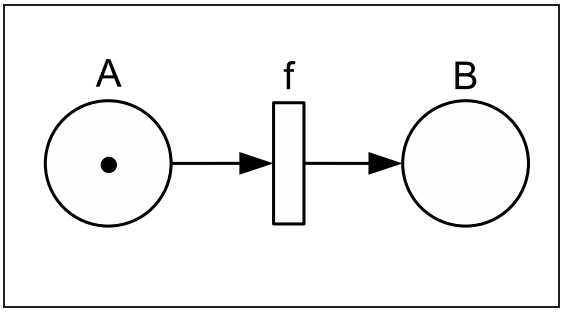

Figure 12: simple petri net

It has an extremely limited state space (the token is either in the place A or in place B), but it can only be converted to a CTMC if the transition f follows an exponential distribution.
However what can be done is to create a CTMC as one would if the transition were exponentially distributed and add additional states and exponential distributed transitions to approximate the desired distribution.

Types of Continuous Time Phase Distributions

Cox

Cox showed that an arbitrary distribution can be approximated arbitrarily exactly using a cox distribution.

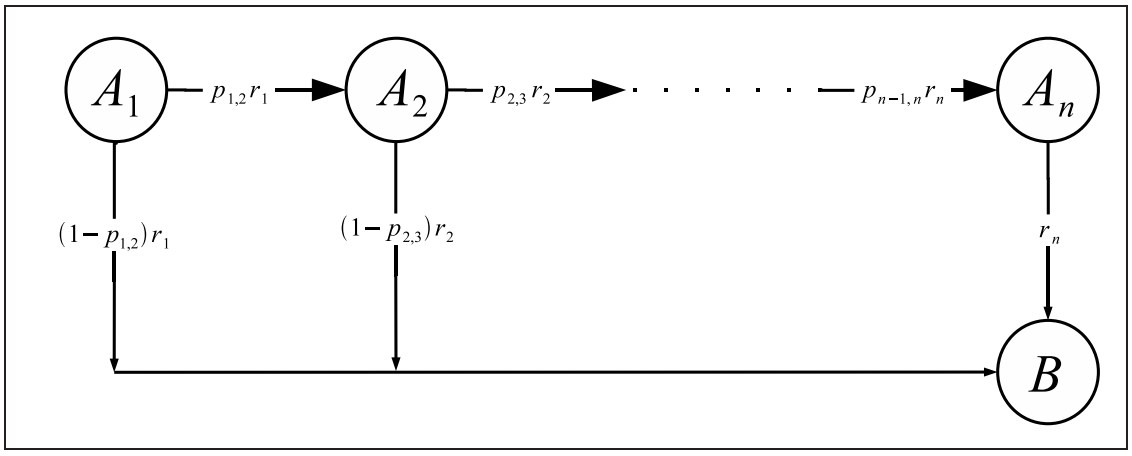

Figure 13: Cox phase distribution

Hypoexponential

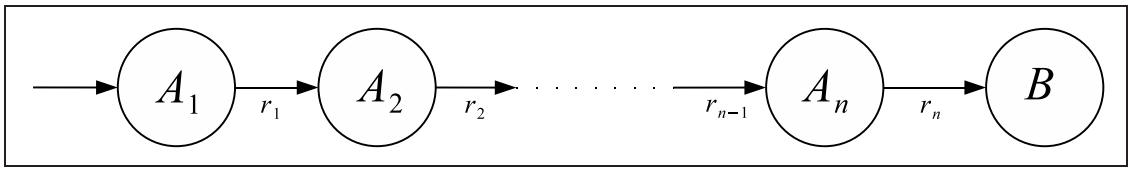

Figure 14: Hypoexponential phase distribution

Hyperexponential

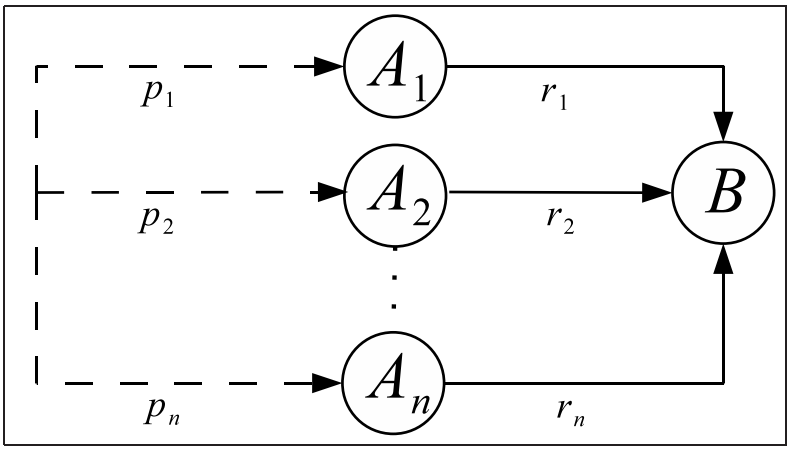

Figure 15: Hyperexponential phase distribution

types of discrete time phase distributions

Cox-like

Strict sequence

The strict sequence corresponds to the hyperexponential phase approximation method.

Age intensity

The age intensity corresponds to the proxel method.

6.1 algorithms for determining the parameters of phase distributions

To find determining parameters standard optimizing algorithm like are in use:

- gradient descension
- simplex method
- simulated annealing

6.2 disadvantages of phase distributions

Drawback of phase distribution is the needed preprocessing time for determining the parameters and the fixed cdf for Δt. Nevertheless the results are at least as bad as the proxel result and often better.

7 Rare Event Simulation

There are problems out there in the world, which have very low probabilities to happen (probabilities of 10^{-3} or even lower) like:

- prob. of the ruin of a bank
- prob. of winning the lottery jackpot
- prob. of an airplane accident
- prob. that your car breaks and the indicator light for this defect are nonfunctional at the same time

Nevertheless it's important to examine them. This costs a huge amount of time often even with Markov Chains so that other approaches like importance sampling or splitting are used. Simplified this methods are working after the following scheme:

1. Model M of rare event $\rightarrow M^*$
2. calculate $result^*$ (much faster)
3. "extract" real result out of $result^*$

Approaches

- importance sampling
- importance splitting

8 Hidden Markov Models

For this topic we want (until we find some more time) allude to the handouts (mindmap and "clean, shop, walk" example) which were given in the lecture or tutorial.

A Translations of technical terms

english term	***German equivalent***
random variable	Zufallsvariable
state space	Zustandsraum
pdf ((probability) density function)	(Wahrscheinlichkeits-) Dichtefunktion
cdf (cumulated distribution function)	Verteilungsfunktion
memorylessness	Gedächtnislosigkeit
change rate	Änderungsrate
outflow	Ausfluss
expected value	Erwartungswert
initial value problem	Anfangswertproblem
reachability graph	Erreichbarkeitsgraph
conditional probability	bedingte Wahrscheinlichkeit
linearly dependent	linear abhängig
identity matrix	Einheitsmatrix

References

[1] Bettina: `Bekki_doc_2.pdf` (found on the adm homepage of summer semester 2003)

[2] Horton, Graham: Introduction to simulation WS04/05 - Lecture 3

[3] Horton, Graham: Introduction to simulation WS04/05 - Lecture 5