Spieltheorien und Theoreme

Seminar: Randomisierte Algorithmen
Prof. Dr. R. Klein

Alexander Hombach

Eine bilinguale Ausarbeitung von Alexander Hombach, Daniel Herrmann und Ibraguim Kouliev
(Teil 1)
Rheinische Friedrich-Wilhelms-Universität Bonn

Inhaltsangabe

Kapitel 1 … Laufzeitbetrachtungen und Optimierung von Spielbäumen

Kapitel 1.1 … Auswertung von Spielbäumen

Kapitel 1.2 … Das Minimax-Prinzip

Kapitel 1.3 … $a - b -$ *pruning*

Kapitel 1.4 … Definitionen

Kapitel 1.5 … Deterministischer Ansatz

Kapitel 1.6 … Vorstellung des randomisierten Algorithmus

Kapitel 1.7 … Laufzeiten

Kapitel 1.8 … Wie geht’s weiter?

Kapitel 1.9 … Das Schere-Papier-Stein Spiel

Kapitel 1.9.1 … Optimale Strategie

1. Laufzeitbetrachtungen und Optimierung von Spielbäumen

1.1 Auswertung von Spielbäumen

Die allgemeine Frage die man sich auf den ersten Blick stellen könnte, wäre die Sinn- und Zweckfrage nach spielbasierten Techniken und der Einführung eines randomisierten Ansatzes. Dazu möchte ich erstmal genauer auf die Definition von Spielbäumen, deren Verwendung, Einsatzgebiet und möglichen Optimierungen eingehen.

Die Art von spielbasierten Techniken die im ersten und zweiten Kapitel (Alexander & Daniel) behandelt werden beziehen sich auf 2-Player-Games, wohingegen sich das dritte Kapitel (Ibraguim) auf den allgemeineren Fall bezieht.

Spielbasierten Techniken sind Algorithmen, welche die Spielsituation aus- oder zu bewerten, oder auch dazu verwendet werden um Gegenspieler zu simulieren oder ein Spiel komplett eigenständig zu Analysezwecken auszuführen. Um einen Algorithmus erfolgreich ausführen zu können, benötigt man die Spieldaten in entsprechend abstrahierter und aufbereiteter Form.

Zur Speicherung der Daten stehen einem dabei die verschiedensten Datenstrukturen wie Arrays, Listen, Bäume, etc. zur Verfügung. Wenn man die 2-Player-Games (wie in dieser Ausarbeitung geschehen) behandelt, so sind Baumstrukturen und speziell Spielbäume am nützlichsten.

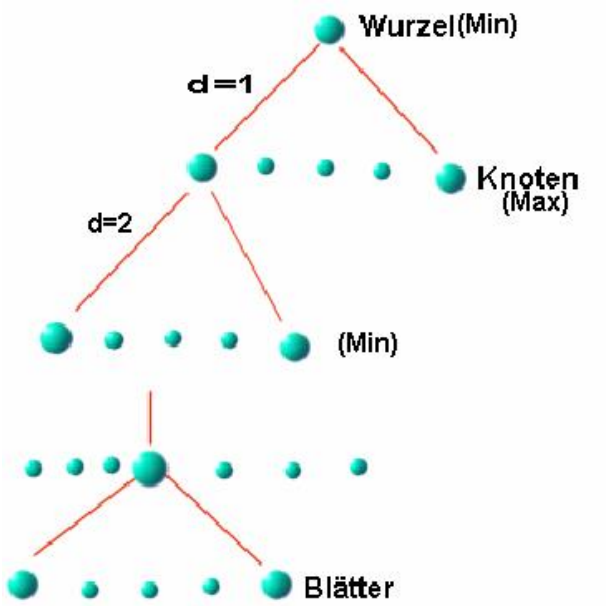
Bei 2-Player-Games gehen wir davon aus, das 2 Spieler gegeneinander in abwechselnden Spielzügen gegeneinander spielen und die Spielsituation / das Spielbrett jederzeit von beiden einsehbar ist. Diese Spiel- bzw. Wissenssituation wird als **perfekte Information** bezeichnet.

Bei der Auswertung von Spielbäumen kann man zwei Ansätzen nachgehen: Dem deterministischem und dem probabilistischem Ansatz. Zuerst werde ich auf den deterministischen Ansatz eingehen, die Vor- und Nachteile behandeln und als Lösungsvorschlag dann zum probabilistischem Ansatz übergehen.

Doch erstmal zu einigen grundlegenden Definitionen.

1.1.2 Was sind Spielbäume ?

Ein Spielbaum ist ein Binärbaum, dessen Knoten folgende Beschriftung aufweisen.

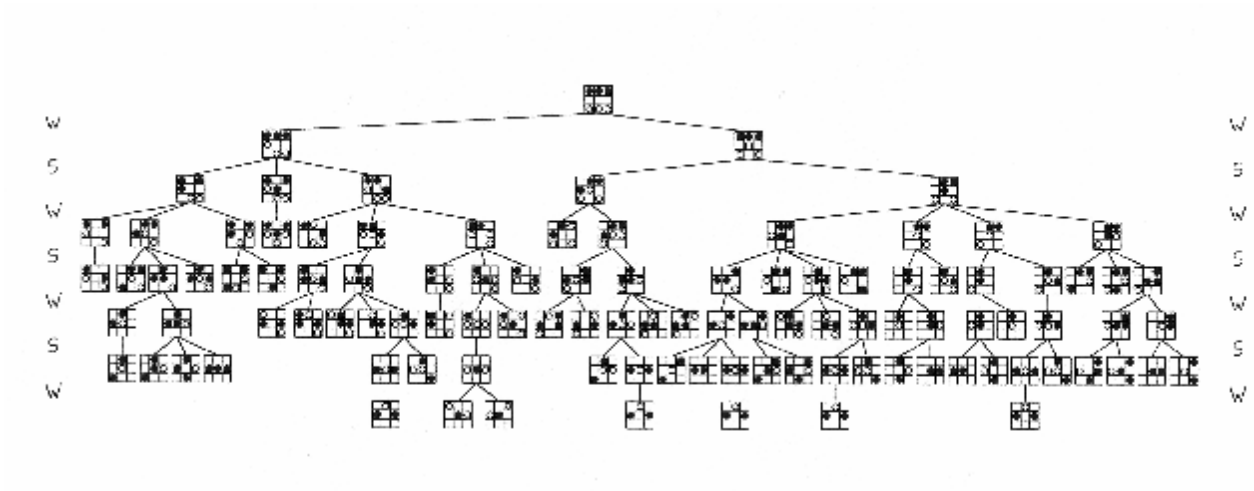


- Ungerade Distanz d von der Wurzel => Max-Knoten.

- Gerade Distanz von der Wurzel => Min-Knoten.

- Jedes Blatt hat einen Wert/ Gewicht (reelle Zahl) und repräsentiert den Ausgang eines möglichen Spieles.

Beispiel für einen Spielbaum



- Die Söhne eines Knotens stellen die Auswahlmöglichkeiten für den jeweiligen der beiden Spieler dar.

- Das Gewicht der Blätter/Knoten repräsentiert den Wert der Spielsituation für den Spieler (z.B. 1=Gewinn / 0=Verlust für den Max-Spieler).

- Die Spieler versuchen den „Wert des Pfades“ entweder zu max- oder minimieren.

Wenn man sich nun den Spielbaum für sein Spiel aufgebaut hat, kann man das Spiel über diesen laufen lassen und so mit der Zeit die Knotengewichte

anpassen. Ein Spieler wird als maximierender Spieler angesehen, der andere als Minimierender. Je nachdem ob ein Pfad zu einer dem Spieler günstigen Situation führt, wird das Gewicht des Knotens entweder maximiert (Max-Player) oder minimiert (Min-Player). Ist ein Spiel zu Ende, werden anhand des Ergebnisses die Gewichte der Knoten, die zu dieser Situation führten, angepasst.

So erhält man nach und nach einen Spielbaum, welcher das Spiel korrekt charakterisiert. Über diesen Baum kann man dann das Spiel bewerten bzw. seinen nächsten Zug planen. Die Wahl des nächsten Zuges ist das eigentliche Gebiet mit dem sich die spielbasierten Techniken beschäftigen.

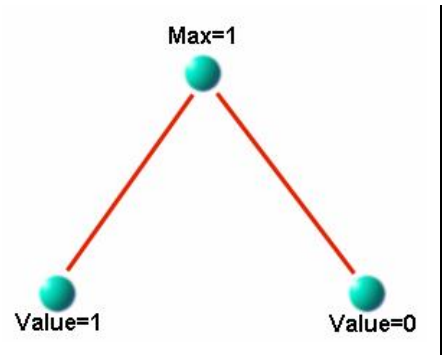
Eine dieser Techniken ist das Minimax-Prinzip.

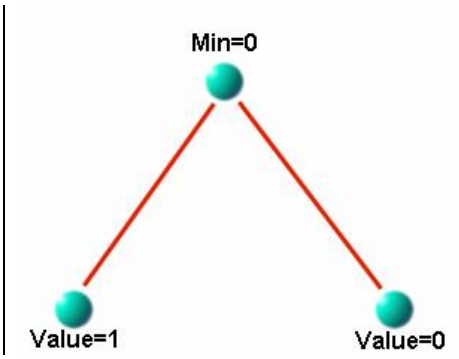
1.2 Das Minimax-Prinzip

Beim Minimax-Prinzip hat der Spielbaum folgenden Aufbau:

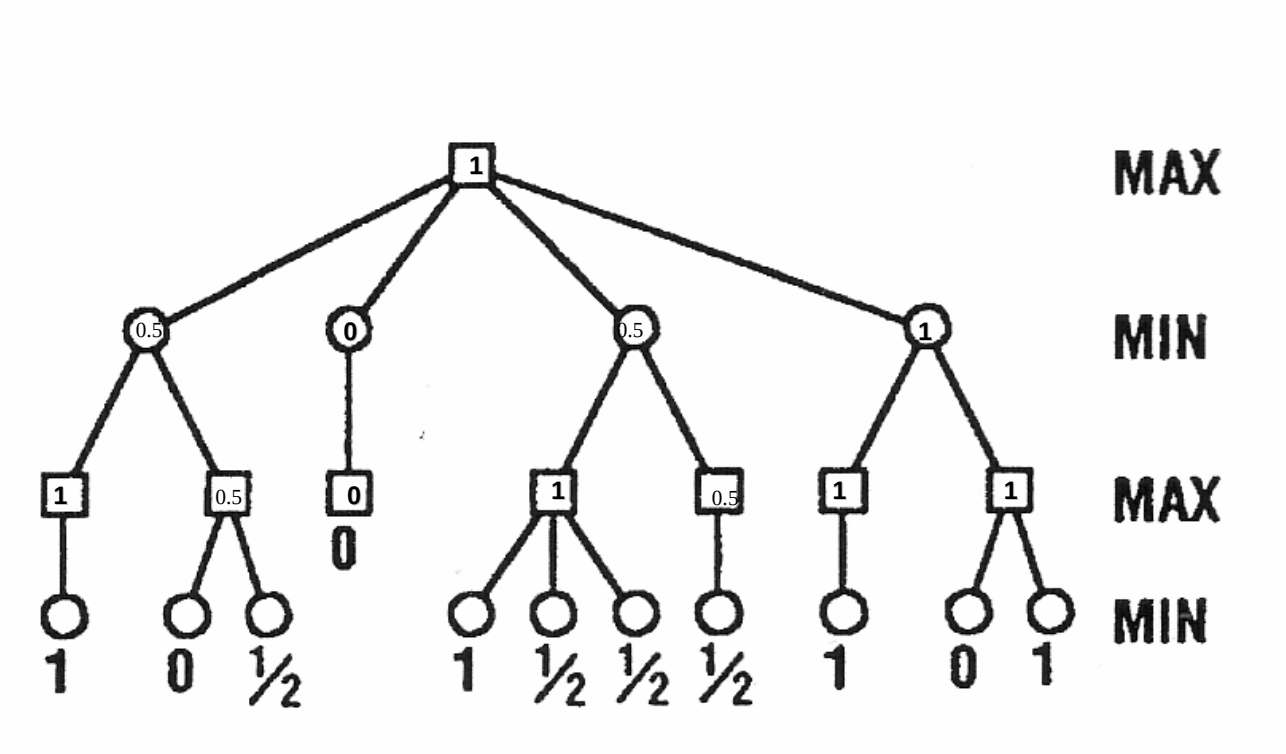
Wurzel / Knoten		Blatt
Max-Knoten (gerade Distanz von der Wurzel)	Min-Knoten (ungerade Distanz von der Wurzel)	
Wert des Knotens: Maximum aller direkten Nachfolger	Wert des Knotens: Minimum aller direkten Nachfolger	Enthält einen Wert, der die Spielsituation bewertet

Man hat also einen binären Spielbaum mit Min- und Max-Knoten. Ein Beispiel der Belegung bei den Knoten ist unten angegeben.





Beispiel für die Anwendung des Minimax-Prinzips:



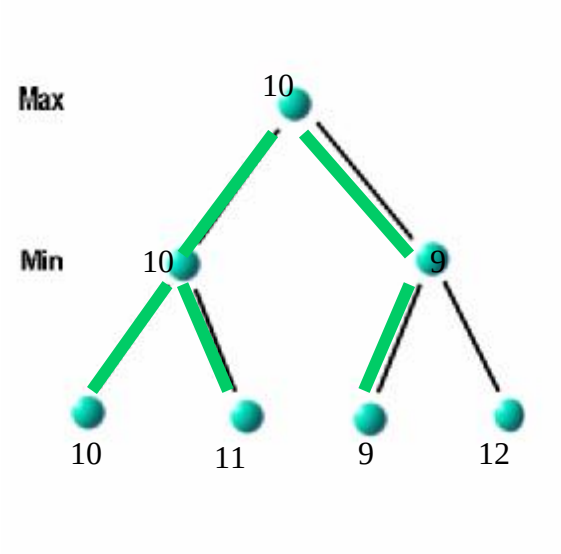
In diesem Beispiel kann man verfolgen, wie man von den Blättern aufsteigend zur Wurzel die Bewertung berechnen kann. In diesem Fall ist das Spiel für den Min-Spieler ein Verlustspiel, was sich einerseits direkt aus der Betrachtung der Wurzel ersehen lässt, und andererseits aus den Siegen und Niederlagen (4 Siege für Max, 2 für Min, 4 Unentschieden) an den Blättern.

Um den besten Zug zu finden wird bei Minimax der komplette Baum ausgewertet. Mit der Baumgröße / -tiefe wächst aber der Aufwand exponentiell.

Dies führt zum zweiten Auswertungsverfahren:

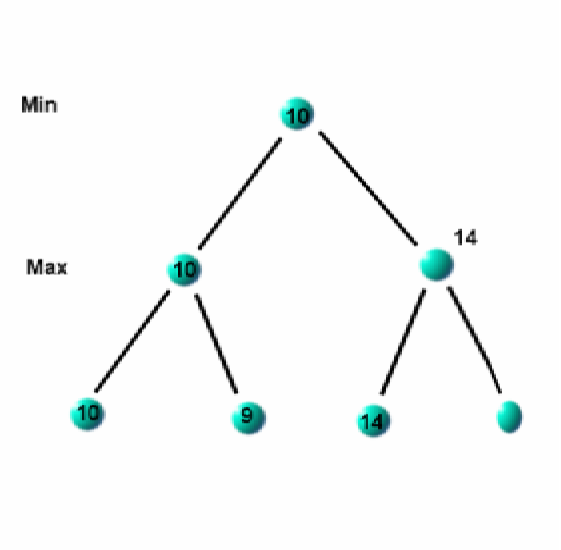
1.3 *a – b – pruning* **:**

Dieses Verfahren nutzt folgende Beobachtung aus.



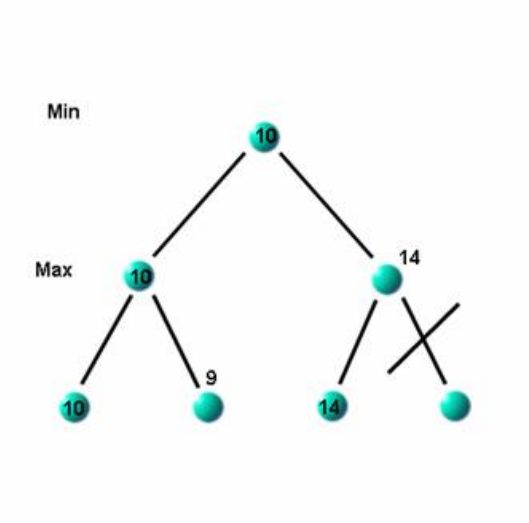
In diesem Beispiel ist es unnötig das 12er Blatt zu betrachten, da sein Vater ein Min-Knoten ist und schon den Wert 9 hat, aber mit der 10 um den Max-Knoten konkurriert.

Hier haben wir einen Min-Knoten, der auf Max-Knoten aufbaut. Auch hier ist es unnötig das letzte Blatt zu betrachten. Im linken Max-Knoten wurde die 10 ermittelt und da im rechten Zweig schon die 14 gefunden wurde (kann Dank Max-Knoten nur noch größer werden), kann man sich die Betrachtung des letzten Knotens sparen (Der Vater-Knoten ist ein Min-Knoten).



Die Idee ist also, bei der Betrachtung des Baumes Teilbäume auszusparen (*to prun engl. : abschneiden von unnützen Teilen*) und damit die Laufzeit zu verbessern. Dazu gibt es zwei Regeln:

a -Regel:



Diese Regel wird nur auf Max-Knoten angewandt.

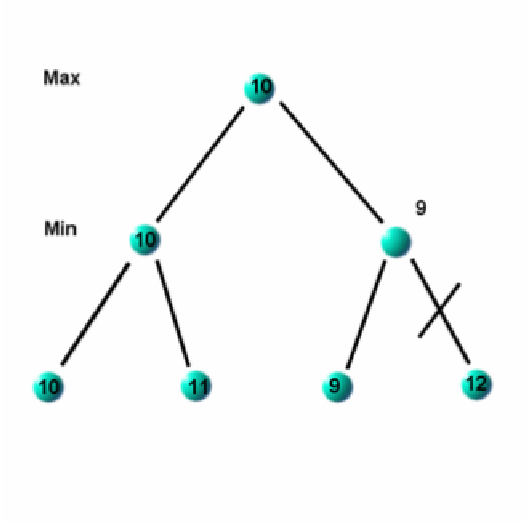
a ist das Minimum aller Min-Vorfahren.

a stellt dann die obere Grenze dar.

Betrachtung der Knoten bricht bei Überschreitung ab.

Im oberen Beispiel ist a = 10. Bei Betrachtung des rechten Teilbaumes hat der Knoten einen Zwischenwert von 14. Da dies ein Max-Knoten ist, können keine kleineren Werte mehr erreicht werden und kann deshalb keinen Einfluss mehr auf den Vater haben. Dies wird durch Überschreitung der a -Grenze indiziert.

b -Regel:



Diese Regel wird nur auf Min-Knoten angewandt.

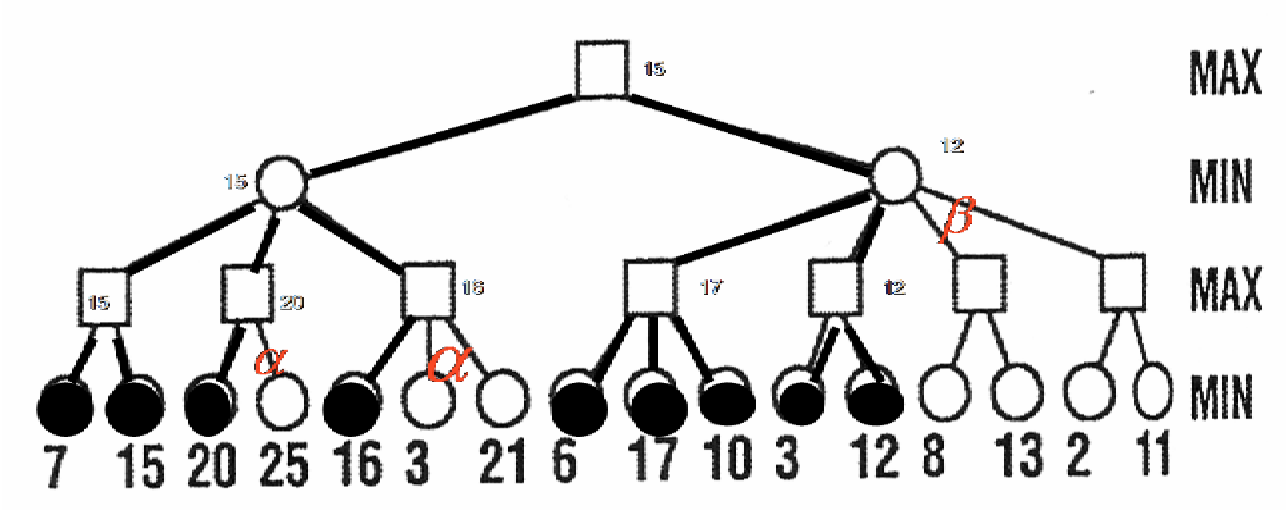
b ist das Maximum aller Max-Vorfahren.

b als untere Grenze.

Die Auswertung bricht bei Unterschreitung ab.

In diesem Beispiel liegt die b -Grenze bei 10. Bei der Betrachtung des rechten Teilbaumes tritt während der Auswertung beim Min-Knoten ein Wert von 9 auf. Da der resultierende Wert nur noch <= 9 sein kann und der Vater Max-Knoten schon einen Wert von 10 hat, kann man sich auch hier jede weitere Betrachtung sparen.

Ein Beispiel für $a-b$ -pruning:



In diesem Beispiel sind die Knoten, die von der Betrachtung ausgespart blieben, weiß eingefärbt. Von 16 Knoten wurden hier also nur 9 Knoten betrachtet, also fast nur 50% aller Knoten.

Fazit:

$a-b$ -pruning ist ein Minimax-Verfahren zur Bewertung von Spielbäumen. Es ermöglicht höhere Performance gegenüber den herkömmlichen Verfahren und sowohl mit unteren als auch oberen Schranken.

Laufzeitvergleiche über einen uniformen Spielbaum:

Interessant wäre jetzt natürlich die Laufzeiten in Zahlen ausdrücken zu können. Vielleicht gibt es ja auch noch weitere Verbesserungen zusätzlich zu $a-b$ -pruning.

Um eine einheitliche Umgebung zu schaffen, auf der wir die Laufzeitabschätzungen vollziehen können, kommen erstmal noch ein paar Definitionen.

1.4 Definitionen

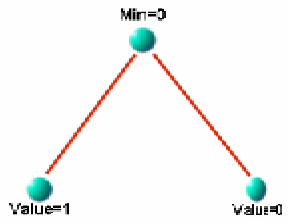
Definition Uniformer Baum:

Sei $T_{d,k}$ ein Baum T mit folgenden Eigenschaften:

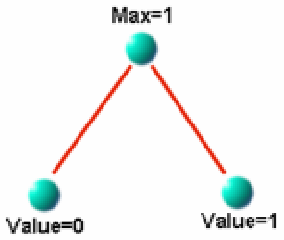
- Jeder Knoten und die Wurzel haben d Söhne.
- In jeder inneren Ebene des Baumes befinden sich abwechselnd nur Min- oder Max-Knoten.
- Jedes Blatt hat eine Distanz zur Wurzel von 2k.

- 2k deswegen, weil ein längster Pfad k Min-Knoten (incl. Wurzel) und k Max-Knoten beinhaltet.
- Das Gewicht der Blätter kann nur 1 oder 0 betragen (wie Bit-Darstellung).

Was einem bei dieser Darstellung auffällt ist, das sich die Min- und Max-Knoten verhalten wie AND- oder OR-Verknüpfungen.



Blatt 1	Blatt 2	Min	And	Max	OR
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1	1	1	1	1



Wie man in der obigen Darstellung sehen kann verhält sich ein Min-Knoten äquivalent zu einer AND-Verknüpfung und ein Max-Knoten stimmt überein mit einer OR-Verknüpfung.

Definition Instanz:

Ein Baum $T_{d,k}$ mit einer Belegung aller d^{2k} Blätter.

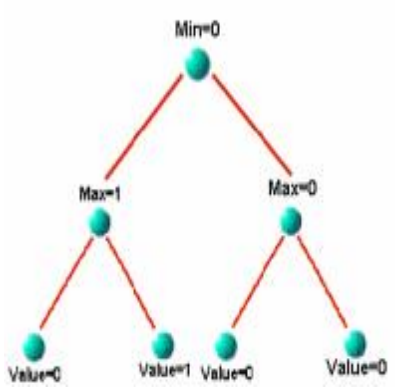
Die folgenden Laufzeitbetrachtungen basieren alle auf der Frage: Wie lange dauert es eine Instanz zu bewerten?
Dazu betrachten wir nun den deterministischen und den probabilistischen Ansatz.

1.5 Deterministischer Ansatz:

Beschreibung des Algorithmus:

Für jeden Vater der 2k-1 Ebene gilt:

- Wähle Blatt und speichere Gewicht
- Wähle nächstes Blatt bis alle Gewichte gesammelt sind.
- Bestimme das Gewicht des Vaters und fahre auf der nächst höherer Ebene fort.



Der Unterschied zwischen dem deterministischen und dem probabilistischen Ansatz liegt in der Wahl des nächsten Knotens.

Deterministisch vs. Randomisiert	
Deterministisch	Randomisiert
Wahl des nächsten Knotens über eine festgelegte Reihenfolge	Wahl des nächsten Knotens erfolgt zufällig

Im Worst-Case muss der deterministische Algorithmus trotz $a-b$ -pruning alle d^{2k} Knoten bearbeiten.

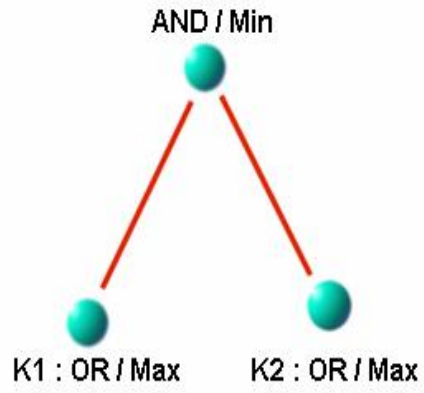
Die Frage ist, wie der gleiche Fall mit dem randomisierten Algorithmus aussieht?

Um diese Frage zu klären betrachten wir einen $T_{2,k}$Baum. Das heißt, dass jeder Knoten d=2 Söhne hat und $2^{2k} = 4^k$ *Blätter*.

Wir können uns nun folgende Besonderheiten zu Nutze machen:

<u>1) Wir betrachten einen AND-Knoten mit 2 OR-Söhnen</u>

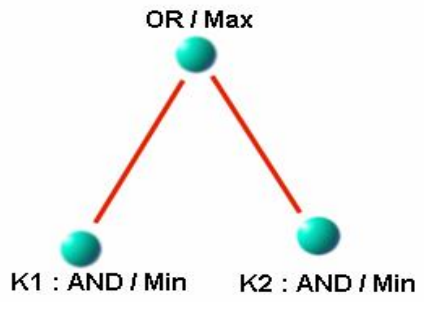
- Sobald an einem Sohn eine 0 auftritt, muss der andere Sohn nicht mehr betrachtet werden.
- Worst-Case: Beim Det.-Algo. ist dies immer der 2te Sohn.



K1	K2	AND
0	0	0
0	1	0
1	0	0
1	1	1

<u>2) Wir betrachten einen OR-Knoten mit 2 AND-Söhnen</u>

- Sobald eine 1 auftritt, muss der andere Sohn nicht mehr betrachtet werden.
- Worst-Case: Siehe oben.



K1	K2	OR
0	0	0
0	1	1
1	0	1
1	1	1

In beiden Fällen besteht für den randomisierten Algorithmus jeweils eine 50% Chance, den Knoten auszuwählen, der die Betrachtung des anderen erspart. Bei diesen Betrachtungen sollte man beachten, dass es sich hierbei um Worst-Case-Bäume handelt.

Die Laufzeiten sehen folgendermaßen aus:

1) AND-Knoten mit 2 OR-Söhnen

K1	K2	AND
0	0	0
0	1	0
1	0	0
1	1	1

Deterministischer Algorithmus:

Kosten = Anzahl der zu bearbeitenden Knoten = 3

Randomisierter Algorithmus:

Dabei betrachten wir folgende Ereignisse:

Ereignis E1 = „Sohn mit Gewicht 0 als erstes überprüft“

P(E1) = 0.5

E[E1] = P(E1) * Kosten // Der Erwartungswert des Ereignisses E1

= 0.5 * 3

= 1.5

In diesem Fall heißt das:

Det.-Algo. Kosten 3 vs Rand.-Algo. Kosten 1.5

2) OR-Knoten mit 2 AND-Söhnen

K1	K2	OR
0	0	0
0	1	1
1	0	1
1	1	1

Deterministischer Algorithmus:

Kosten = Anzahl der zu bearbeitenden Knoten = 3

Randomisierter Ansatz:

Ereignis E1 = „Sohn mit Gewicht 1 als erstes überprüft“

P(E1) = 0.5
E[E1] = P(E1) * Kosten
= 0.5 * 3
= 1.5

Auch in diesem Fall heißt das:

Det.-Algo. Kosten 3 vs Rand.-Algo. Kosten 1.5

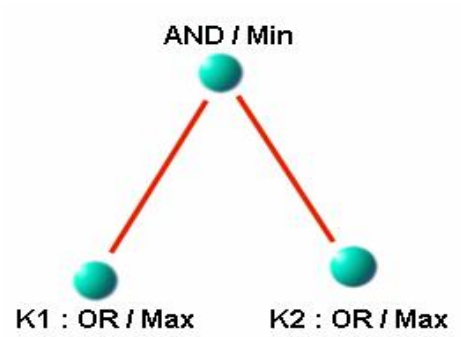
Im Worst-Case ist also der Rand.-Algorithmus bezüglich der erwarteten Laufzeit der Schnellere

1.6 Vorstellung des randomisierten Algorithmus:

Fallunterscheidung: Betrachtet man einen Max- (OR) bzw. einen Min-Knoten (AND) und dessen Söhne?

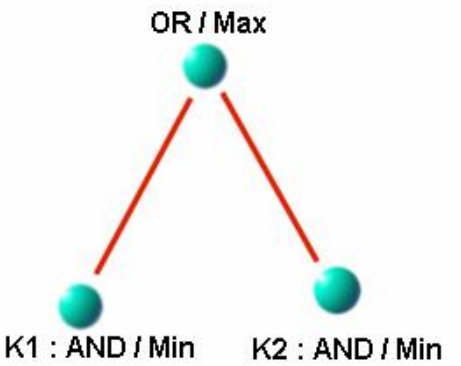
- Vater als AND-Knoten mit 2 OR-Söhnen:

Gewicht des gelesenen Sohnes	
1	0
Lese den nächsten Sohn aus	Setze direkt das Gewicht des Vaters auf 0



- Vater als OR-Knoten mit 2 AND-Söhnen:

Gewicht des gelesenen Sohnes	
1	0
Setze direkt das Gewicht des Vaters auf 1	Lese den nächsten Sohn aus



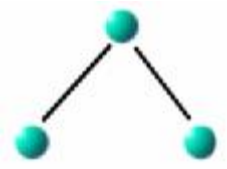
1.7 Laufzeiten

Nachdem wir nun die Vorgehensweise des Algorithmus kennen, können wir auch Aussagen über seine Laufzeit treffen. Um direkt mit der Tür ins Haus zu fallen: Die Laufzeit für einen beliebigen $T_{2,k}$ beträgt 3^k.

Diese Behauptung beweisen wir per Induktion.

Induktionsbeginn mit k = 1:

$$T_{2,1} = 3$$

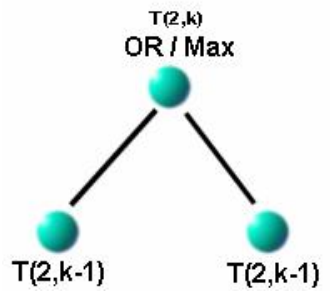


Induktionsschluss k-1 -> k: (Dieser Schluss ist schnell geführt…)

$$T_{2,k-1}\ mit\ Laufzeit\ 3^{k-1} \rightarrow T_{2,k}\ mit\ Laufzeit\ 3^k$$

Nun kommen wir auf die Fallunterscheidung von vorhin zurück.

1) Wurzel ist ein OR-Knoten



Dann betrachten wir folgende Ereignisse:

E1 = „Teilbaum mit Gewicht 1 gewählt“
E2 = „Beide Teilbäume überprüft wegen Gewicht 0“

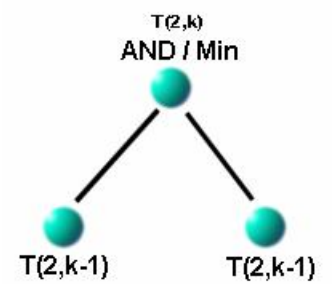
Dann betragen die Wahrscheinlichkeiten:

P(E1) = 0.5
P(E2) = 0.5

Und die Kosten betragen:

Kosten(E1) = 3^{k-1} // nur ein Teilbaum wird betrachtet
Kosten(E2) = $2 \cdot 3^{k-1}$ // beide Teilbäume werden betrachtet

2) Wurzel ist ein AND-Knoten



Auch hier betrachten wir wieder 2 Ereignisse:

E1 = „Teilbaum mit Gewicht 0 ausgewählt“

E2 = „Beide Teilbäume überprüft wegen Gewicht 1“

Die dazu gehörigen Wahrscheinlichkeiten betragen:

P(E1) = 0.5

P(E2) = 0.5

Die Kosten liegen bei:

Kosten(E1) = $\frac{1}{2} \cdot \frac{3}{2} \cdot 3^{k-1}$ // Es wird nur 1 Teilbaum betrachtet

Kosten(E2) = $\frac{1}{2} \cdot 2 \cdot \frac{3}{2} 3^{k-1}$ // Beide Teilbäume werden betrachtet

Aus beiden Ereignissen folgt, dass sich die insgesamt zu erwartete Laufzeit wie folgt zusammensetzt:

$$\frac{1}{2} \cdot 2 \cdot \frac{3}{2} \cdot 3^{k-1} + \frac{1}{2} \cdot \frac{3}{2} \cdot 3^{k-1} = \left(\frac{3}{2} + \frac{3}{4} \right) \cdot 3^{k-1} = \frac{9}{4} \cdot 3^{k-1} \leq 3^k$$

Aus Fall 1) und 2) ergibt sich die Laufzeitabschätzung:

Wurzel	AND (MIN)	OR(MAX)
	3^k	$\frac{3}{2} * 3^{k-1}$

Damit ergibt sich unsere Behauptung.

Theorem:

Eine beliebige Instanz eines $T_{2,k}$ hat eine erwartete Laufzeit von 3^k Schritten.

Aus dieser Laufzeitabschätzung lässt sich nun eine genaue Schranke angeben.
Denn ein $T_{2,k}$ hat n = $d^{2k} = 2^{2k} = 4^k$ Blätter.
Daraus folgt, dass k = log 4n
Wenn man dies einsetzt, erhält man: $3^k \rightarrow 3^{\log_4 n} = n^{\log_4 3} \approx n^{0,793}$

Ergebnis:
Die Auswertung eines uniformen binären AND-OR-Baumes mittels des randomisierten Algorithmus ist in einer Zeit von $\approx n^{0,793}$ zu bewerkstelligen.

1.8 Wie geht's weiter?

Bis jetzt haben wir uns mit der deterministischen und randomisierten Auswertung eines binären Spielbaumes beschäftigt. Dazu haben wir das Minimax-Prinzip eingeführt, die Instanz definiert, was man unter einem uniformen Baum versteht und eine Schranke für den randomisierten Algorithmus hergeleitet.
Aber gibt es denn vielleicht einen deterministischen Algorithmus mit einer besseren Laufzeit als $n^{0,793}$ oder generell eine kleinere Schranke für dieses Problem?
Wir werden dazu weiterhin mit dem Minimax-Prinzip arbeiten, welches wir für Algorithmen mit endlicher Laufzeit und beliebiger Eingabe verwenden. Dieses benutzen wir nun zur Beweisführung unterer Schranken.

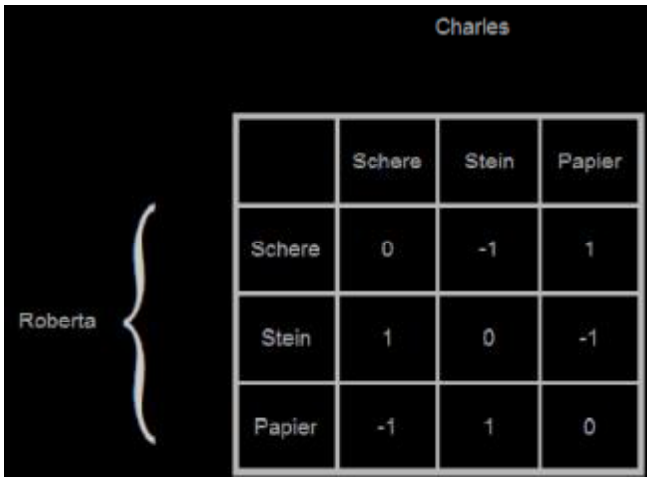
1.9 Das Schere-Papier-Stein Spiel

Zur Veranschaulichung spielen wir auf der Suche nach Antworten ein weiteres Spiel. Das Schere-Papier-Stein Spiel (SPS). Auch hier haben wir 2 Spieler; nennen wir sie mal Roberta und Charles. Jeder von beiden entscheidet sich geheim für ein Symbol (Schere, Papier, Stein) und die Entscheidung über Sieg und Niederlage läuft nach diesem Schema:

- Schere schlägt Papier
- Papier schlägt Stein
- Stein schlägt Schere

Der Gewinner dieser Runde erhält vom Verlierer 1€.

Diese Spielsituation kann man auch durch eine so genannte Payoff-Matrix darstellen.



In den Zeilen / Columns der Matrix befinden sich die Wahlmöglichkeiten, die Charles zur Verfügung stehen; in den Spalten / Rows stehen die von Roberta. Ein Eintrag in der Matrix entspricht dem Gewinn für Roberta und dem Verlust für Charles.

SPS ist ein typisches two-person zero-sum Spiel. Generell kann jedes two-person zero-sum Spiel durch Payoff-Matrix dargestellt werden.
Zero-Sum bedeutet, dass es keine Bank gibt, die einen Betrag für sich einheimst. Die Gewinne und Verluste der Spieler gegeneinander aufsummiert ergeben also den Wert 0.

<u>Nun generalisieren wir die Darstellung</u>

Die Strategien von
Spieler R stehen in den Rows / Zeilen
Spieler C stehen in den Columns / Spalten

Payoff-Matrix:

Beispiel	C1	C2
R1	-1	1
R2	1	-1

M_{ij} : Betrag den C an R zahlt, wenn Spieler R die Strategie i und Spieler C die Strategie j wählt.

Das Ziel der Spieler lautet:
R: Maximierung des Payoff's (das was er erhält)
C: Minimierung des Payoff's (das was sie zahlen muss)

Zusätzlich haben die Spieler während des Spiels keine Information über die gegnerische Strategie (zero-information).

1.9.1 Optimale Strategie

Sagen wir mal, das Spieler R die Strategie i wählt und hat dadurch einen garantierten minimalen Payoff.

$$\min_j M_{ij}$$

Payoff	1	.	.	.	M
1			P(1, j)		
.			.		
i			.		
.			.		
M			P(M, j)		

Die optimale Strategie für Spieler R würde sich also so ergeben, das er versucht mit seinem Zug seinem Gegenspieler möglichst schlechte Möglichkeiten zu bieten. Er würde also versuchen, denn minimalen Gewinn, den er selber erhalten kann (durch die Wahl einer Strategie) zu maximieren.

Definition:
Eine optimale Strategie für Spieler R bedeutet:

$$i \xrightarrow{\max} \min_j M_{ij} \rightarrow \quad \max_i \min_j M_{ij}$$

$$V(alue)_R = \max_i \min_j M_{ij}$$

Payoff	1	.	.	.	M
1					
.					
i	P(i,1)	.	.	.	P(i,M)
.					
M					

$$V(alue)_C = \max_j \min_i M_{ij}$$

Payoff	1	.	j	.	M
1			P(1, j)		
.			.		
.			.		
.			.		
M			P(M, j)		

Normalerweise gilt, das wenn ein Spieler 1 Euro gewinnt, das der andere Spieler diesen Euro verliert. Das würde dann z.B. Vc=1 und Vr=-1

entsprechen. Sollte allerdings Vc = Vr = V sein, so sagt man, dass das Spiel eine Lösung mit dem Wert V hat.

Definition Lösung / Sattelpunkt:
Die Wahl der optimalen Strategien mit Strategie $r \quad für \quad Spieler \quad R, g \quad für \quad Spieler \quad C$ und dem Wert $V = M_{rg}$ nennt man Lösung eines Spieles.

Beispiel, eine modifizierte Variante

Dazu schauen wir uns nun ein Beispiel an. Es handelt sich wieder um SPS, allerdings liegt jetzt eine leicht modifizierte Payoff-Matrix vor.

	Schere	Papier	Stein
Schere	0	1	2
Papier	-1	0	1
Stein	-2	-1	0

In diesem Fall gibt es nur eine Lösung mit V=0, bei der Wahl von $r = 1 \quad und \quad g = 1$.

$V_R = \max_i \min_j M_{ij}$

$V_C = \min_j \max_i M_{ij}$

Hierbei muss es auch nicht immer eine optimale Lösung oder Strategie geben. Zusätzlich können die Spieler von den Gegnerstrategien wissen und davon Gebrauch machen. Dies ist allerdings nur bei einem deterministischem Spielverlauf möglich.

fin.

Literatur:
Motwani, Raghavan: Randomized Algorithms, Cambridge Press, 1995