Zusammenfassende Berechnungen

Grundproblem

Spielewelt soll aktiv wirken
Aber: Nicht die Berechnungsressourcen vorhanden, um immer die ganze Spielewelt zu simulieren

Mögliche Lösungen:

- Berechnungen ganz sparen, Kartengeneratortechnik, beispielsweise Noise, anstelle von Simulation verwenden: Wettergeschehen, Bevölkerungsverteilungen, Füllstand von Mülltonnen, ...
  Alle Methoden aus Abschnitt Landschaftsgenerator verwendbar
- Berechnungen vortäuschen: Verschiedene Techniken, um Berechnungsabkürzungen zu finden, die zwar nicht immer korrektes Ergebnis liefern, aber gut genug, um glaubwürdig zu sein.

Kartengeneratortechnik statt Simulation

Methoden aus Landschaftsgenerator, aber eine Dimension mehr, diese als Zeitachse verwenden

Wenn erzeugte Raum-Zeit-Karte stetig in Zeitachse:
Beständige und konsistente Änderung in der Zeit wahrnehmbar

Ergänzende Technik für Kreislaufsysteme (z.B. Wind)

- Diese Systeme sollten an jedem Knoten gleichviel hineinfließende wie hinausfließende Menge haben
- Erreichbar, indem man immer Features mit dieser Eigenschaft addiert
- Beispiel:
  Immer Kreisströme, die 4 Kästchen abdecken addieren

ActiveBlockModifier

Eingebaute Möglichkeit für automatische Blockänderung, ohne individuelle Timer, mit Möglichkeit Zeit nachzuholen (z.B. Feuerausbreitung und -löschen, Blumen verbreiten sich)

- `nodenames` welche Nodetypen sind betroffen
- `neighbors` welche Nachbarn sind notwenig zum Ausführen
- `interval` wie oft wird der ABM ausgeführt
- `chance` mit welcher Wahrscheinlichkeit
- `catch_up` Soll versucht werden, Zeit die in Abwesenheit vergangen ist, auszuholen?
- `action = function...` die Funktion, die als ABM aufgerufen werden soll

Zeit

Zusammenfassende Berechnungen berechnen die Effekte eines Zeitraums auf einmal

$\Rightarrow$ Wichtig zu wissen, wie viel Zeit vergangen ist

Es gibt unter anderem folgende Zeitfunktionen

- `minetest.get_gametime()`: Wie viele Sekunden sind seit Welterzeugung vergangen?
- `minetest.get_us_time()`: Zeit in Mikrosekunden; nicht persistent über Serverneustart hinweg
- `minetest.get_timeofday()`: Gibt die Zeit am Tag zurück
- `minetest.get_day_count()`: Wie viele Tage sind seit Welterzeugung vergangen?

Meist macht `gametime` das gewünschte

Zeitereignisse

Jeder Knoten kann ein gesetztes Zeitereignis haben

- `on_timer` legt fest, welcher Code ausgeführt werden soll
- Auch bei mehrfachem Starten läuft immer nur einer
- Priority-Queue verwaltet Timer
  $\Rightarrow$ langlaufende Timer verursachen fast keinen Berechnungsaufwand während sie laufen
- Jeder einzelne Timeraufruf geht relativ schnell
  Aber: Minetest hat sehr viele Knoten, wird insgesamt langsam
- Einzelne Ruckler, bei vielen Timern auf einmal
  $\Rightarrow$ Nach Möglichkeit immer etwas Zufall für Zeitpunkt des Timeraufrufs verwenden (auch optisch oft schöner)
- Timer werden auf nicht geladenen Blocks (weit weg von Spielern) nicht ausgeführt; ist die Zeit inzwischen vergangen aber gleich beim wieder Laden des Blocks

Zusammenfassen von Berechnungen

Nachteile

- Berechnungen aufheben und am Stück abarbeiten kann zu ungleichmäßigem Spielablauf führen, wenn dabei viel gerechnet werden muss
- Leerlaufzeit des Servers wird nicht ausgenützt, wenn Berechnungen aufgehoben werden

$\Rightarrow$ Berechnungen zusammenfassen lohnt sich aus Sicht der Effizienz nur, wenn sie dadurch deutlich schneller ausgeführt werden

Aber: Da entfernte Teile des Servers nicht simuliert werden, können sich Zusammenfassungen auch lohnen, um den Server lebending wirken zu lassen

Problemfall Ofen

Typischer Ablauf:

- Jede Sekunde wird ein wenig Brennstoff verbraucht
- Jede Sekunde wird der Verarbeitungszustand ein wenig hochgesetzt
- Jede Sekunde wird geschaut, ob der Gegenstand fertig gebacken wurde, ggf. ein fertiger Gegenstand hinzugefügt, ein Verbrauchsgegenstand entfernt

Probleme:

- Berechnung jede Sekunde auf dem Feld, wird langsam bei vielen Öfen
- Bei großer Entfernung zum Ofen ist dieser in der Zwischenzeit stehen geblieben: Maximal eine Sekunden gespart durch Wiederaufruf

Berechnungszusammenfassung Ofen

Eine Möglichkeit den Ofen zu beschleunigen

- Ofentimer seltener aufrufen, ggf. seltener bei großem Spielerabstand; bei offenem Ofeninventar wieder einmal pro Sekunde
- In der Berechnung schauen, wie viel Zeit seit der letzten vergangen ist t_{delay}
- Treibstoff reicht t_{fuel} =Brennstoffdauer·Brennstoffanzahl
- Ofengut reicht t_{educt} =Backdauer·Ofengutanzahl
- Platz reicht t_{space}=Backdauer·freier Platz
- Ofen kann $t := \min\{t_{\text{delay}}, t_{\text{fuel}}, t_{\text{educt}}, t_{\text{space}}\}$ Schritte laufen
- $t/$Brennstoffdauer viel Treibstoff verbrauchen, $t/$Backdauer viele Produkte backen

Mehr Berechnungszeit pro Zeitschritt (5-fach?), aber kann viel seltener aufgerufen werden, Ofen läuft auch ohne Rechenaufwand, wenn alle Spieler weit weg sind

Produktionswarteschlange

Allgemeinere Version des Ofens

- Verbrauchsliste, Produktionsliste; Berechnung, wie viel Zeit damit möglich ist
- Mehrere Produktionen hintereinander hängen, wenn eine fertig ist, Restzeit für nächste verwenden
- Bei sehr großen Warteschlangen kann es trotzdem wieder Lag geben: Anzahl der Berechnungen beschränken, letzte Berechnung davon nur teilweise ausführen: Spieler soll nicht bei jedem Nachschauen einen frisch angefangenen Arbeitsschritt sehen

Zustandsübergangssysteme

Zustandsübergangssysteme sind ein allgemeines Konzept mit vielfälltigen Anwendungen

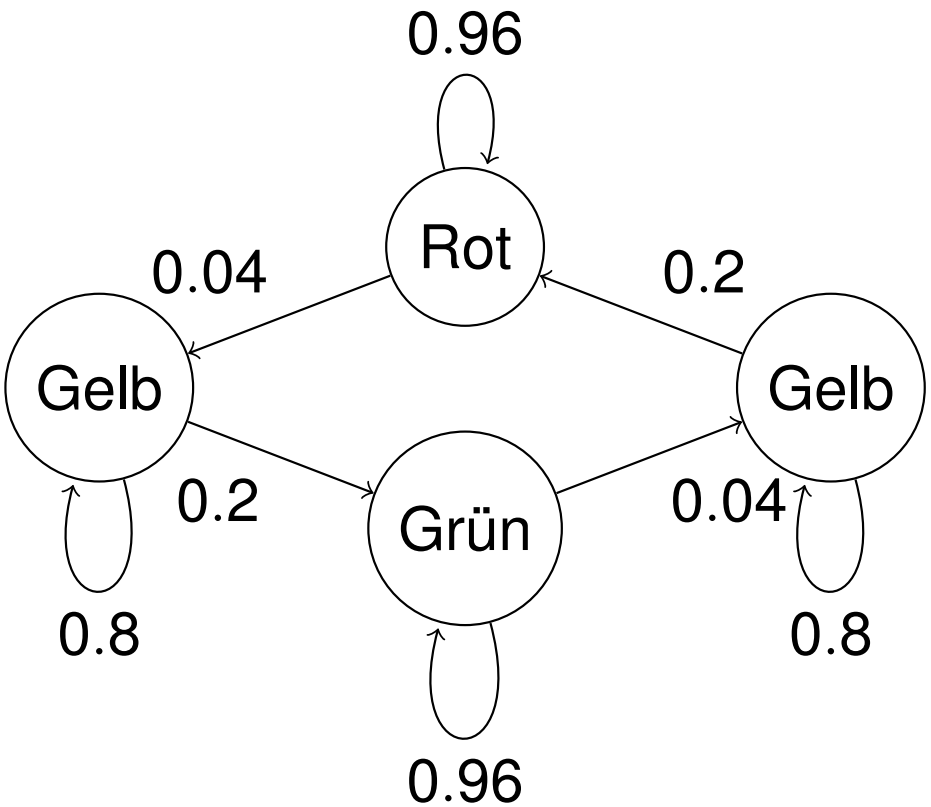
- Endliche Automaten
- Markovketten
- Zustandsbasierte Programmierung

Grundlegend: Ein Zustandsübergangssystem hat eine Menge von Zuständen Q, Beschriftungen der Zustände und Regeln, um von einem Zustand zu einem anderen zu kommen (=ein Graph mit beschrifteten Knoten und Kanten)

Wir verwenden Zustandsübergangssysteme zum Beschleunigen von Berechnungen

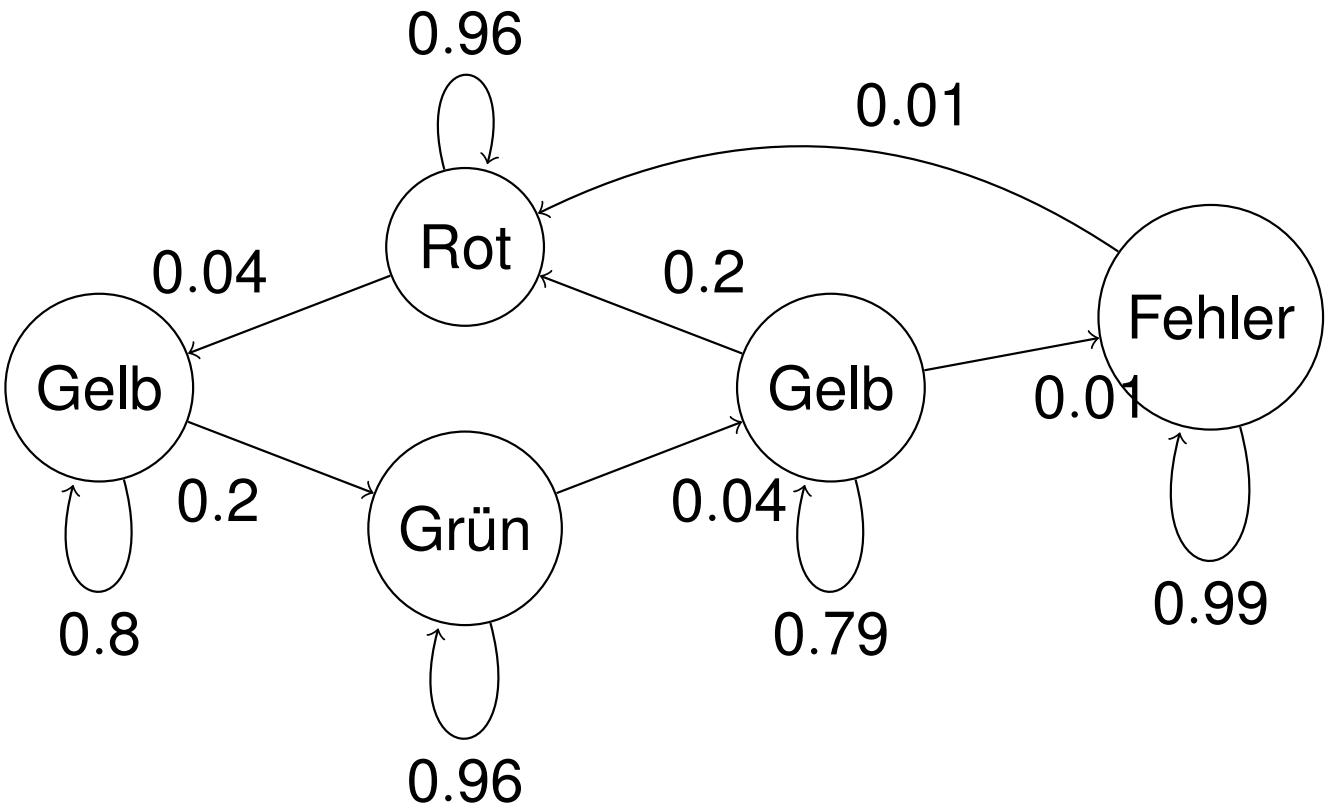
Markov-Ketten

Simulationsprobleme, die sich als Markov-Kette beschreiben lassen, können nahezu unbegrenz beschleunigt werden
Markov-Kette: Zustandsmenge Q, Effekt an jedem Zustand (verschiedene Zustände können den gleichen Effekt haben), Übergangswahrscheinlichkeiten zur Zustandsänderung in jedem Zeitschritt, z.B. jeder Sekunde
Bsp: Zufallsampel



Markov-Ketten

Simulationsprobleme, die sich als Markov-Kette beschreiben lassen, können nahezu unbegrenz beschleunigt werden
Markov-Kette: Zustandsmenge Q, Effekt an jedem Zustand (verschiedene Zustände können den gleichen Effekt haben), Übergangswahrscheinlichkeiten zur Zustandsänderung in jedem Zeitschritt, z.B. jeder Sekunde
Bsp: Zufallsampel, auch mit Fehlerzustand

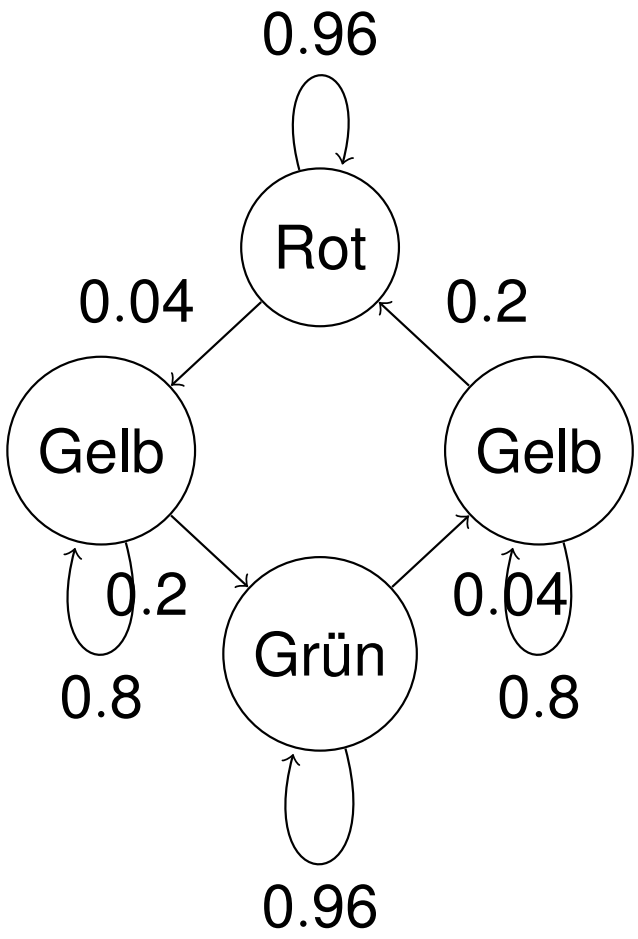


Markov-Ketten beschleunigen

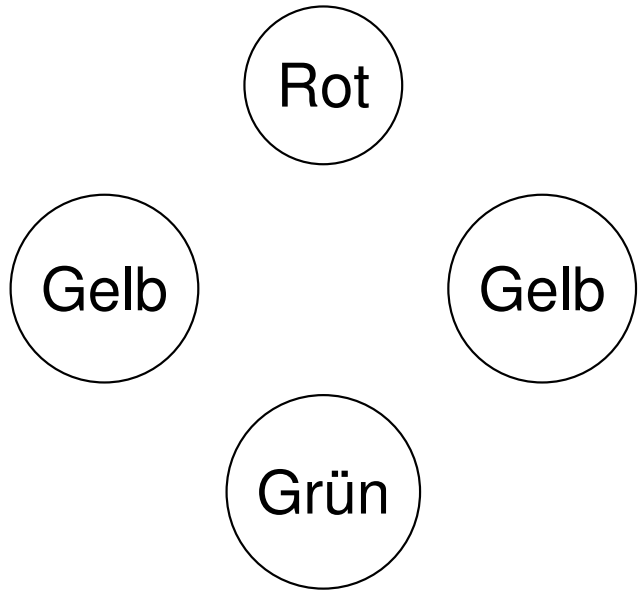
Wo kommt man in zwei Schritten hin ist auch Markov-Kette

$\Rightarrow$ Markovkette für doppelte Zeit berechenbar: Alle Möglichkeiten in zwei Schritten von einem Zustand zum anderen zu kommen zusammenzählen

1 Sekunde



2 Sekunden

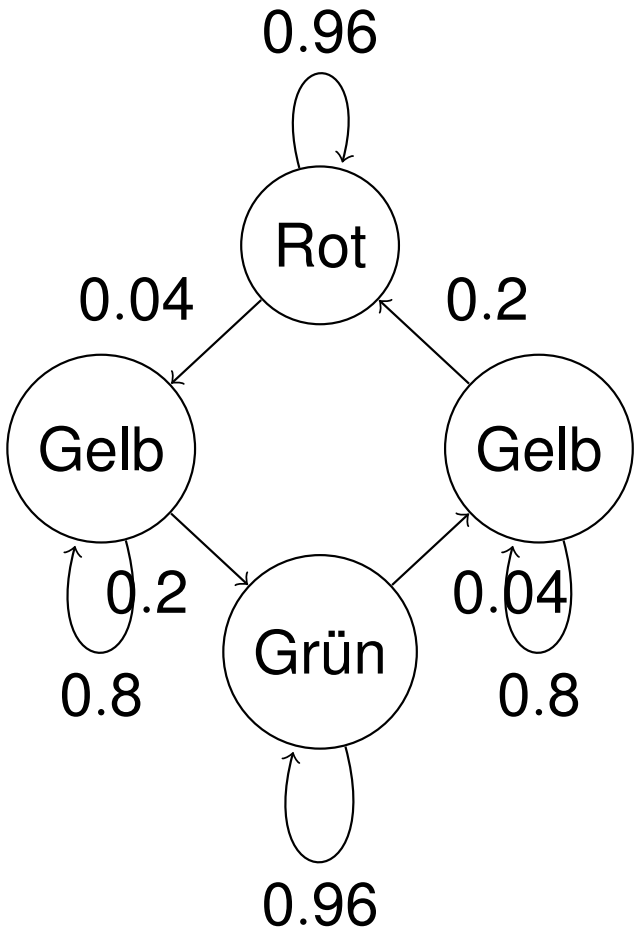


Markov-Ketten beschleunigen

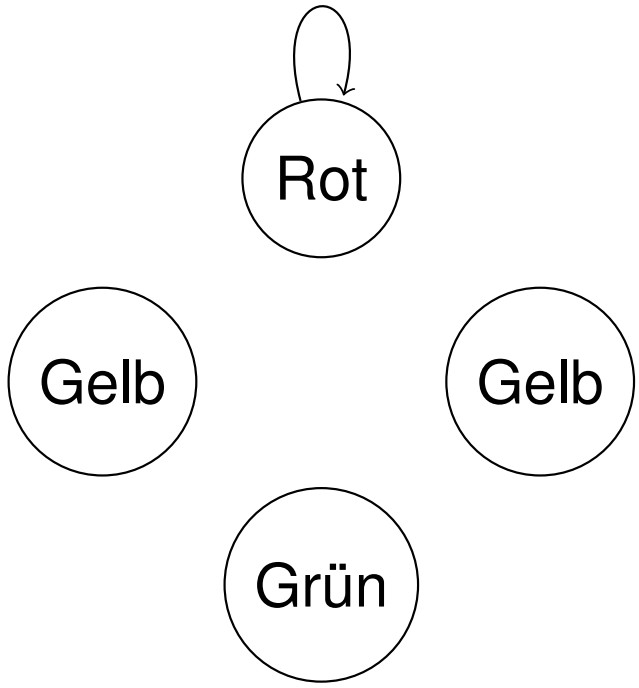
Wo kommt man in zwei Schritten hin ist auch Markov-Kette

$\Rightarrow$ Markovkette für doppelte Zeit berechenbar: Alle Möglichkeiten in zwei Schritten von einem Zustand zum anderen zu kommen zusammenzählen

1 Sekunde



2 Sekunden

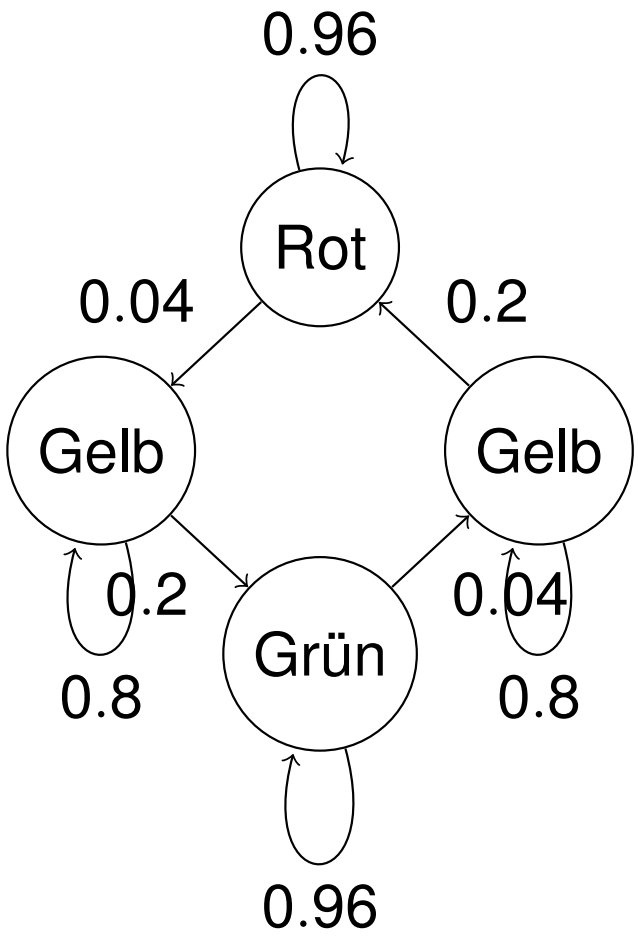


Markov-Ketten beschleunigen

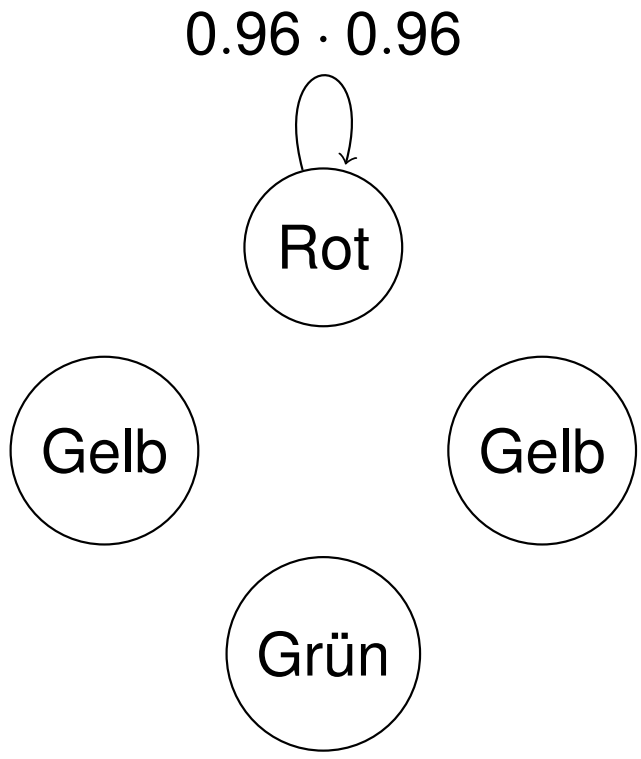
Wo kommt man in zwei Schritten hin ist auch Markov-Kette

$\Rightarrow$ Markovkette für doppelte Zeit berechenbar: Alle Möglichkeiten in zwei Schritten von einem Zustand zum anderen zu kommen zusammenzählen

1 Sekunde



2 Sekunden

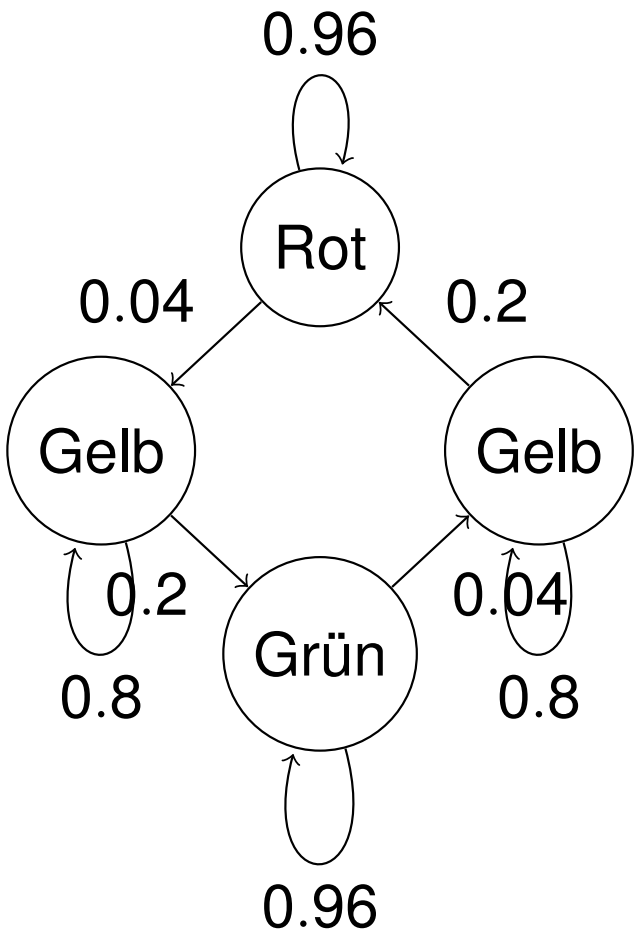


Markov-Ketten beschleunigen

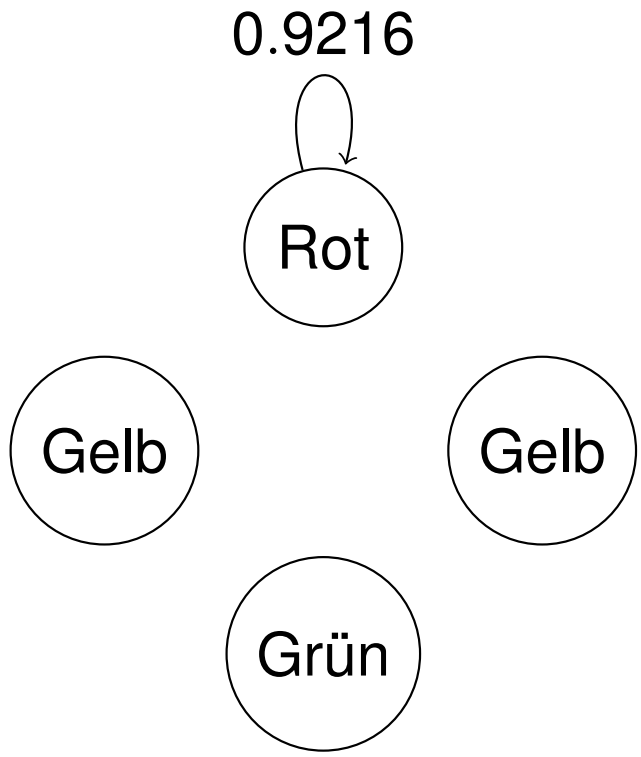
Wo kommt man in zwei Schritten hin ist auch Markov-Kette

$\Rightarrow$ Markovkette für doppelte Zeit berechenbar: Alle Möglichkeiten in zwei Schritten von einem Zustand zum anderen zu kommen zusammenzählen

1 Sekunde



2 Sekunden

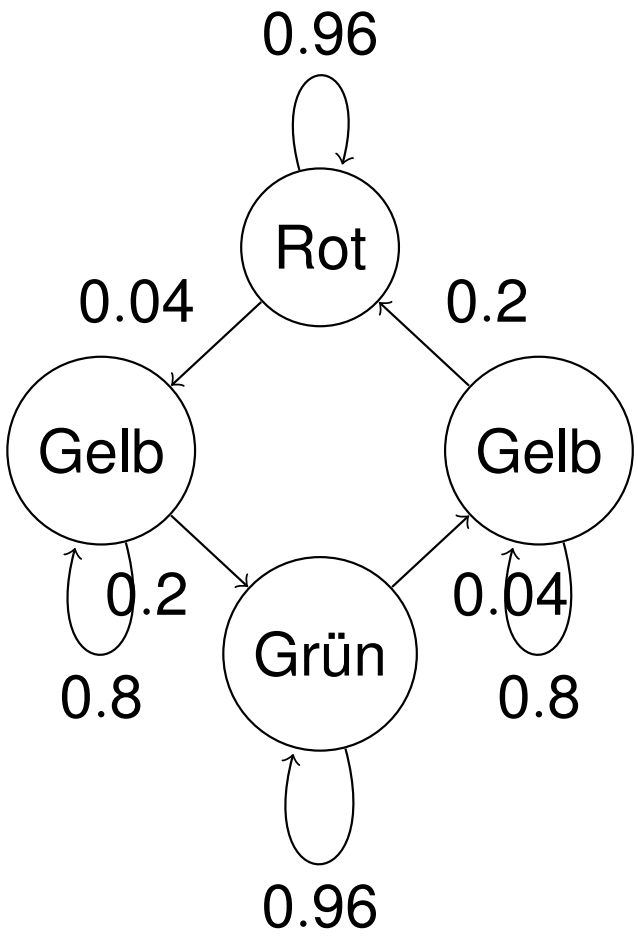


Markov-Ketten beschleunigen

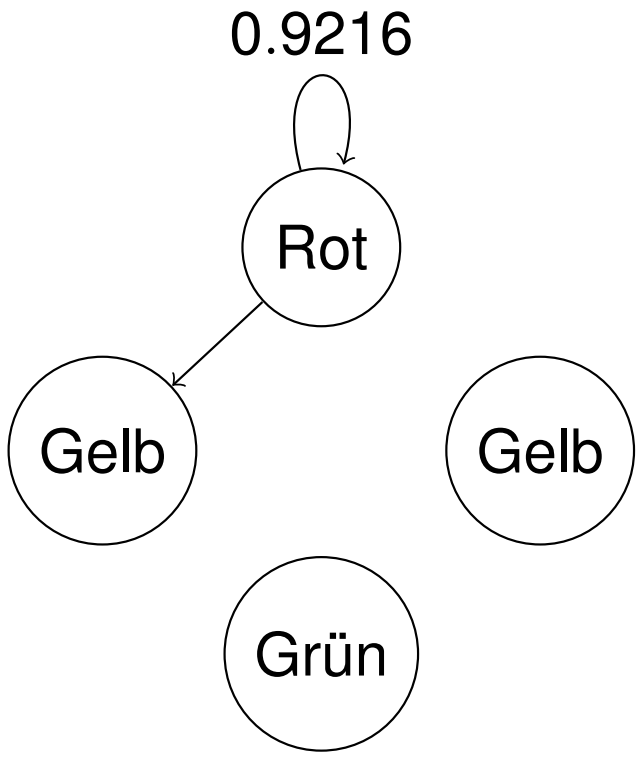
Wo kommt man in zwei Schritten hin ist auch Markov-Kette

$\Rightarrow$ Markovkette für doppelte Zeit berechenbar: Alle Möglichkeiten in zwei Schritten von einem Zustand zum anderen zu kommen zusammenzählen

1 Sekunde



2 Sekunden

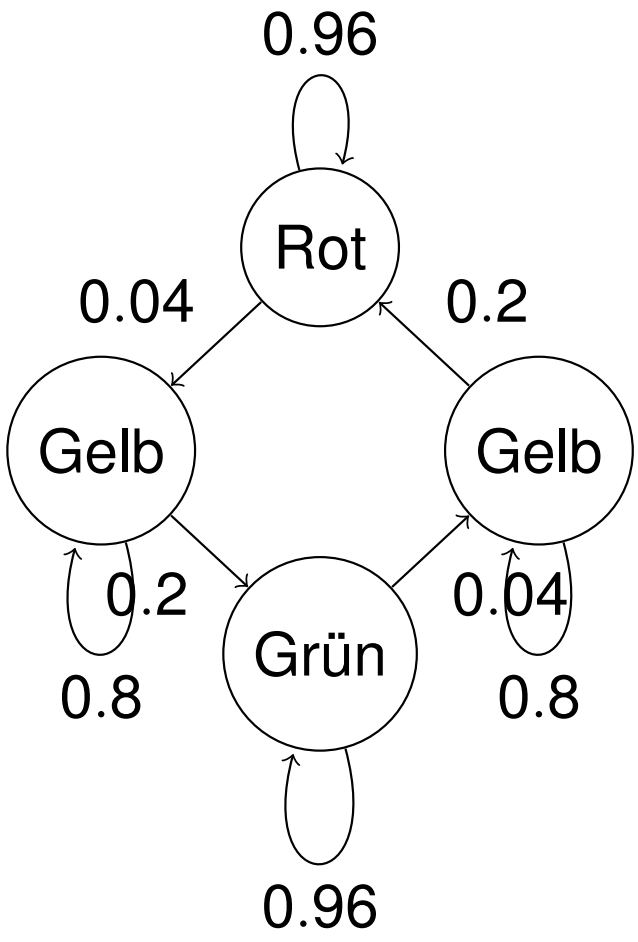


Markov-Ketten beschleunigen

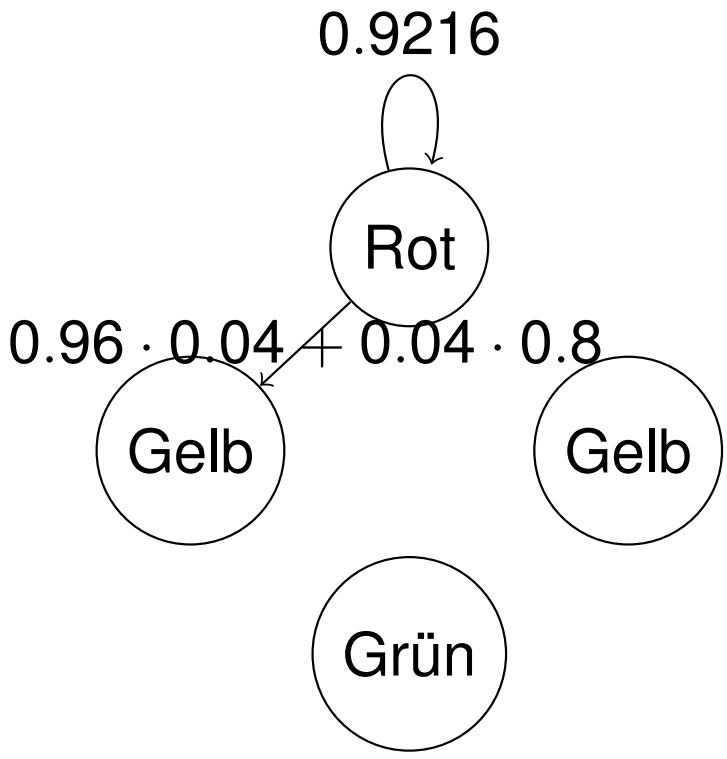
Wo kommt man in zwei Schritten hin ist auch Markov-Kette

$\Rightarrow$ Markovkette für doppelte Zeit berechenbar: Alle Möglichkeiten in zwei Schritten von einem Zustand zum anderen zu kommen zusammenzählen

1 Sekunde



2 Sekunden

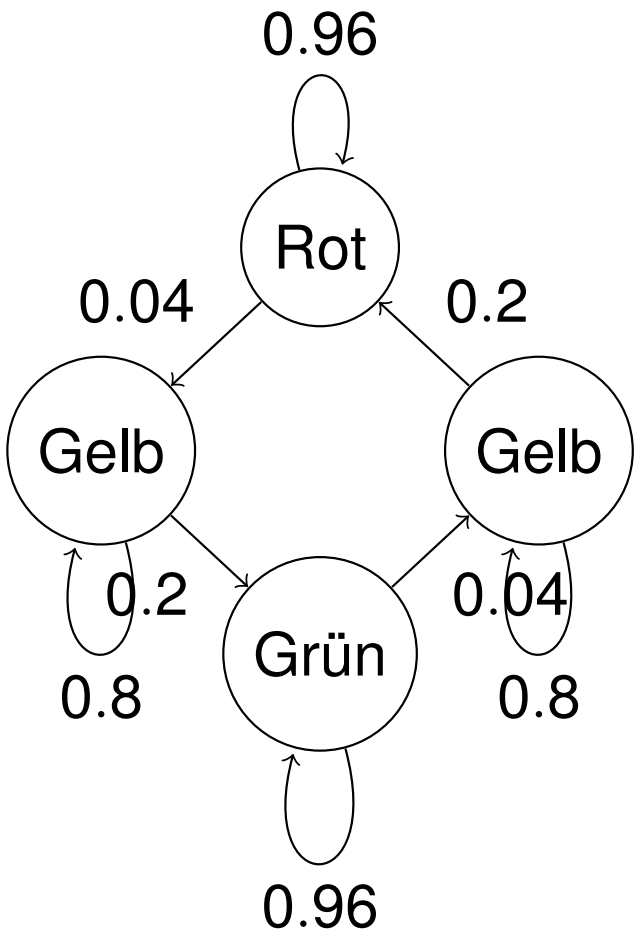


Markov-Ketten beschleunigen

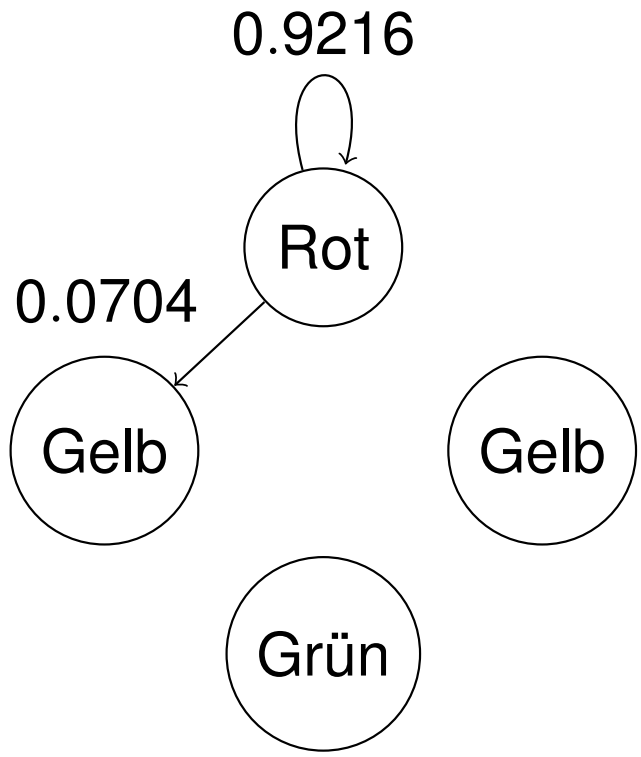
Wo kommt man in zwei Schritten hin ist auch Markov-Kette

$\Rightarrow$ Markovkette für doppelte Zeit berechenbar: Alle Möglichkeiten in zwei Schritten von einem Zustand zum anderen zu kommen zusammenzählen

1 Sekunde



2 Sekunden

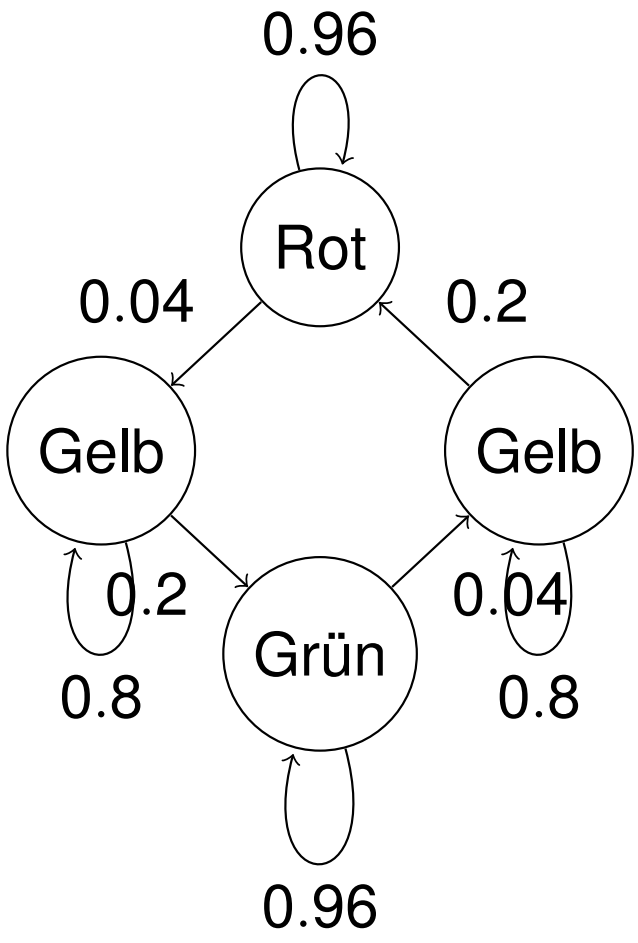


Markov-Ketten beschleunigen

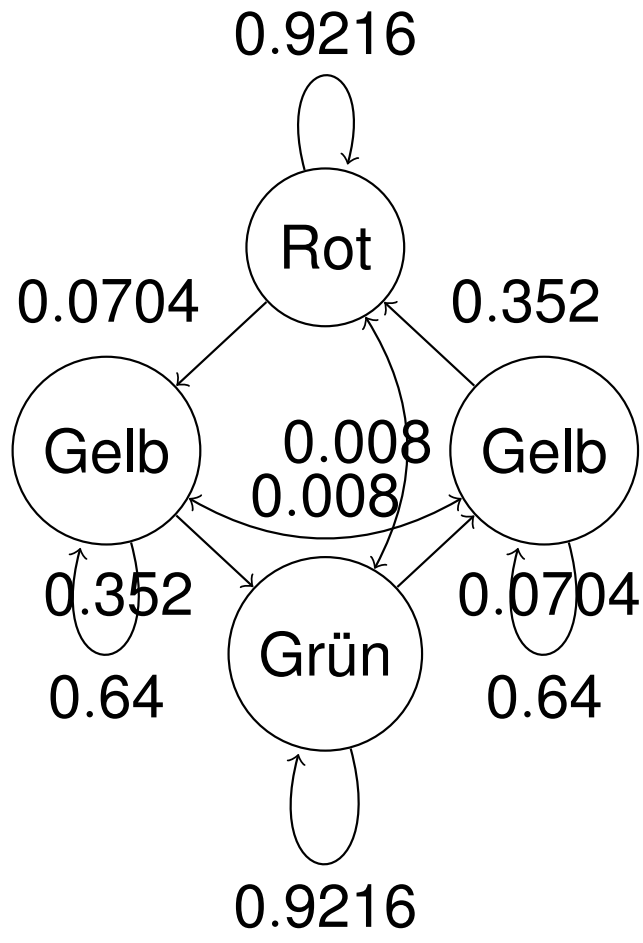
Wo kommt man in zwei Schritten hin ist auch Markov-Kette

$\Rightarrow$ Markovkette für doppelte Zeit berechenbar: Alle Möglichkeiten in zwei Schritten von einem Zustand zum anderen zu kommen zusammenzählen

1 Sekunde

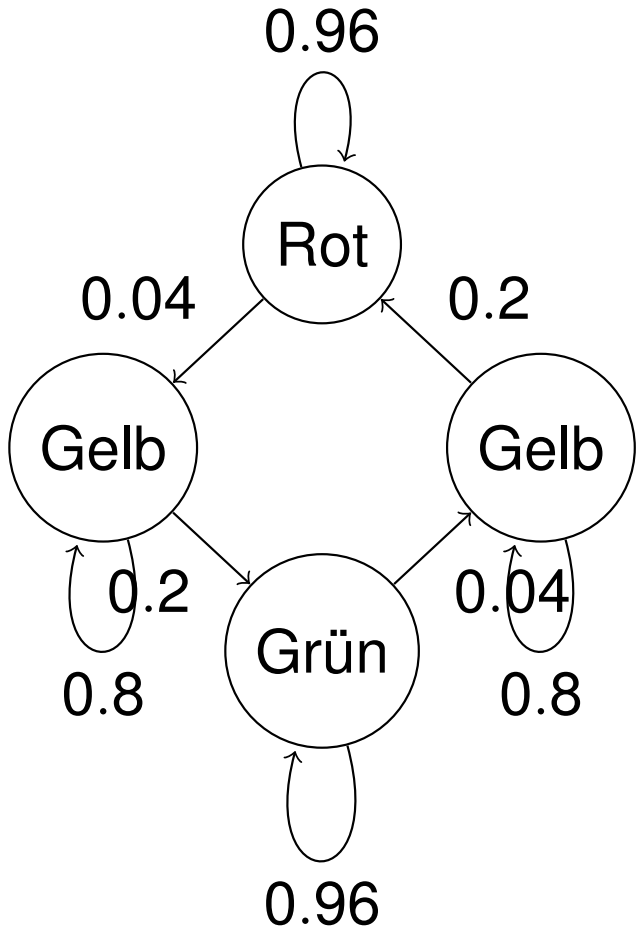


2 Sekunden



Markov-Ketten in Matrixdarstellung

Übergangswahrscheinlichkeiten als Matrix



	rot	gelb1	grün	gelb2
rot	0.96	0.04	0	0
gelb1	0	0.8	0.2	0
grün	0	0	0.96	0.04
gelb2	0.2	0	0	0.8

Markov-Ketten beschleunigen in Matrixdarstellung

Markov-Kette mit Übergangsmatrix A für t Zeitschritte

Übergangsmatrix für $2t$ Zeitschritte: $A \cdot A$

1 Sekunde	rot	gelb1	grün	gelb2
rot	0.96	0.04	0	0
gelb1	0	0.8	0.2	0
grün	0	0	0.96	0.04
gelb2	0.2	0	0	0.8

2 Sekunden	rot	gelb1	grün	gelb2
rot	0.9216	0.0704	0.008	0
gelb1	0	0.64	0.352	0.008
grün	0.008	0	0.9216	0.0704
gelb2	0.352	0.008	0	0.64

Markov-Ketten stärker beschleunigen

Wiederholte Quadrieren ergibt Markovketten für immer größere Zeitschritte

2 Sekunden	rot	gelb1	grün	gelb2
rot	0.9216	0.0704	0.008	0
gelb1	0	0.64	0.352	0.008
grün	0.008	0	0.9216	0.0704
gelb2	0.352	0.008	0	0.64

4 Sekunden	rot	gelb1	grün	gelb2
rot	0.8494	0.1099	0.0395	0.0011
gelb1	0.0056	0.4096	0.5496	0.0350
grün	0.0395	0.0011	0.8494	0.1099
gelb2	0.5496	0.0350	0.0056	0.4096

Markov-Ketten stärker beschleunigen

Wiederholte Quadrieren ergibt Markovketten für immer größere Zeitschritte

4 Sekunden	rot	gelb1	grün	gelb2
rot	0.8494	0.1099	0.0395	0.0011
gelb1	0.0056	0.4096	0.5496	0.0350
grün	0.0395	0.0011	0.8494	0.1099
gelb2	0.5496	0.0350	0.0056	0.4096

8 Sekunden	rot	gelb1	grün	gelb2
rot	0.7242	0.1385	0.1275	0.0096
gelb1	0.0480	0.1702	0.6925	0.0891
grün	0.1275	0.0096	0.7242	0.1385
gelb2	0.6925	0.0891	0.0480	0.1702

Markov-Ketten stärker beschleunigen

Wiederholte Quadrieren ergibt Markovketten für immer größere Zeitschritte

8 Sekunden	rot	gelb1	grün	gelb2
rot	0.7242	0.1385	0.1275	0.0096
gelb1	0.0480	0.1702	0.6925	0.0891
grün	0.1275	0.0096	0.7242	0.1385
gelb2	0.6925	0.0891	0.0480	0.1702

16 Sekunden	rot	gelb1	grün	gelb2
rot	0.5542	0.1259	0.2811	0.0386
gelb1	0.1930	0.0502	0.6299	0.1267
grün	0.2811	0.0386	0.5542	0.1259
gelb2	0.6299	0.1267	0.1930	0.0502

Markov-Ketten stärker beschleunigen

Wiederholte Quadrieren ergibt Markovketten für immer größere Zeitschritte

16 Sekunden	rot	gelb1	grün	gelb2
rot	0.5542	0.1259	0.2811	0.0386
gelb1	0.1930	0.0502	0.6299	0.1267
grün	0.2811	0.0386	0.5542	0.1259
gelb2	0.6299	0.1267	0.1930	0.0502

32 Sekunden	rot	gelb1	grün	gelb2
rot	0.4348	0.0919	0.3984	0.0747
gelb1	0.3736	0.0672	0.4595	0.0995
grün	0.3984	0.0747	0.4348	0.0919
gelb2	0.4595	0.0995	0.3736	0.0672

Markov-Ketten stärker beschleunigen

Wiederholte Quadrieren ergibt Markovketten für immer größere Zeitschritte

32 Sekunden	rot	gelb1	grün	gelb2
rot	0.4348	0.0919	0.3984	0.0747
gelb1	0.3736	0.0672	0.4595	0.0995
grün	0.3984	0.0747	0.4348	0.0919
gelb2	0.4595	0.0995	0.3736	0.0672

64 Sekunden	rot	gelb1	grün	gelb2
rot	0.4165	0.0833	0.4167	0.0832
gelb1	0.4164	0.0831	0.4168	0.0835
grün	0.4167	0.0832	0.4165	0.0833
gelb2	0.4168	0.0835	0.4164	0.0831

Markov-Ketten stärker beschleunigen

Wiederholte Quadrieren ergibt Markovketten für immer größere Zeitschritte

64 Sekunden	rot	gelb1	grün	gelb2
rot	0.4165	0.0833	0.4167	0.0832
gelb1	0.4164	0.0831	0.4168	0.0835
grün	0.4167	0.0832	0.4165	0.0833
gelb2	0.4168	0.0835	0.4164	0.0831

128 Sekunden	rot	gelb1	grün	gelb2
rot	0.4166	0.0833	0.4166	0.0833
gelb1	0.4166	0.0833	0.4166	0.0833
grün	0.4166	0.0833	0.4166	0.0833
gelb2	0.4166	0.0833	0.4166	0.0833

Markovketten, Numerische Stabilität

Jede Zeile in der Matrix zur Markovkette hat in der Summe 1.
Kann geringfügig durch Rundungsfehler abweichen.
$\Rightarrow$ Zeilen normalisieren (=durch ihre Summe teilen) erhöht die Genauigkeit

Markovketten, Schnelle allgemeine Berechnung

Durch Quadrierung alle Markovketten für 2^k viele Zeitschritte vorhanden.
Konkrete Zustandsänderung von t Zeitschritten:
Binärdarstellung von t bestimmen, entsprechende Markovketten anwenden.

Weitere Beschleunigung: Spieler merken sich nicht exakt wie viel Zeit vergangen ist, sondern nur ungefähr: Oberste 3 gesetzte Bits meist ausreichend: Im Mittel 2.8% Unterschied, maximal 12.5% Unterschied

Markovketten, bessere Beschleunigungszahlenfolgen

Effizientere Approximationen durch Zahlenfolgen der Form $e_0 = 1 \qquad e_{i+1} = [e_i \cdot c]$ mit $1 < c \leq 2$; [ ] steht für Runden, egal ob ab- oder aufgerundet, nur muss das nächste Element mindestens 1 größer sein

Um n zu approximieren, größte Zahl aus der Zahlenfolge, die kleinergleich ist, auswerten, Rest nach gleichem Schema auswerten

Bei k Auswertungen maximaler Unterschied: $(1 - 1/c)^k$; im Mittel kleinerer Unterschied

Beispiel, Zahl 123456789, $k = 2$

- $c = 2$, $67108864 + 33554432 = 100663296$, 18.5% Fehler
- $c = 1.5$, $90811045 + 26906977 = 117718022$, 4.6% Fehler
- $c = 1.2$, $107964794 + 14530766 = 122495560$, 0.78% Fehler
- $c = 1.1$, $119388179 + 3862115 = 123250294$, 0.17% Fehler

Markovketten für Beschleunigungsfolgen berechnen

Für kleine Zahlenwerte: iterierte Matrixmultiplikation berechnen

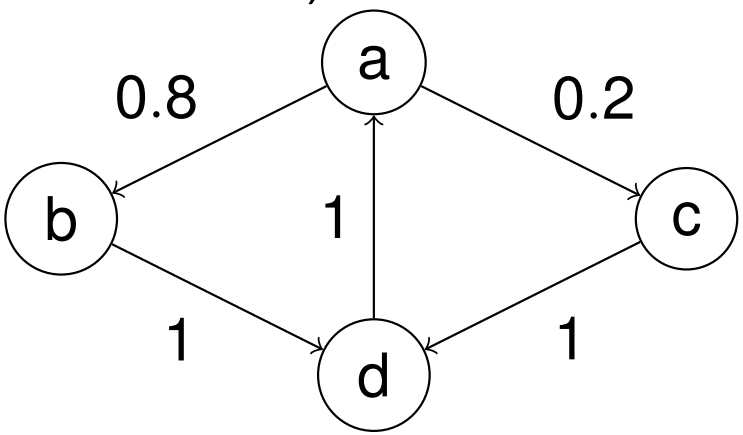
Für größere Zahlenwerte: Matrizen für geeignete kleinere Werte dieser Folge miteinander Multiplizieren

Beispiel: $c = 1.2$, Matrix für Markovkette für 12345 Zeitschritte berechnen
Approximationsalgorithmus für Auswertung verwenden, aber beliebig viele Schritte berechnen
$123456 = 105780 + 17086 + 537 + 52 + 1$, damit Matrizen für 105780, 17086, 537, 52, 1 miteinander multiplizieren

Markovketten, Wertstabilität

Werte stabilisieren sich oft nach häufigem Quadrieren
$\Rightarrow$ nur so weit quadrieren, bis sich die Werte stabilisiert haben, für größere Werte die letzte Matrix verwenden
Bis auf konkrete Gegenbeispiele: Analog für die anderen Konstruktionsfolgen

Beispiel für nichtstabilisierende Markov-Kette (beim Quadrieren)



Markovketten, Stabilität durch Zeitapproximation

Mehrere Markov-Ketten mit gleichem Zustandsraum können gewichtet zu einer zusammengefasst werden, indem alle Matrixeinträge gewichtet addiert werden

Markovkette C erzeugen, so dass Zustandsübergang mit Wahrscheinlichkeit 0.3 gemäß Kette A, mit 0.7 gemäß Kette B erfolgt: $C_{i,j} = 0.3 \cdot A_{i,j} + 0.7 \cdot B_{i,j}$

Neue Konstruktion der Markovketten in der Berechnungsfolge: Ab einer gewissen Anzahl vergangener Schritte wird mit kleiner Wahrscheinlichkeit ein Schritt ausgelassen oder zusätzlich gemacht

Dann stabilisieren sich die Markovketten in der Konstruktion immer

Markovketten, dynamische Erzeugung

Neben einmal festgelegten Markovketten können sie auch dynamisch erzeugt werden

- Explizit durch Spieler festlegbare Markovketten
- Indirekt durch Spieler festlegbare Markovketten, z.B. eine andere Beschreibungsform von Objektaktionen
- Durch Zusammenbau von Markovketten entstehende Markovketten:
  Mehrere Markovketten, die sich gegenseitig beeinflussen können zu einer einzigen zusammengesetzt werden, welche keinen äußeren Einflüssen unterworfen ist

Dynamische Markovketten, optimale Berechnung

Dynamisch erzeugte Markovketten haben andere optimalen Zusammenfassungen, als einmalig festgelegte

- Speicherplatz für sehr viele vorberechnete beschleunigte Markovketten spielt bei einmaligen keine große Rolle; langsam wachsende Zahlenfolge für Konstruktion gut
- Für einzelne Objekte erzeugte Ketten werden nicht so oft verwendet; Markovketten für 2-er-Potenzen-Zeitschritte vermutlich optimal, Mehraufwand der langsam wachsenden Berechnungen lohnt sich kaum

Einflussabhängige Markovketten

Manches lässt sich gut durch eine Markovkette in Abhängigkeit von äußeren Einflüssen beschreiben (Pflanzenwachstum nur tagsüber; Maschine, die ausgeschaltet werden kann; . . .)

Für jeden Einflusszustand eine Markovkette, Einflussdauern merken, separat rechnen

Aber: Bei häufiger Einflussänderung wieder das Grundproblem, dass zu lang gerechnet werden würde (wenn auch schon stark abgeschwächt)

Verschiedene Markovketten auf dem gleichen Zustandsraum in Abhängigkeit von einer Einflussgröße nennt man auch probabilistischen Automaten

Einflussabhängige Markovketten, einheitliche Berechnung

Ansätze trotz unterschiedlicher Einflüsse einheitlich zu rechnen

- periodische Einflüsse (wie Sonnenstand):
  Eine Markovkette für einen Periodendauer (ganzer Tag bei Sonnenstand) bestimmen: Hintereinanderschalten der Stunden durch Multiplikation.
  Von letztem Zustand bis Mitternacht rechnen.
  Anzahl vergangener Tage auf einmal durchrechnen.
  Ergebnisse seit Mitternacht durchrechnen.
- aperiodische Einflüsse:
  Von hinten zurückrechnen, immer Matrix für zuletzt anzuwendende Markovkette aufmultiplizieren. In vielen Fällen konvergieren die Zeilen gegeneinander
  $\Rightarrow$ Wahrscheinlichkeit irgendwann unabhängig von sehr alten Zuständen, dann kein Weiterrechnen notwendig

Kombinierte Markovketten

Hängt eine Markovkette vom Zustand einer anderen Markovkette ab, so kann man aus den beiden Markovketten A, B eine gemeinsame erzeugen (sogenannte Produktkonstruktion):

Für jeden Zustand p aus A und q aus B erzeugt man einen Zustand (p, q), Übergänge entsprechend der Originalketten.

$\Rightarrow$ Damit lassen sich voneinander abhängende Markovketten ebenfalls beschleunigen

Aber: Zustandszahl ist Produkt der Zustände der Basisketten
$\Rightarrow$ bei veränderlicher Anzahl an Eingabeketten biszu exponentiell groß

Kombinierte Markovketten, approximative Berechnung

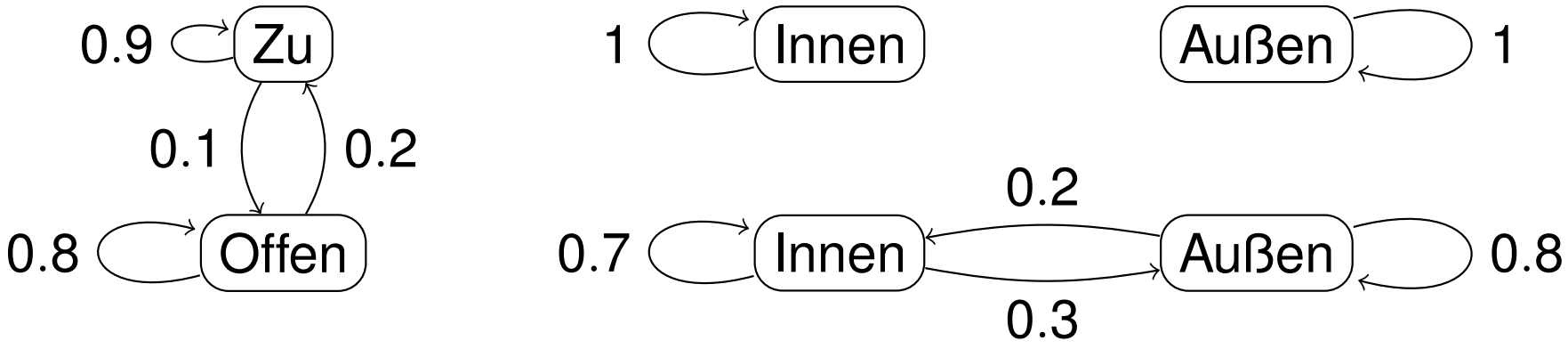
Hängen viele Markovketten von einer zentralen Markovkette ab, aber nicht umgekehrt, ist effiziente approximative Berechnung möglich:

Zentrale Markovkette vorausberechnen führt zu Zustand q; für jede andere aus Produktkonstruktion Zustand ablesen: Zeile passend zu dem Originalzustand nehmen; unter den Einträgen mit Zustand q im Tupel einen mit gewichteter Wahrscheinlichkeit nehmen

Es gibt Markovketten, bei denen diese Berechnung zu ungenauen Ergebnissen führt, in vielen Fällen kommt aber tatsächlich das gleiche heraus wie bei durchgehender Simulation

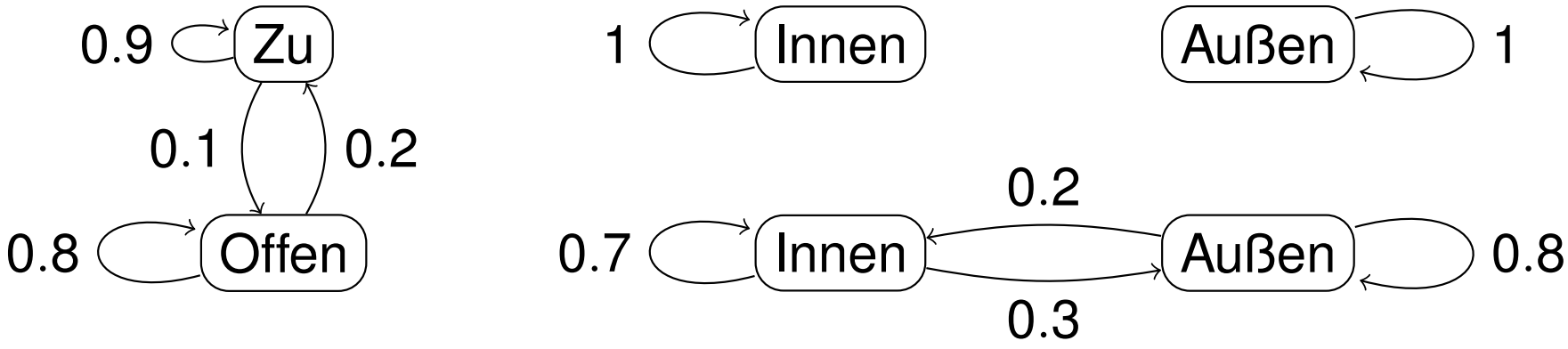
Beispiel Kombinierte Markovketten

Stalltüre geht auf oder zu, Hase kann innen oder außen sein, aber nur wechseln, wenn die Türe offen ist

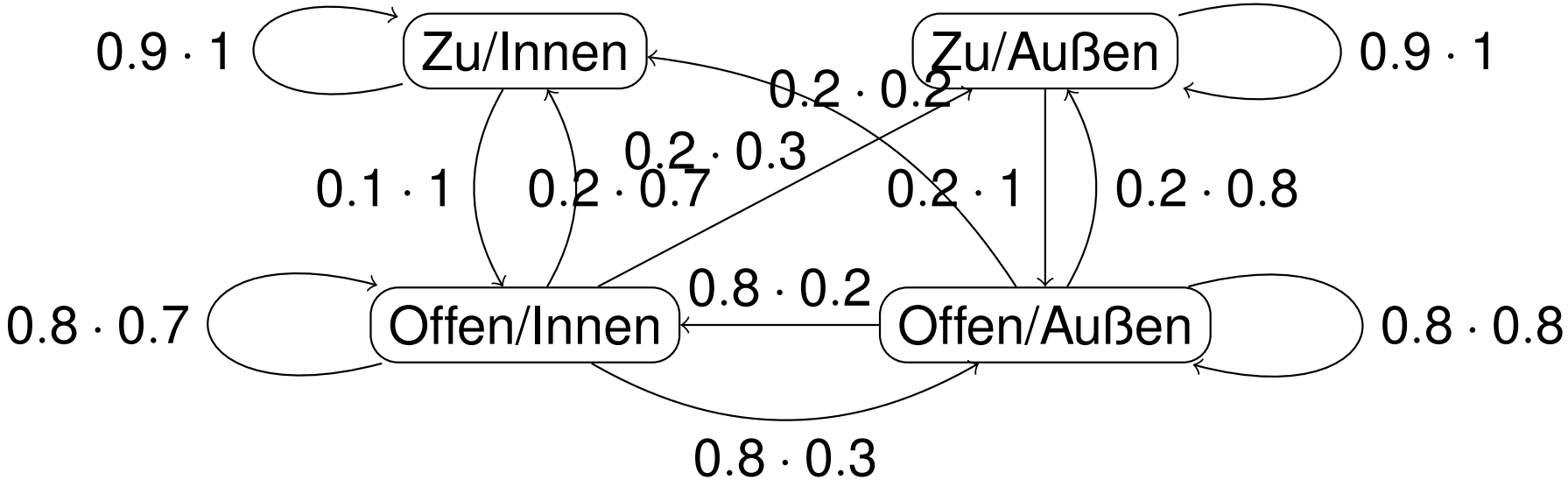


Beispiel Kombinierte Markovketten

Stalltüre geht auf oder zu, Hase kann innen oder außen sein, aber nur wechseln, wenn die Türe offen ist

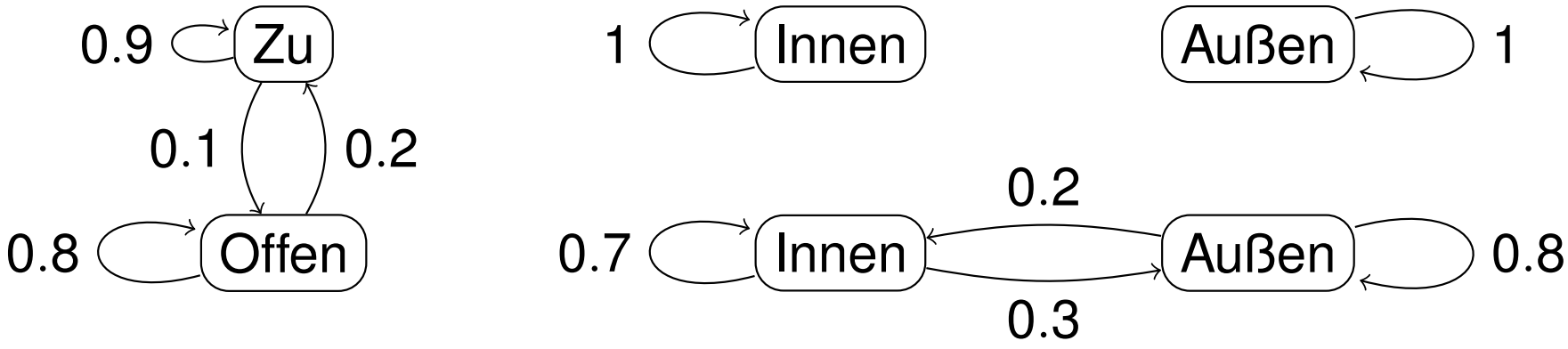


Als kombinierte Markovkette

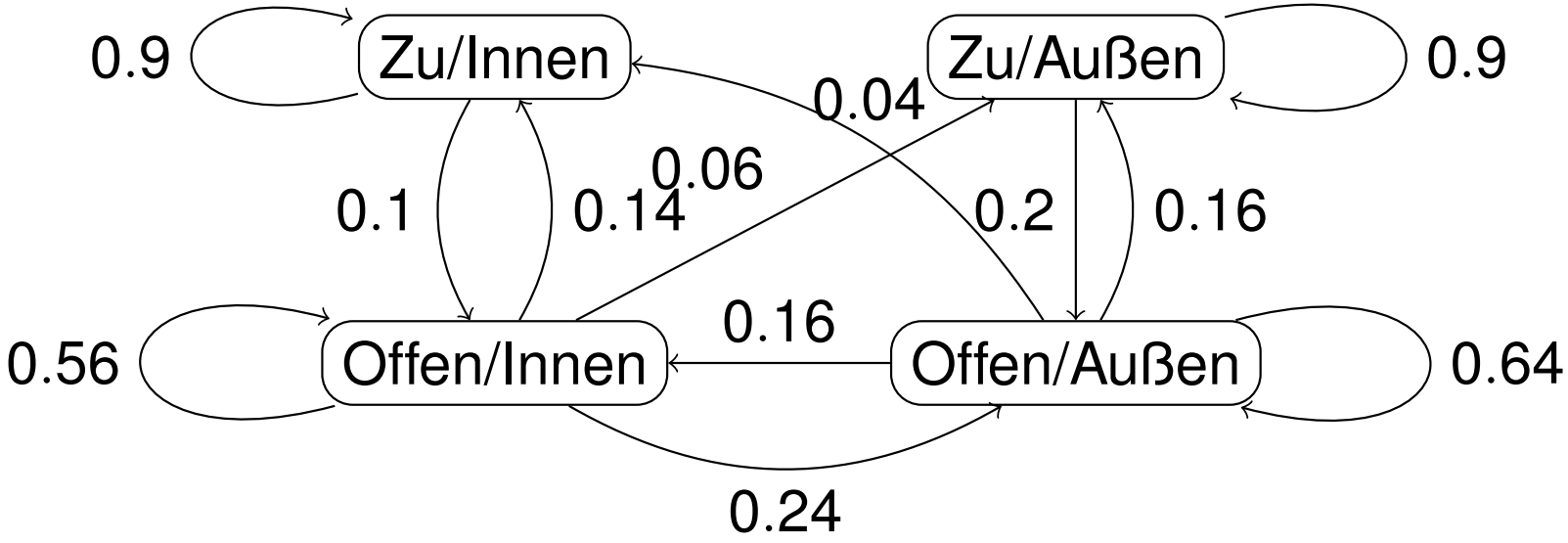


Beispiel Kombinierte Markovketten

Stalltüre geht auf oder zu, Hase kann innen oder außen sein, aber nur wechseln, wenn die Türe offen ist

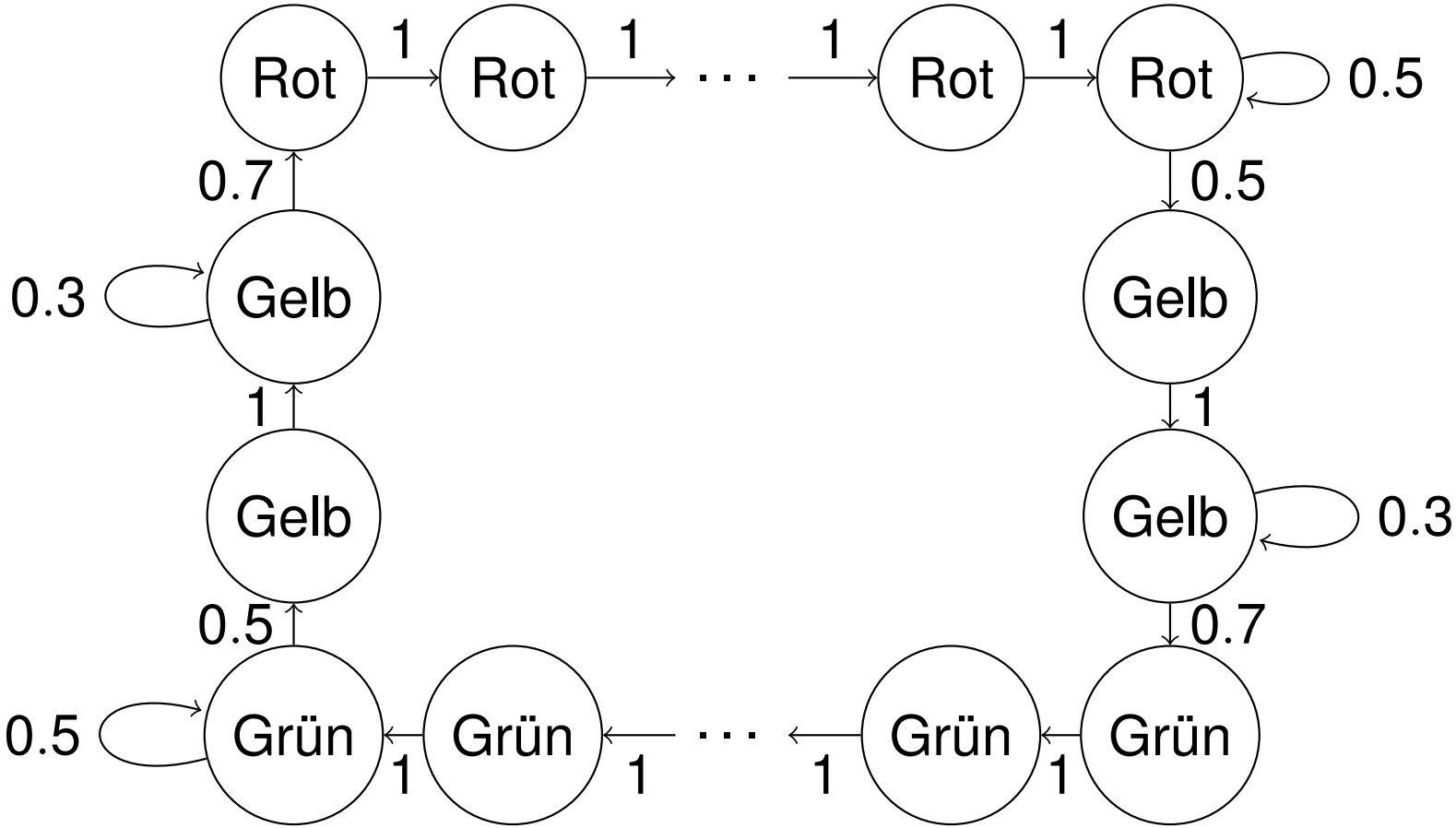


Als kombinierte Markovkette



Markovketten, Zustandsketten

Manche Situationen lassen sich gut mit einer langen Kette an Zuständen beschreiben. Beispiel: Genauere Zufallsampel



Problem: Viele Zustände notwendig

Lösungen:

Mehrere Markovketten, periodisch verwendet, manche selten

Markovketten mit Ressourcen

Markovketten, periodische Alternierung

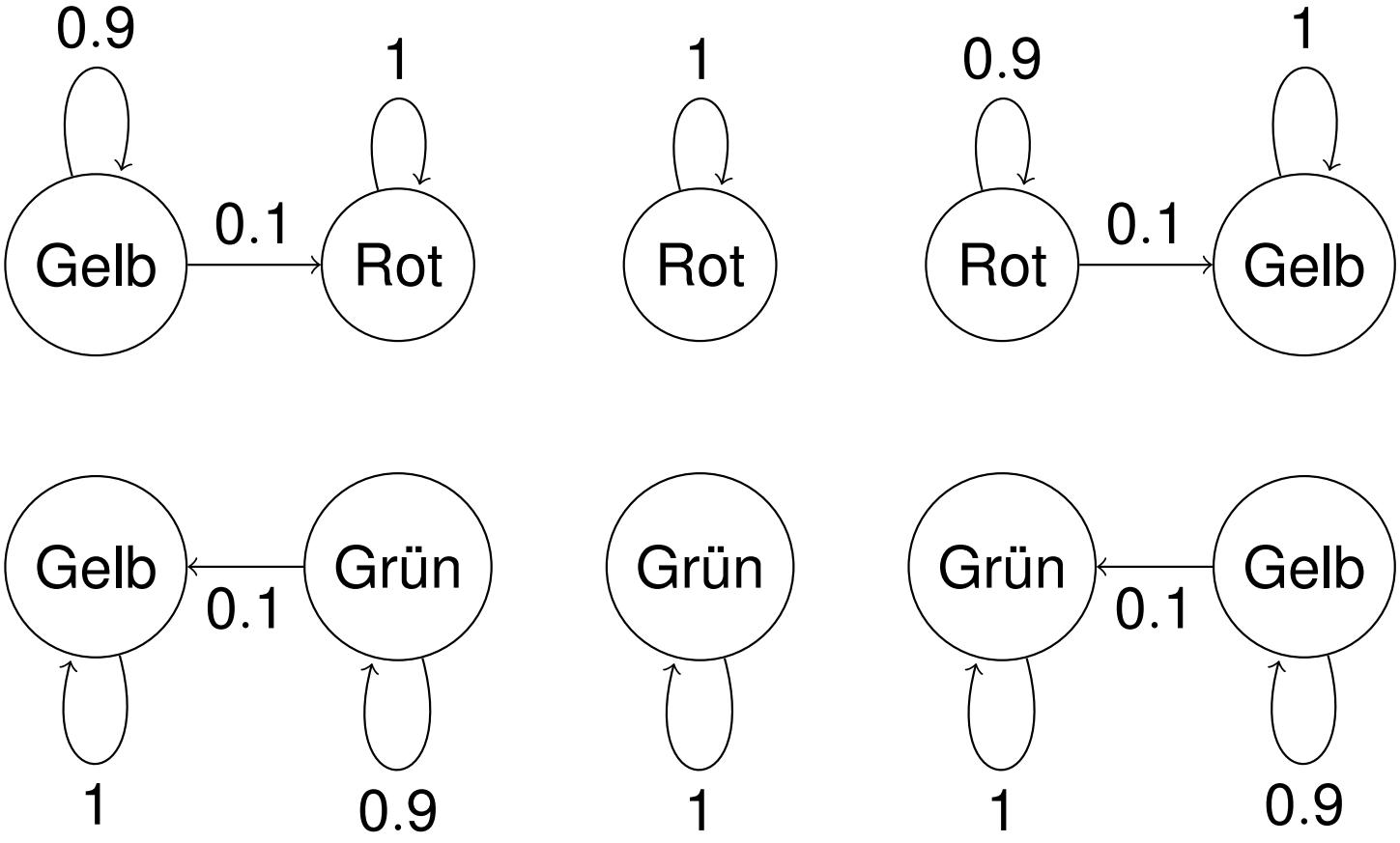
Werden mehrere Markovketten in einem periodischen Muster verwendet, manche davon seltener, so kann mehr vergangene Zeit mit weniger Zuständen beschrieben werden

Techniken zum einheitlichen Behandeln eines periodischen Musters legen diese beim Beschleunigen zu einer einzigen zusammen

Markovketten, periodische Alternierung, Beispiel

Zwei Markovketten, *A*, *B* für Zufallsampel. Periodische Wiederholung von: *AAAAAAAB*

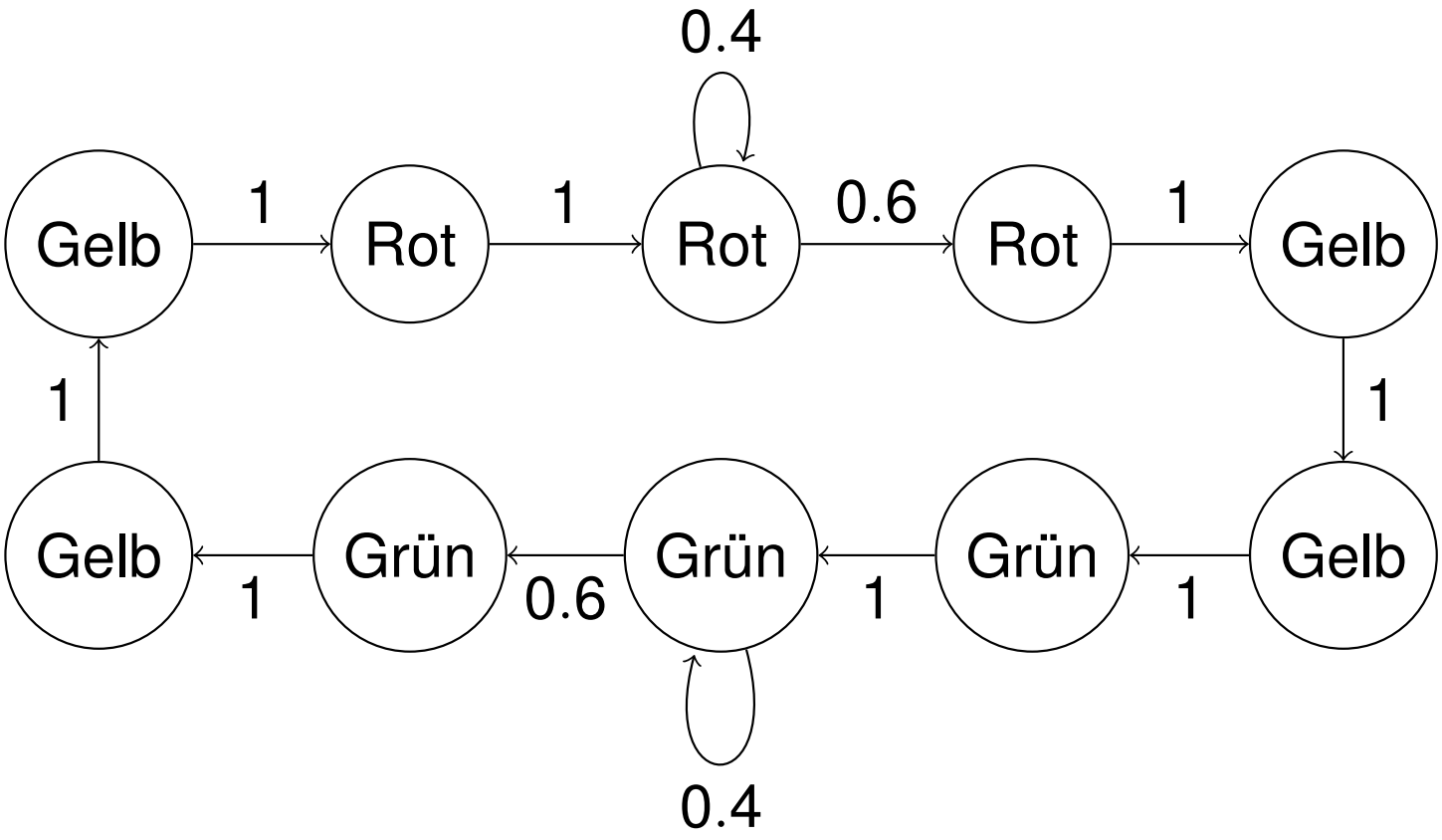
Markovkette *A*



Durch Änderung der Periode mehr Zeitbedarf notwendig, ohne mehr Zustände zu brauchen

Markovketten, periodische Alternierung, Beispiel

Zwei Markovketten, *A*, *B* für Zufallsampel. Periodische Wiederholung von: *AAAAAAAB*
Markovkette *B*



Durch Änderung der Periode mehr Zeitbedarf notwendig, ohne mehr Zustände zu brauchen

Beschreibungsstärke Markovkette

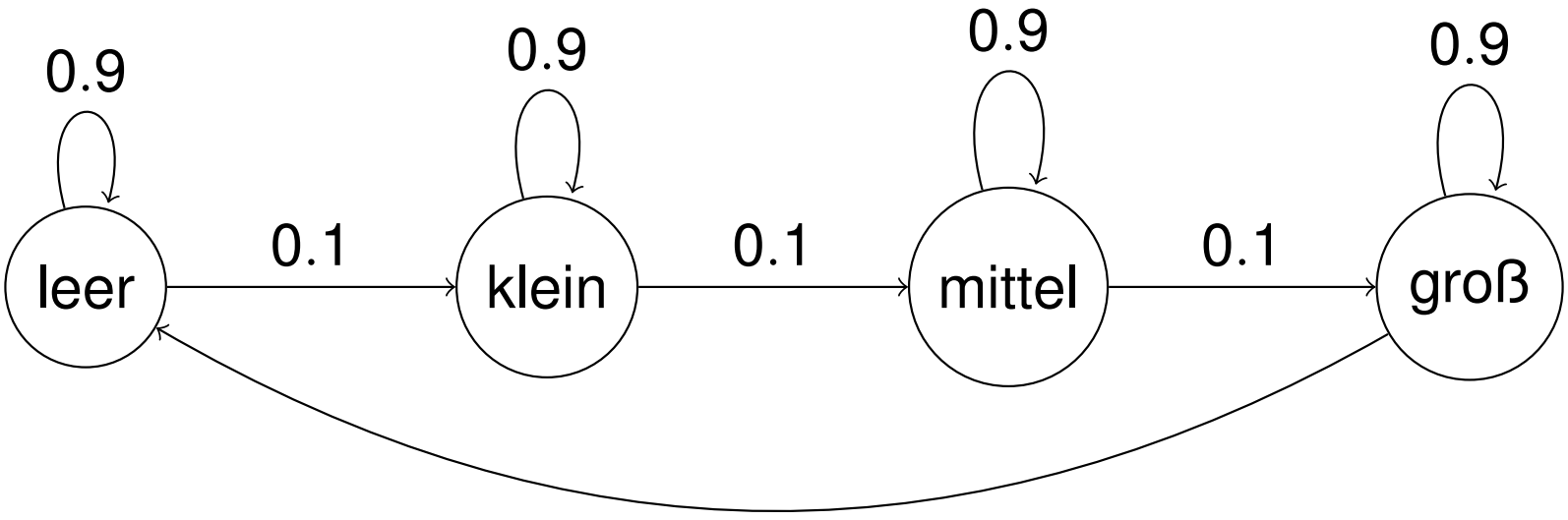
Viele Zusammenhänge lassen sich gut als Markovkette beschreiben.

Im Grundmodell aber keine Möglichkeit beliebig wiederholte Produktionsprozesse zu beschreiben; zudem viele Zustände bei geforderten Mindestzeiten

Darum: Erweitertes Modell mit Ressourcenzählern

Markovketten mit Ressourcenzähler

Als Effekt an Kanten: Liste möglicher hergestellter Ressourcen
Beispiel: Wachsende Pflanze, erzeugt 1 Blatt, wenn sie ausgewachsen ist



Hintereinanderausführung zweier Markovketten: Nicht nur Übergansmatrix multiplizieren, auch Liste möglicher Produktionen sammeln, inklusive Wahrscheinlichkeiten dafür

Markovketten mit Ressourcenzähler, Produktionszusammenfassung

Problem: Liste möglicher Produktionen wird sehr groß!

Darum: Sobald Liste zu lang wird (guter Wert ist ca. 10), nur noch Erwartungswert und Standardabweichung speichern. Wenn Produktionsergebnis gefragt wird: Gaußverteilung mit diesen Werten simulieren, dann runden

Probleme:

- Produktionszusammenhänge (nach jedem x wird ein y hergestellt) mehrerer Produkte werden nicht korrekt erfasst
- Werden Produkte in Gruppen hergestellt, können auch Zwischenwerte dadurch produziert werden

Markovketten mit Ressourcenzähler, Produktionszusammenfassung

Problem: Liste möglicher Produktionen wird sehr groß!

Darum: Sobald Liste zu lang wird (guter Wert ist ca. 10), nur noch Erwartungswert und Standardabweichung speichern. Wenn Produktionsergebnis gefragt wird: Gaußverteilung mit diesen Werten simulieren, dann runden

Probleme:

- Produktionszusammenhänge (nach jedem x wird ein y hergestellt) mehrerer Produkte werden nicht korrekt erfasst
  $\Rightarrow$ Nur Markovketten verwenden, die keine Produktionszusammenhänge haben
- Werden Produkte in Gruppen hergestellt, können auch Zwischenwerte dadurch produziert werden

Markovketten mit Ressourcenzähler, Produktionszusammenfassung

Problem: Liste möglicher Produktionen wird sehr groß!

Darum: Sobald Liste zu lang wird (guter Wert ist ca. 10), nur noch Erwartungswert und Standardabweichung speichern. Wenn Produktionsergebnis gefragt wird: Gaußverteilung mit diesen Werten simulieren, dann runden

Probleme:

- Produktionszusammenhänge (nach jedem x wird ein y hergestellt) mehrerer Produkte werden nicht korrekt erfasst
  $\Rightarrow$ Nur Markovketten verwenden, die keine Produktionszusammenhänge haben
- Werden Produkte in Gruppen hergestellt, können auch Zwischenwerte dadurch produziert werden
  $\Rightarrow$ Produktion skalieren, so dass immer 1 auf einmal hergestellt wird, hinterher zurückskalieren

Markovketten: Produktzusammenhänge eliminieren

Durch Kombination von Effekten können Abhängigkeiten eliminiert werden, damit Beschleunigung verwendbar wird

Beispiel: Es sollten immer 2 Kohle, 1 Eisenerz verbraucht werden und 1 Eisen hergestellt werden

- 3 Ressourcenzähler: Problem, unabhängige Zufallszahlen können für unpassende Ressorcenmengen sorgen; sind 11 Produktionszyklen bei eine Standardabweichung von 2 vergangen, so könnten 11 Eisenerz, 19 Kohle verbraucht und 14 Eisen hergestellt worden sein
- Besser: 1 Ressourcenzähler mit der Bedeutung „2 Kohle, 1 Eisenerz verbraucht werden und 1 Eisen hergestellt“ verwenden
  $\Rightarrow$ Werte passen immer zusammen

Markovketten: Ressourcenverbrauch

Bei Zählern ohne Werteeinschränkung: Eine Markovkette kann verwendet werden, kann Zählerwerte beliebig verändern

Werteeinschränkungen: Beschränktes Lager, Rohstoffverbrauch

Mehrere Markovketten: V für Zähler kann verändert werden (noch Rohstoffe da, ...), F für Zähler fertig.

- V anwenden, solange Veränderung möglich, F, wenn Veränderung nicht mehr möglich
- Ändert V den Zähler weit mehr als möglich: V stattdessen für einen kürzeren Zeitraum anwenden, Rest mit F abdecken
- Ändert V den Zähler ein wenig mehr als möglich: Zähler wird nur um maximalen Wert geändert; Messen, wie viel Zeit pro Veränderung verbraucht wurde, dann für geschätzte Zeit der überflüssigen Zähleränderungen noch F ausführen

Markovketten: Zeitsimulation durch Ressourcen

Zeitverbrauch als Ressource modellieren; es kann nur so viel Zeit verbraucht werden, wie vergangen ist

Vorteil: Beliebig große Zeitdauern können mit fester Zustandszahl modelliert werden
Nachteil: Markovketten mit Ressourcen notwendig

Markovketten mit Ressourcen: Grenzen der Beschleunigung

Markovketten mit Ressourcen können komplexe Berechnungen eines Computers simmulieren
$\Rightarrow$ exakte Berechnungsbeschleunigungen nicht immer beliebig möglich

Gut gewählte Markovketten können diesen Trick gut genug verstecken; mit beliebigen Markovketten nicht immer möglich
$\Rightarrow$ Durch Spieler festlegbare Markovketten mit Ressourcen können auffällige Fehler in beschleunigter Simulation aufweisen

Geschlossen lösbare Differentialgleichungen

Ressourcensysteme lassen sich durch Differentialgleichungen gut beschreiben

Haben Differentialgleichungen „geschlossene“ Lösungen, so lassen sie sich effizient beschleunigen

Beispiel:
Jeder Hase gräbt pro Tag ein Loch im Boden
Jedes Loch im Boden sammelt pro Tag ein Liter Wasser
$\Rightarrow$ Bei h Hasen, l Löchern und k Litern Wasser am Tag 0 hat man am Tag n: h Hasen, $l + h \cdot n$ Löcher und
$k + h \cdot l + h \cdot n \cdot (n-1)/2$ Liter Wasser
Effizienter und ungenauer: Integrieren, man hat
$k + h \cdot l + h \cdot n^2/2$ Liter Wasser

Allgemeine Differentialgleichungen

Differentialgleichungen, die sich geschlossen lösen lassen: Handbuch zu Differentialgleichungen, Internet, ...

Was geschlossen bedeutet, hängt von Satz verfügbarer Funktionen ab; machmal möglich: Vorausberechnete Differentialgleichung wiederholt verwenden

Allgemeiner Fall: Durch viele Methoden immernoch deutliche Beschleunigung gegenüber tickweiser Berechnung möglich; Lösungsmethoden von Differentialgleichungen: Handbuch, Internet, eigenständige Lehrveranstaltungen, ...

Flusssysteme

Flusssysteme: Netzwerke von fließenden Ressourcen,
Beispiele: Wasserfluss, Nährstofftransport in Bäumen, . . .

Abstrakt: Netzwerk von Knoten, jeweils mit Inhalt (Füllhöhe), Potential (auf welcher Höhe steht der Behälter), sowie Fluss zu Nachbarknoten

Ineffiziente Berechnung: Inhalt wird immer wieder Richtung Mittelwert des Inhalts umliegender Knoten gesetzt (Offset um Potential; Änderungen werden mit den umliegenden Knoten ausgetauscht)

Flusssysteme effizienter berechnen

Fluss zwischen Knoten speichern

Fluss ändert sich langsam so, dass Füllstand ausgeglichen wird

Füllstand ändert sich gemäß Summe der Flüsse

Fluss ändert sich im Laufe der Zeit immer weniger
$\Rightarrow$ Bei ausreichend geringer Änderung: Keine weitere Änderung berechnen, bei Nebeneffekten aus Fluss (z.B. Ansammlung von Flüssigkeit in einer Maschine) Nebeneffekt mit der Zeitdauer multiplizieren für Restzeit

Flusssysteme, Rechengenauigkeit

Was ist ausreichend genau gerechnet?

Alle Berechnungen erhöhen die Genauigkeit, schnelle Berechnung wichtiger als immer hohe Genauigkeit.
Rechenzeit vor allem in Situationen sehr hoch, in denen Genauigkeit nicht so gut durch Menschen eingeschätzt werden kann
$\Rightarrow$ Rechenzeitgrenzen vorgeben, innerhalb derer immer weiter rechnen; wenn Zeit aus, Rechenpause und Trigger zum Weiterrechnen setzen

Weitere Optimierung: Bei sehr geringer Änderung Berechnung ganz stoppen und in gewissen Fällen erst wieder starten (Änderungen auf Nachbarfeldern, Berechnung auf Nachbarfeldern, ggf. Active Block Modifier setzen)

Flusssysteme, Einsatzbeispiele

Einige Beispiele für die vielfältigen Möglichkeiten, die sich mit Flusssystemen berechnen lassen

- Wasserfluss; mit Drucksimulation (z.B. Wasserfelder unter anderen Wasserfelder können geringfügig mehr Wasser aufnehmen, das drückt Wasser auf seitliche Felder) sogar mit kommunizierenden Röhren
- Rohstofftransport in Pflanzen, damit lebende Bäume, die in Abhängigkeit von verfügbarem Wasser, Licht und Hindernissen realistischer wachsen: Wachstum an Stellen, an denen Versorgung ausreicht
- Bauwerkstabilität: Kräfteflusssimulation, wenn Kräfte nicht reichen Bauwerk zu tragen, stürzt es ein