Künstliche Intelligenz

Suchmethoden für Spiele

Dr. Christian Meilicke
Research Group Data and Web Science
Universität Mannheim

Teile der Vorlesung basieren auf einem
Foliensatz von Prof. Dr. Heiner Stuckenschmidt

Übersicht

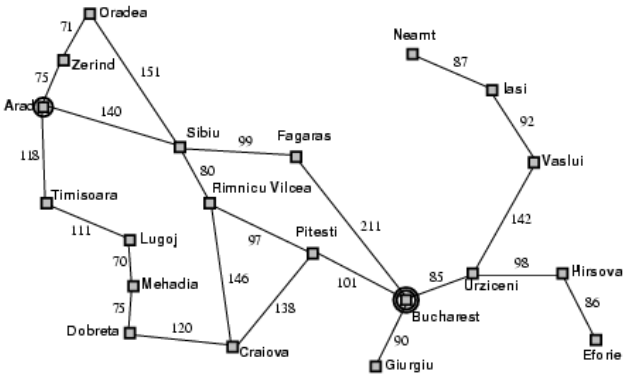
- Bisher:
  - (Vollständige) Suchverfahren von Breitensuche bis A* Suche
  - Lokale Suchverfahren
- Heute:
  - Spiele als Suchprobleme
  - Deterministisch und vollständig beobachtbar
    - Heuristiken und Min-Max Bäume, Alpha-Beta Pruning, Eröffnung und Endspiel
  - Nicht deterministische Spiele
  - Partiell beobachtbare Spiele
- Nächste Woche: MCTS Spielbaumsuche
- Danach: Programmierprojekt I

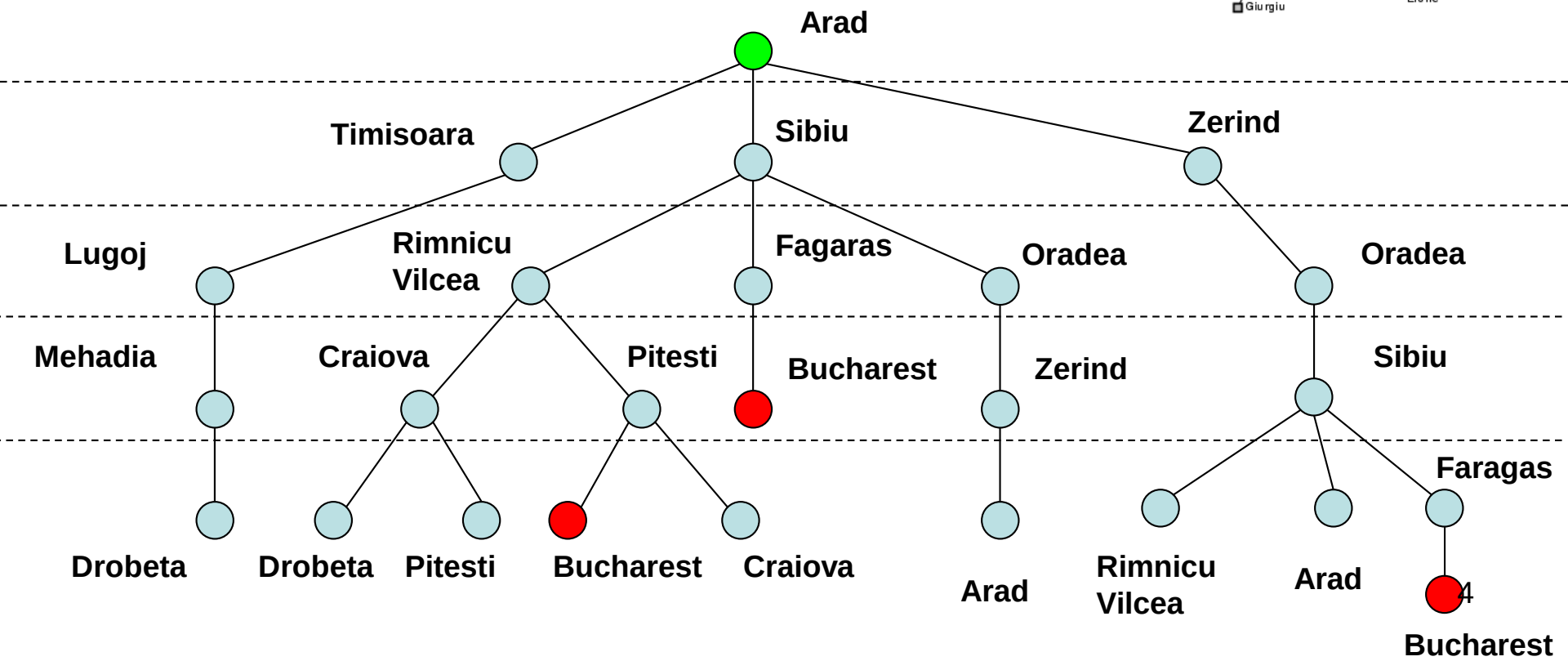
Spieltypen

	Deterministic	Stochastic
Fully Observable	Schach, Go	Backgammon, Monopoly
Partially Observable	Schiffe Versenken	Bridge, Poker

- Single-Agent vs. Multi Agent
- Zunächst (und vor allem)
  - **Fully Observable, Deterministic, Multi-Agent**

Erinnerung Suchverfahren



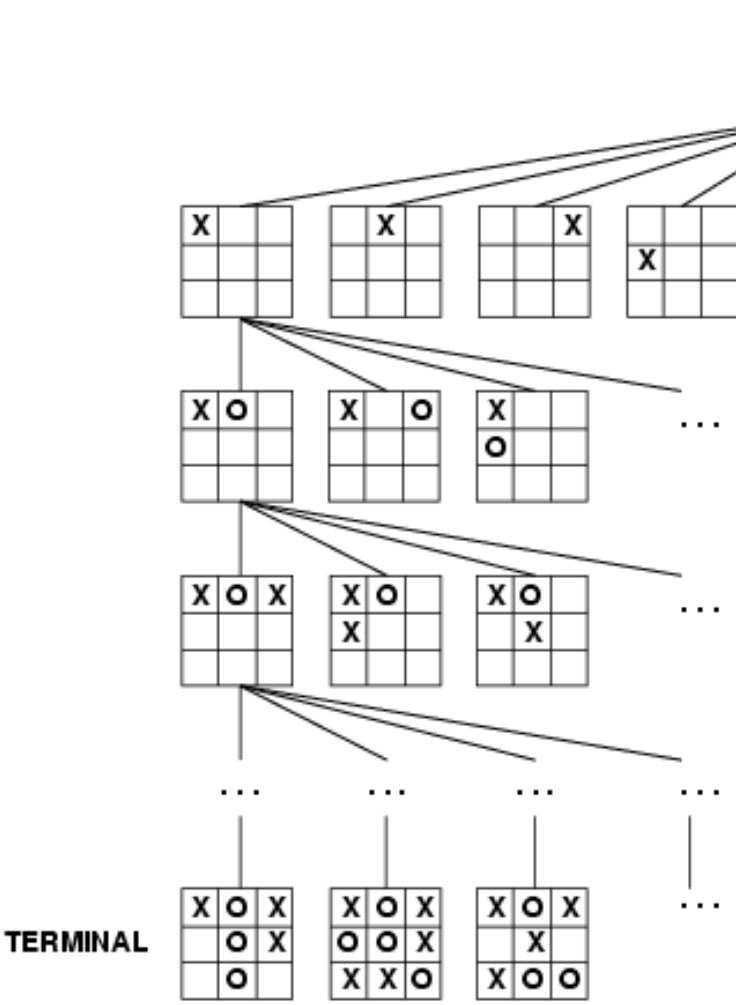


Spiele als Suchproblem

- Formulierung
  - Lösungsschritte: Spielzüge
  - Zielzustand: Gewinn oder Verlust
  - Startzustand: Aktuelle Position / Spielstand
  - Kosten?
  - Suchraum: “Game-Tree” (Spielbaum)

- Problem:
  - Züge des Gegners sind nicht planbar, oder doch ?

Beispiel: Tic-tac-toe



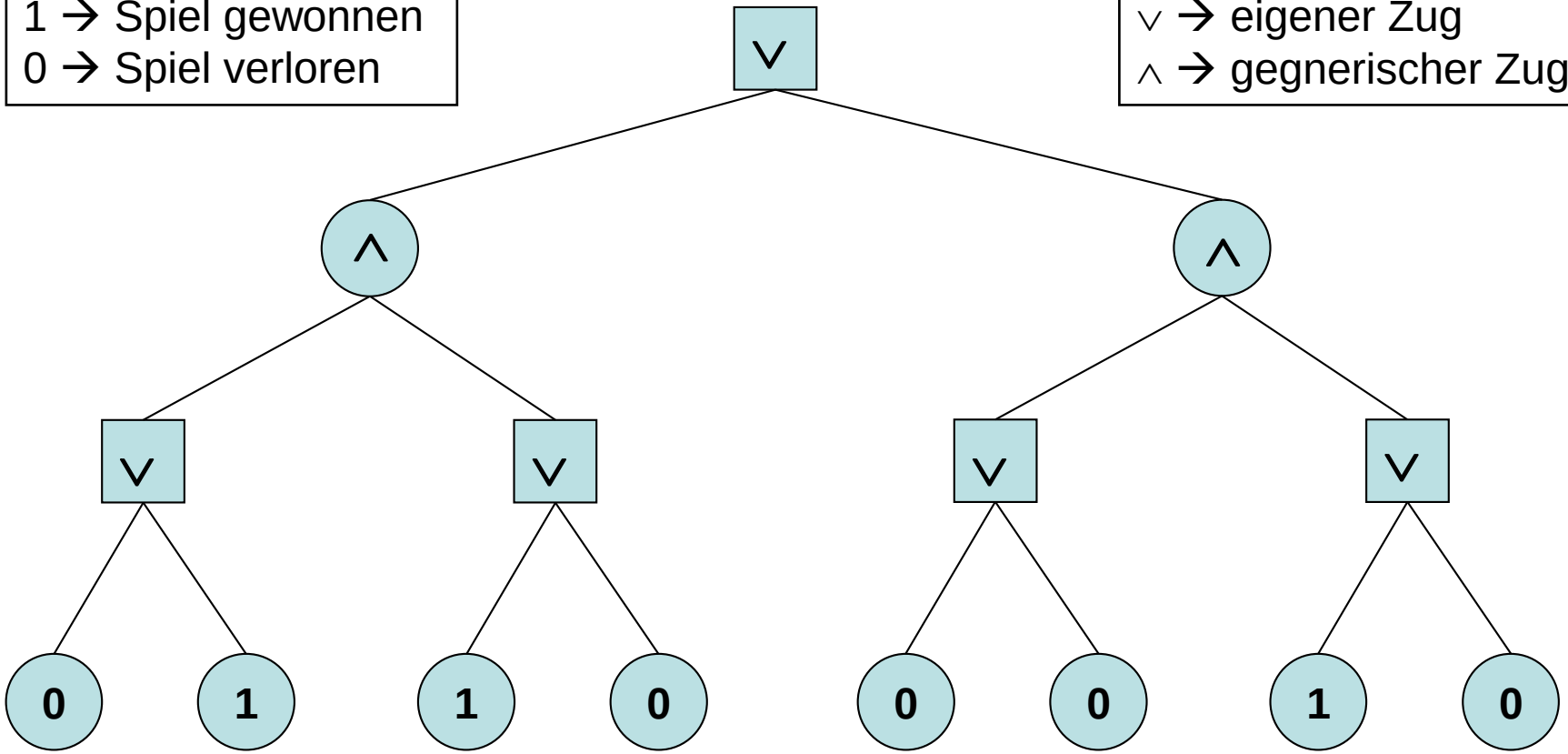
2 Spieler: o und x

Deterministic,
Fully Observable
Nullsummen Spiel
(Summe von Verlust und Gewinn ist immer Null)

Spiele als UND/ODER Bäume

1 → Spiel gewonnen
0 → Spiel verloren

∨ → eigener Zug
∧ → gegnerischer Zug



Werden wir dieses Spiel gewinnen ?

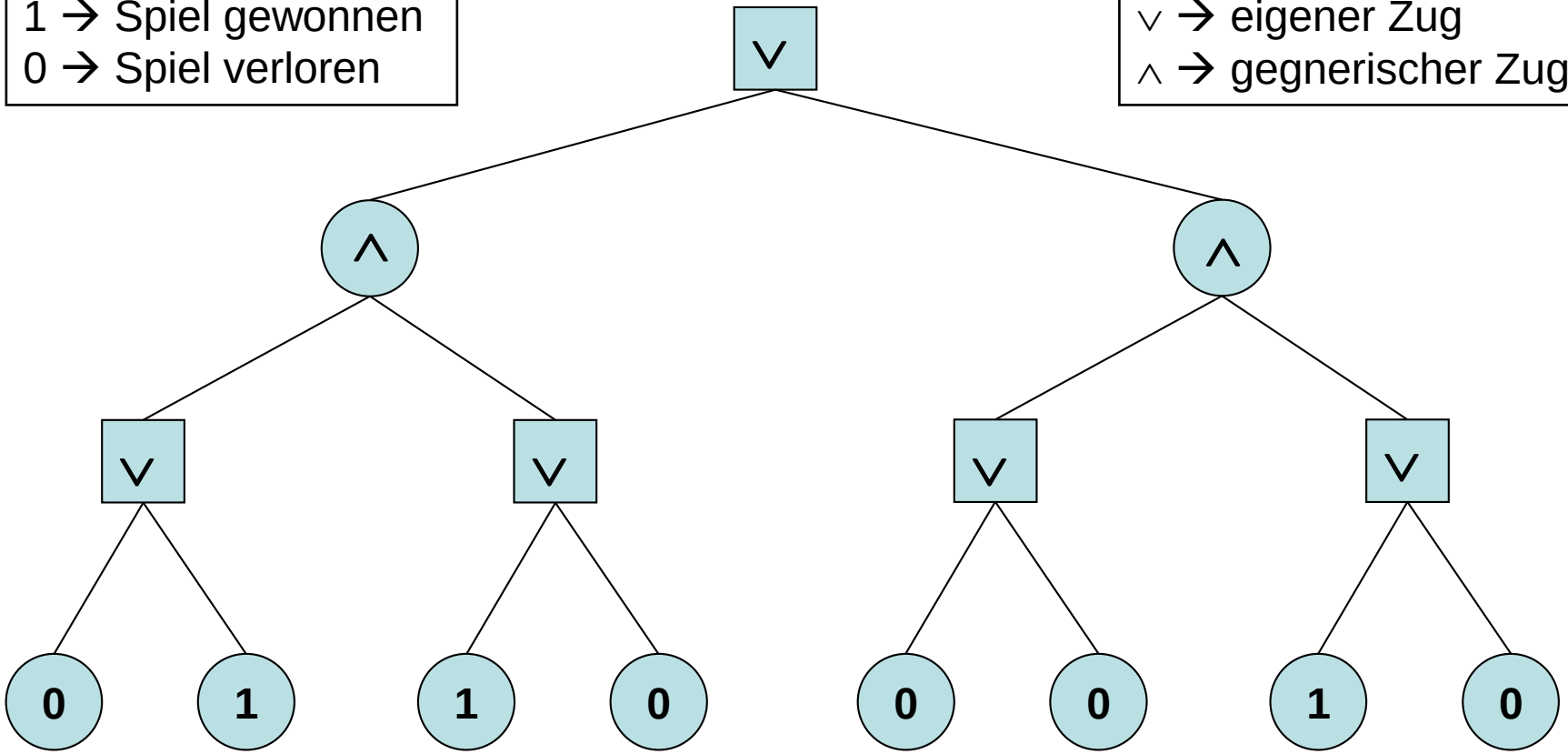
Spiele als UND/ODER Bäume

MIN

MAX

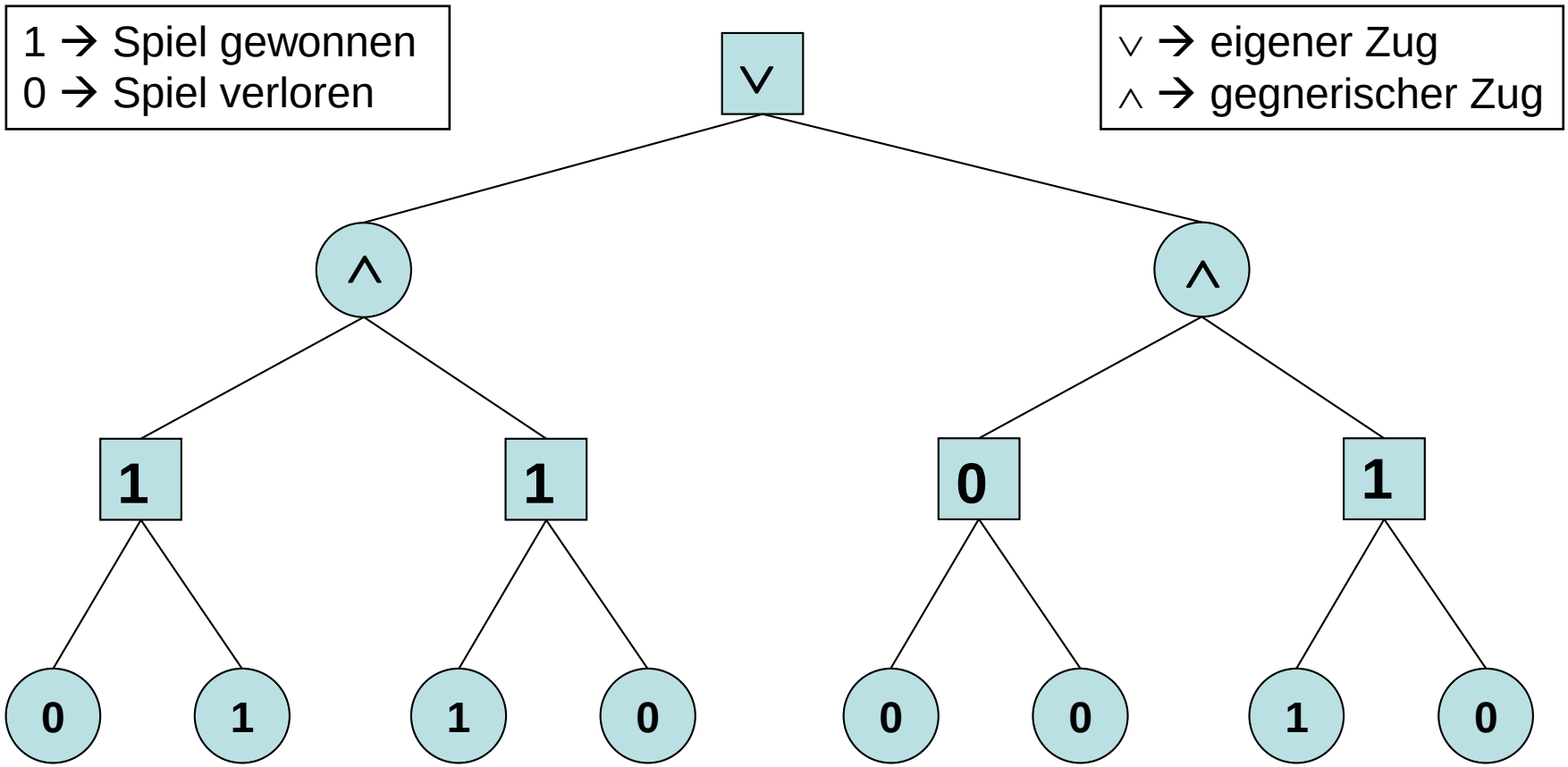
1 → Spiel gewonnen
0 → Spiel verloren

∨ → eigener Zug
∧ → gegnerischer Zug



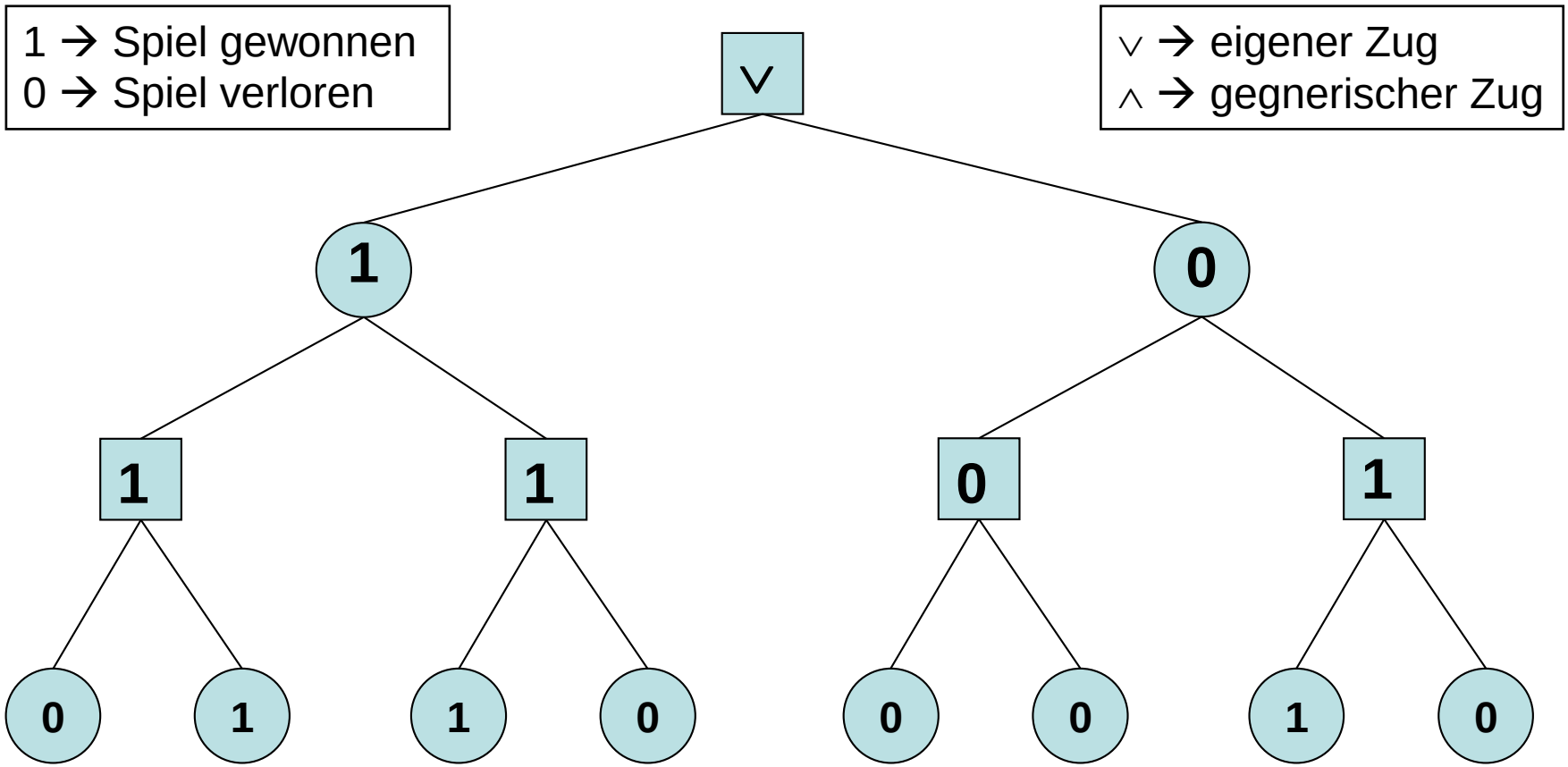
Werden wir dieses Spiel gewinnen ?

Spiele als UND/ODER Bäume



Werden wir dieses Spiel gewinnen ?

Spiele als UND/ODER Bäume

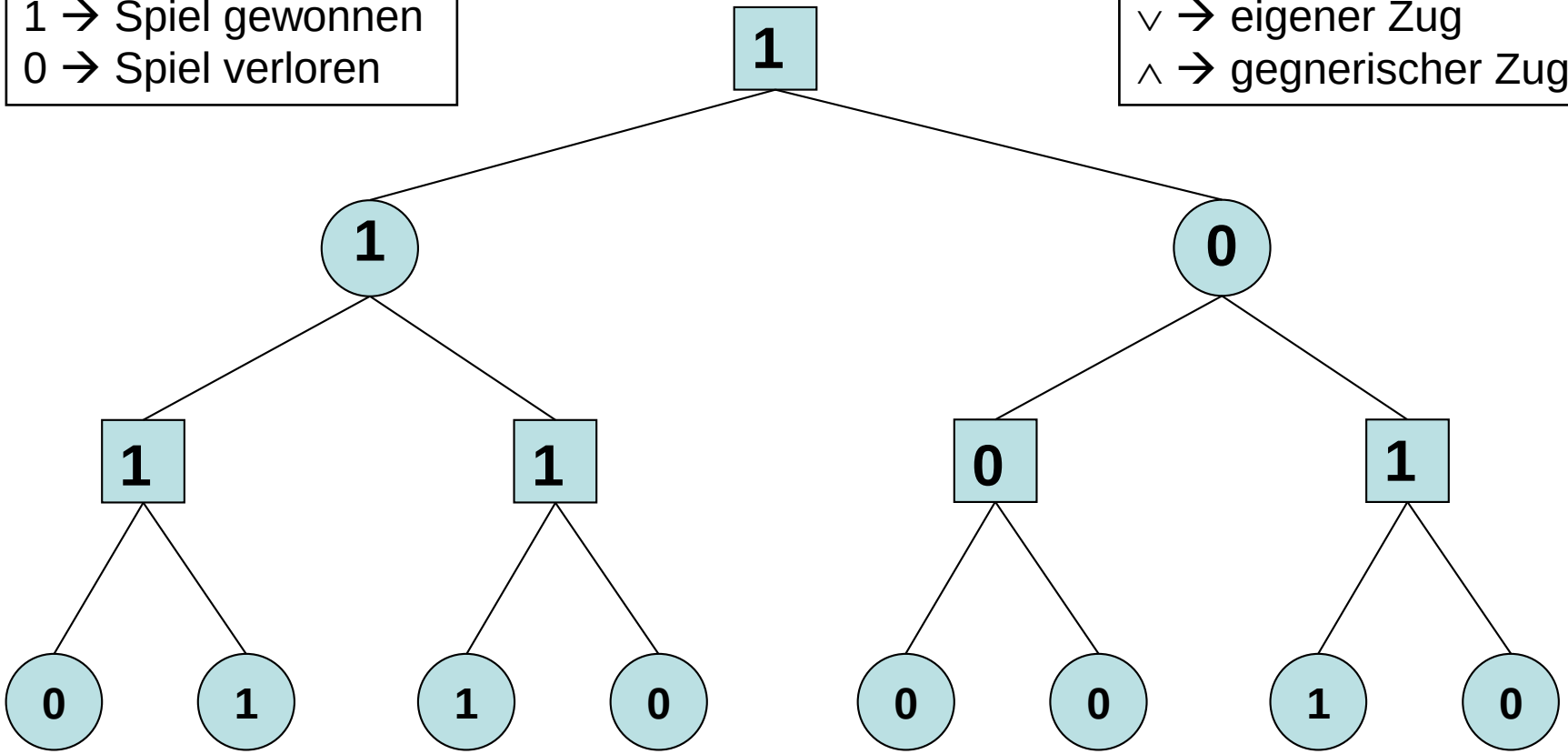


Werden wir dieses Spiel gewinnen ?

Spiele als UND/ODER Bäume

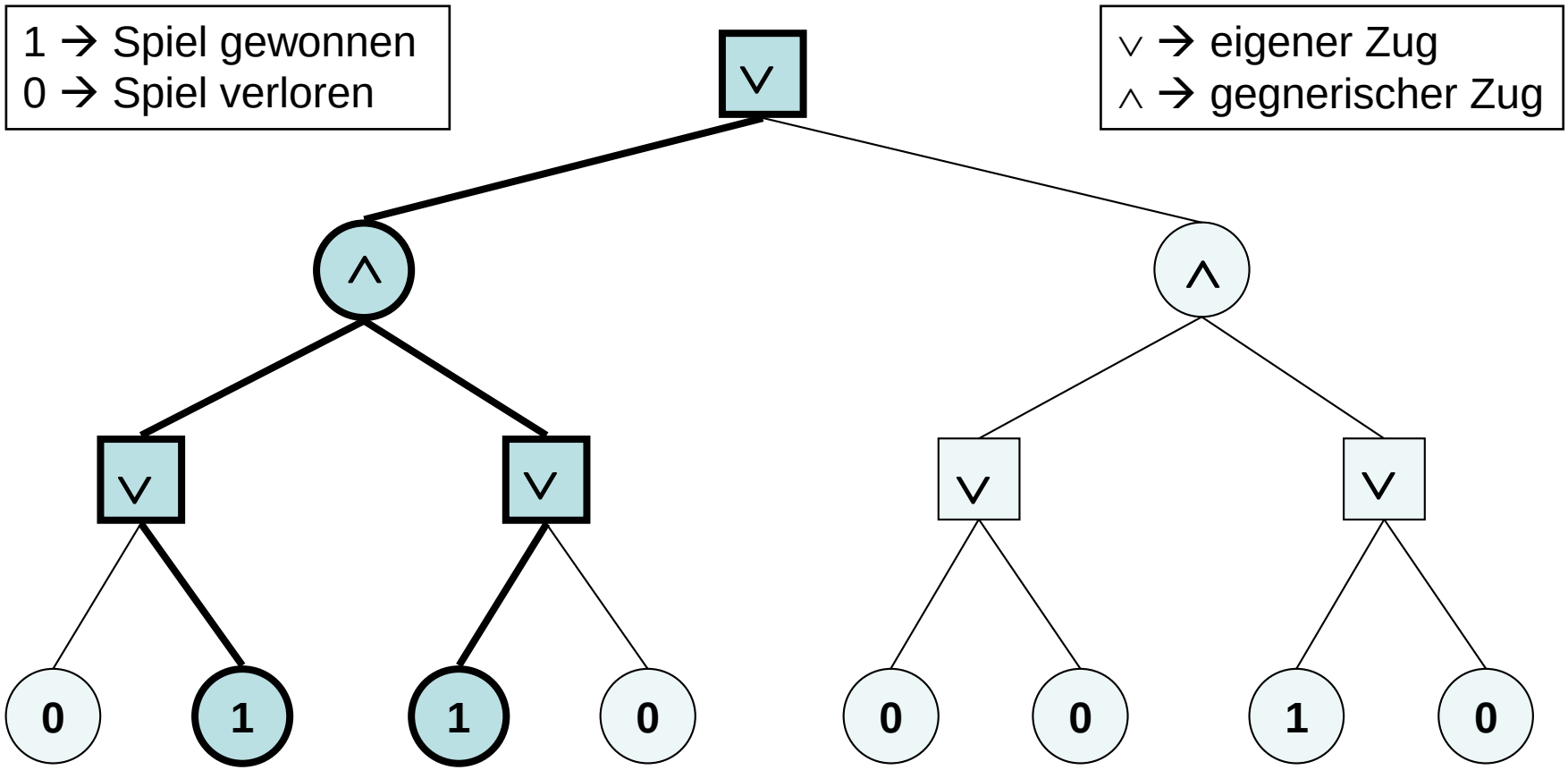
1 → Spiel gewonnen
0 → Spiel verloren

∨ → eigener Zug
∧ → gegnerischer Zug



Werden wir dieses Spiel gewinnen ?

Spiele als UND/ODER Bäume



Ja, wenn wir richtig ziehen !!

Probleme der Spielesuche

Wir können bestimmen, welche Züge zum Sieg führen, aber:

- Problem:
  - Suchbaum in der Regel zu groß, um ihn vollständig zu durchsuchen
- Lösung:
  - Tiefenbeschränkte Suche bis zu einer realisierbaren Tiefe
- Problem:
  - Woher wissen wir, wie das Spiel wirklich ausgeht ?
- Lösung:
  - Bewertungsheuristik für Zwischenzustände

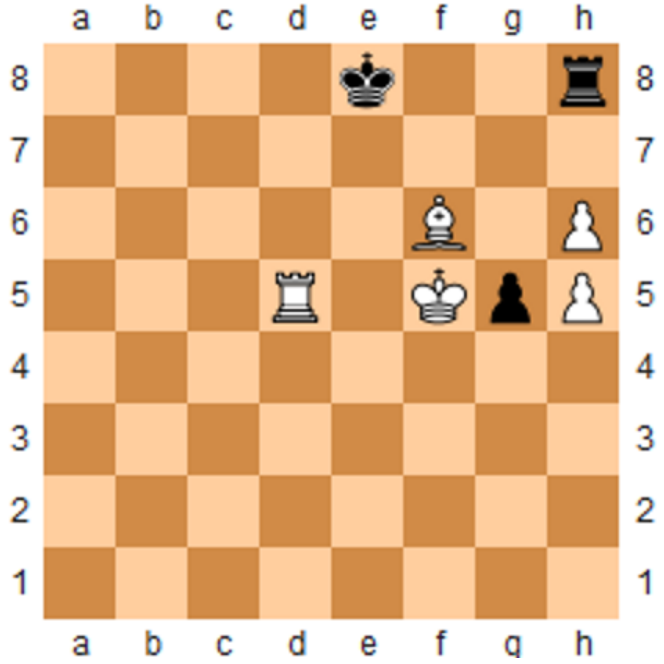
Bewertungsheuristik

- Ähnlich zur Heuristik mit der man die Restkosten schätzt
  - Aber andere Semantik, nicht durcheinander bringen!

- Beispiel Schach:
  - Dame=8, Turm=5, Läufer=3, Springer=2, Bauer=1
  - Heuristik = Summe eigene Figuren – Summe Figuren des Gegners
  - Muss erweitert werden um Stellungbewertungen, z.B. „Springer am Rand bringt Unglück und Schande“
- Terminalzustände: +/- ∞

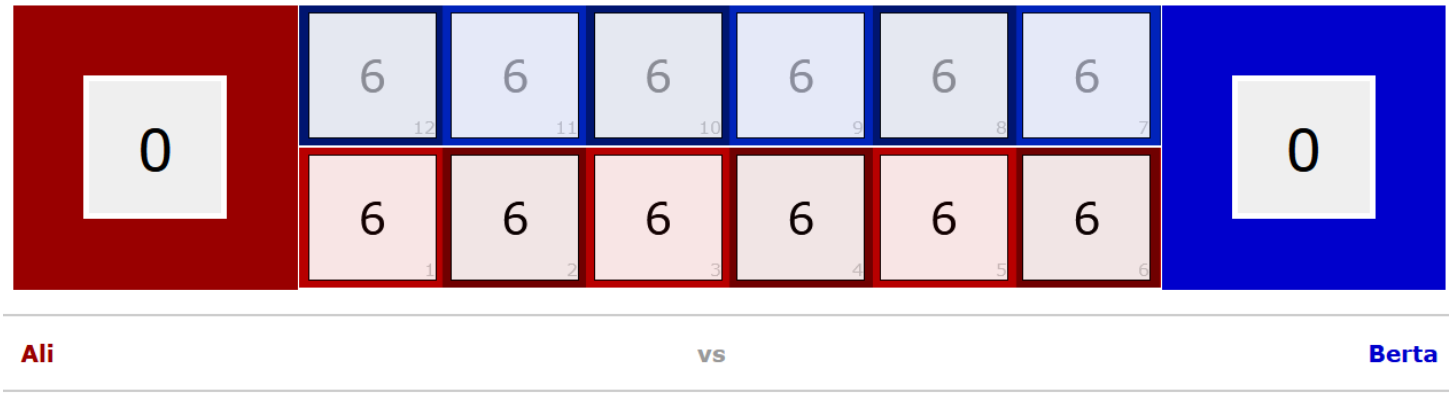
Anwendung einer Heuristik

- Beispiel
  - Weiß: 1 + 1 + 3 + 5 = 10
  - Schwarz: 5 + 1 = 6
  - Ergibt 10 - 6 = 4
- Oft keine angemessene Bewertung des aktuellen Zustandes
  - Bei komplexen Spielen schwer sinnvoll zu definieren
- Beachten: Wird auf einen Zustand einige Züge in der Zukunft angewendet!



Bohnenspiel

Erklärung via 2-Spieler-Modus im Online-Interface der Bohnen WM
http://bohnenspiel.informatik.uni-mannheim.de/



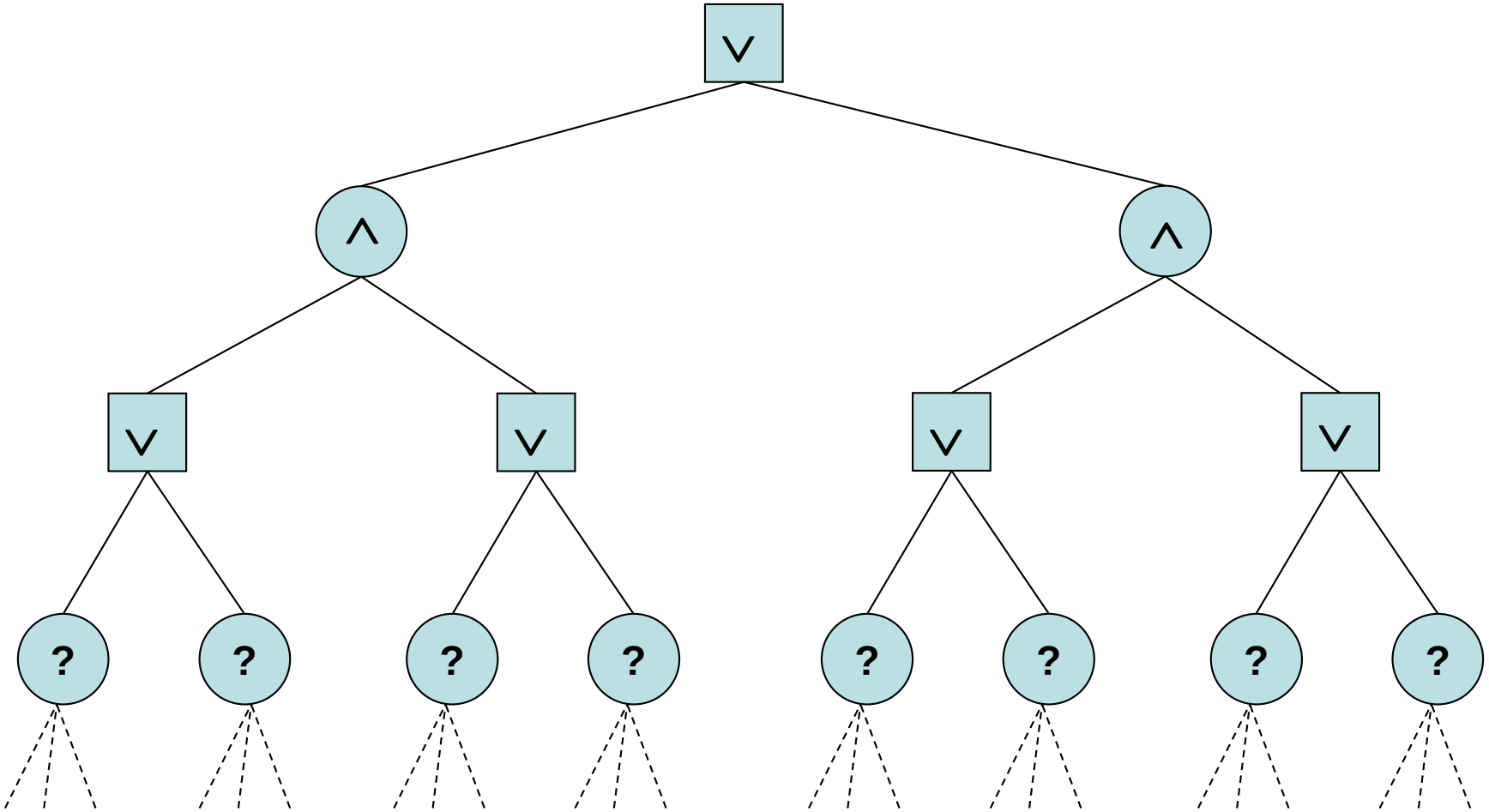
Heuristik Bohnenspiel

- Benötigt Expertenwissen

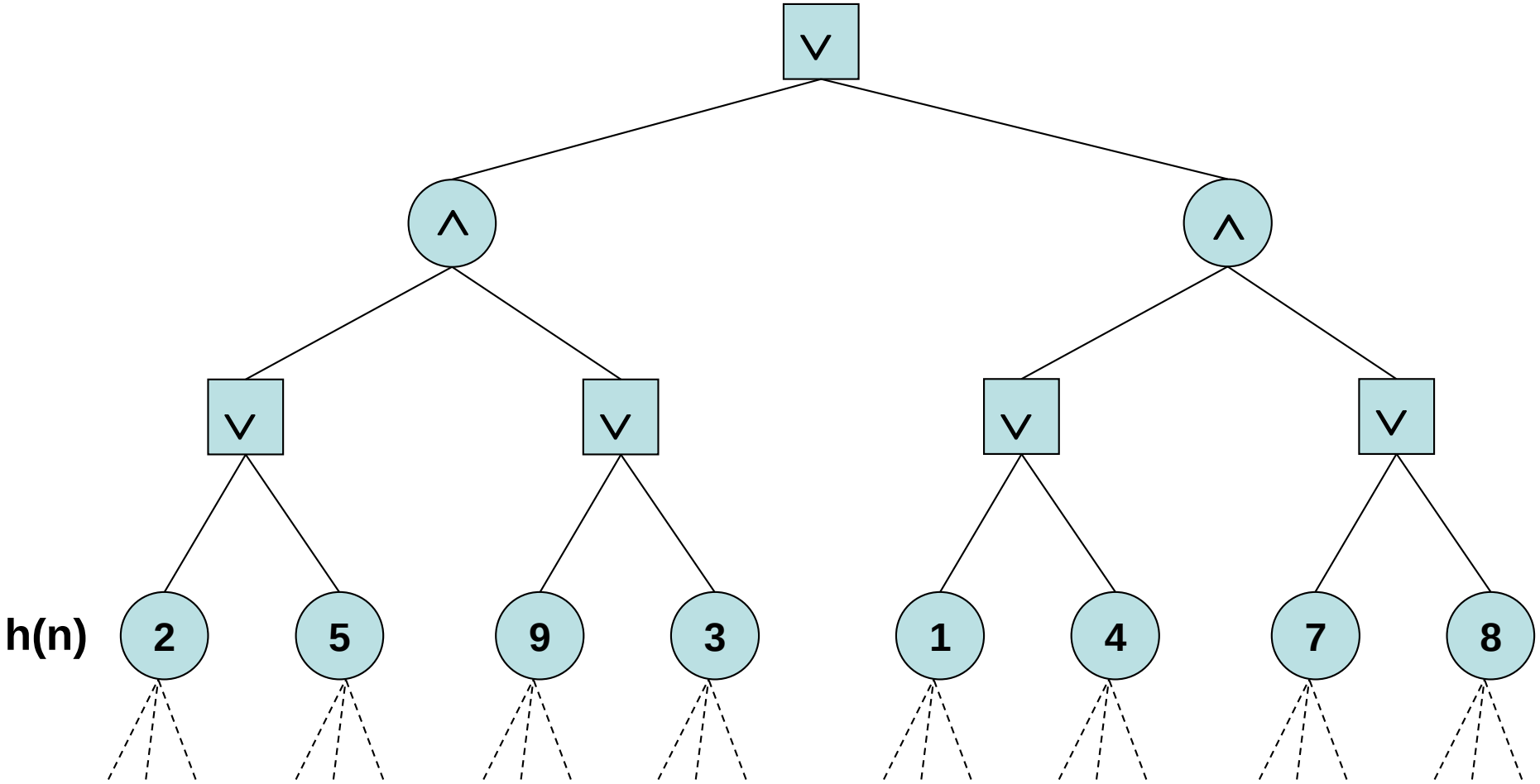
- In den letzten Jahren wurden folgende Aspekte betrachtet
  - **Vergleich der Schatzkammern**
  - Anzahl möglicher Züge (= Anzahl nicht leerer Mulden)
  - Gleichverteiltheit der Bohnen
  - Anzahl von Mulden mit 6 oder mehr Bohnen
  - ...

- Angemessene Gewichtung verschiedener Aspekte relevant, kann ja nach Spielphase variieren

Partielle Spielbäume

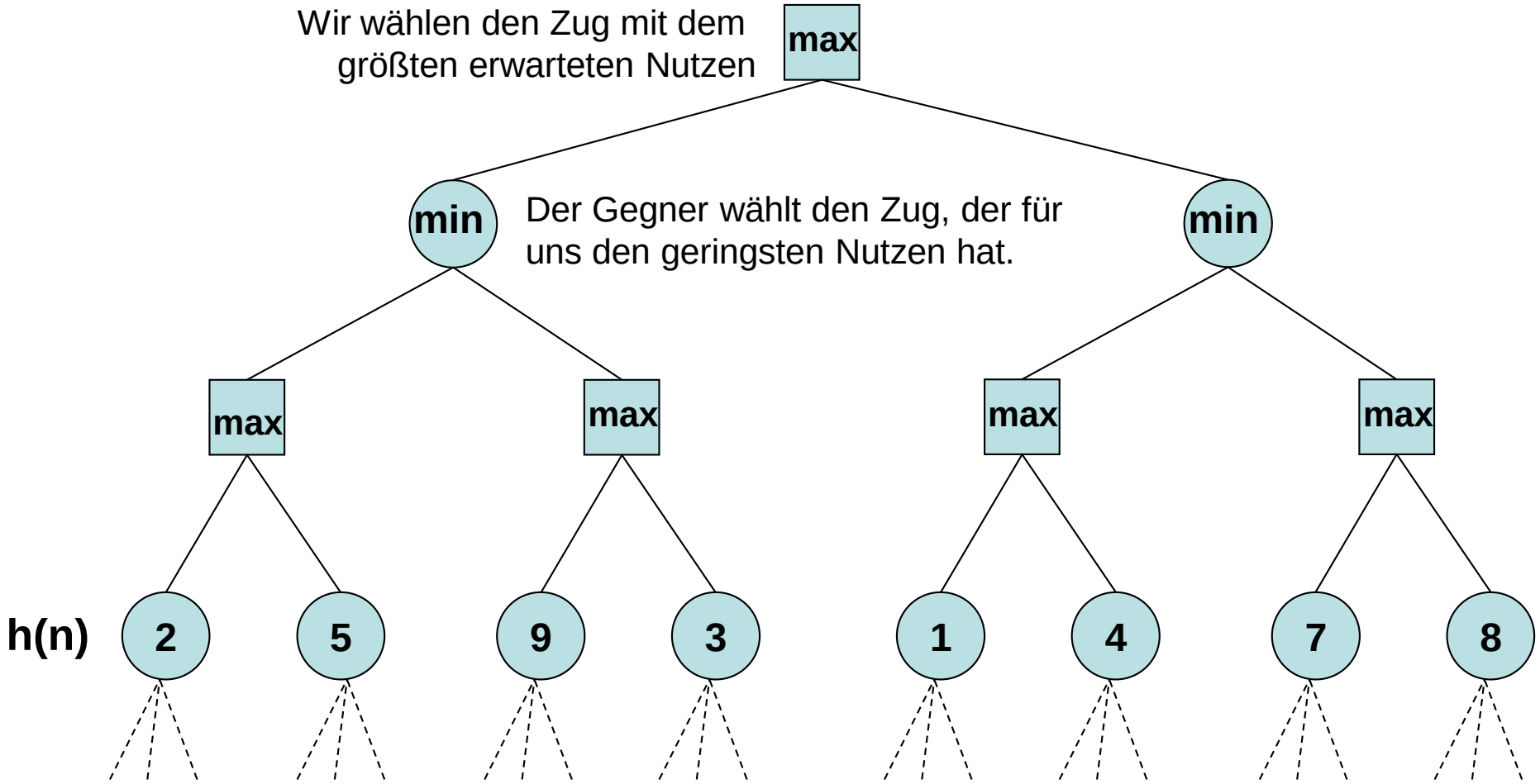


Partielle Spielbäume

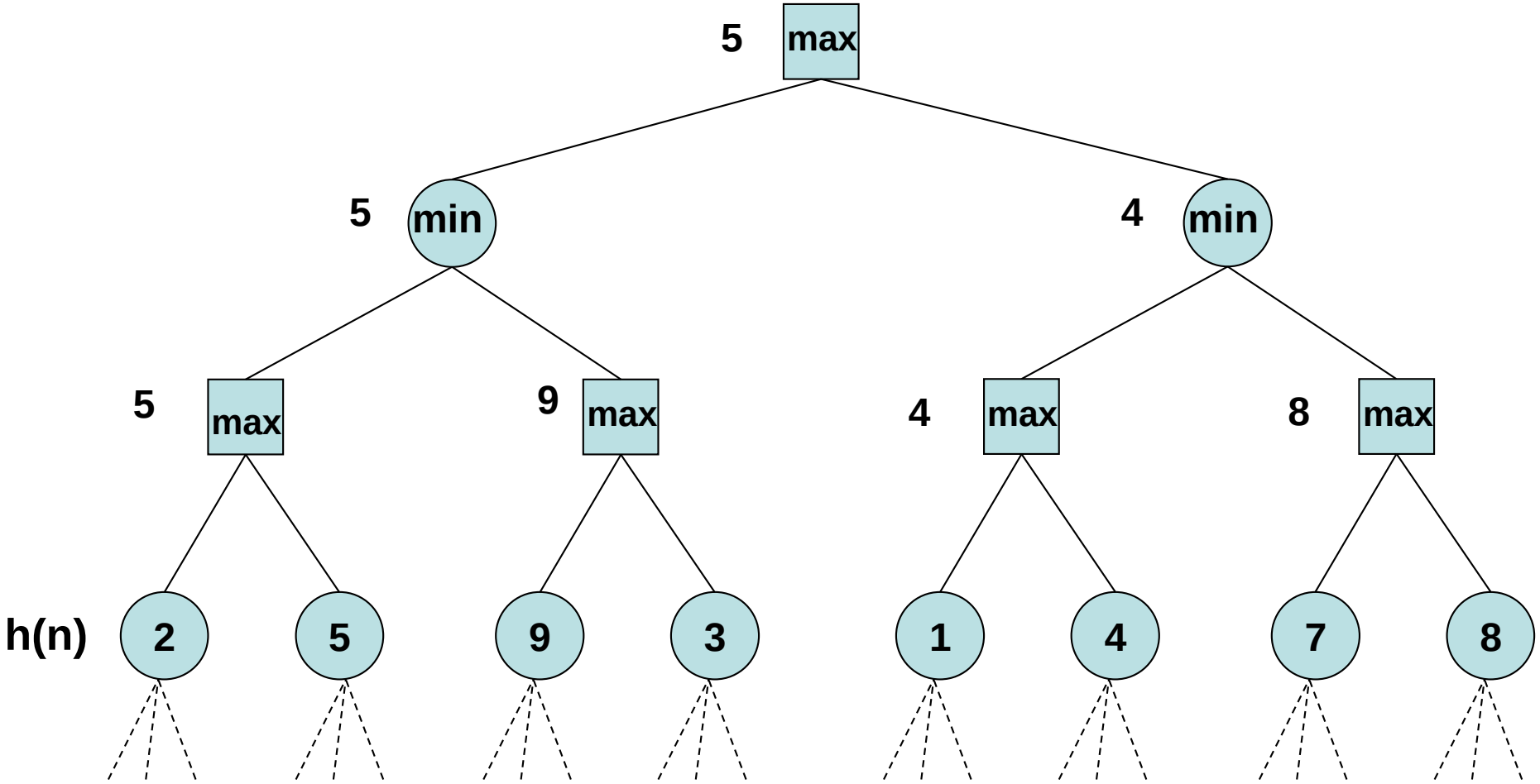


Achtung: h(n) steht hier für Nutzen, nicht für Kosten !!

Min-Max Verfahren

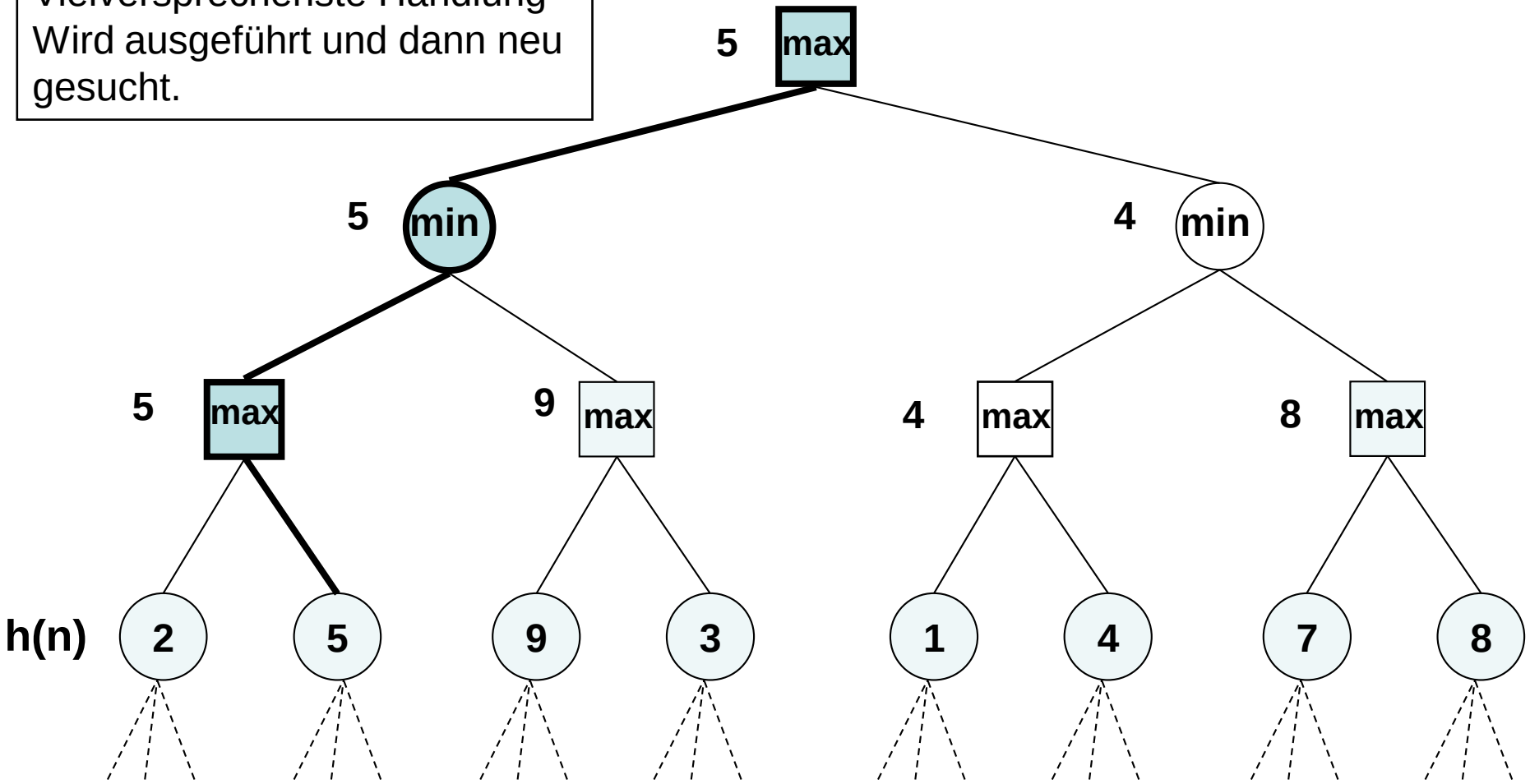


Min-Max Verfahren



Min-Max Verfahren

Vielversprechenste Handlung
Wird ausgeführt und dann neu gesucht.



Annahmen

- Wir gehen von rationalem Verhalten aus:
  - Jeder Spieler versucht, seinen Gewinn zu maximieren
  - Der Zug des Gegners minimiert den eigenen Gewinn (Nullsummenspiel)

- Optimale Strategie ergibt sich hieraus:
  - Strategie bestimmt sich durch höchsten Wert von

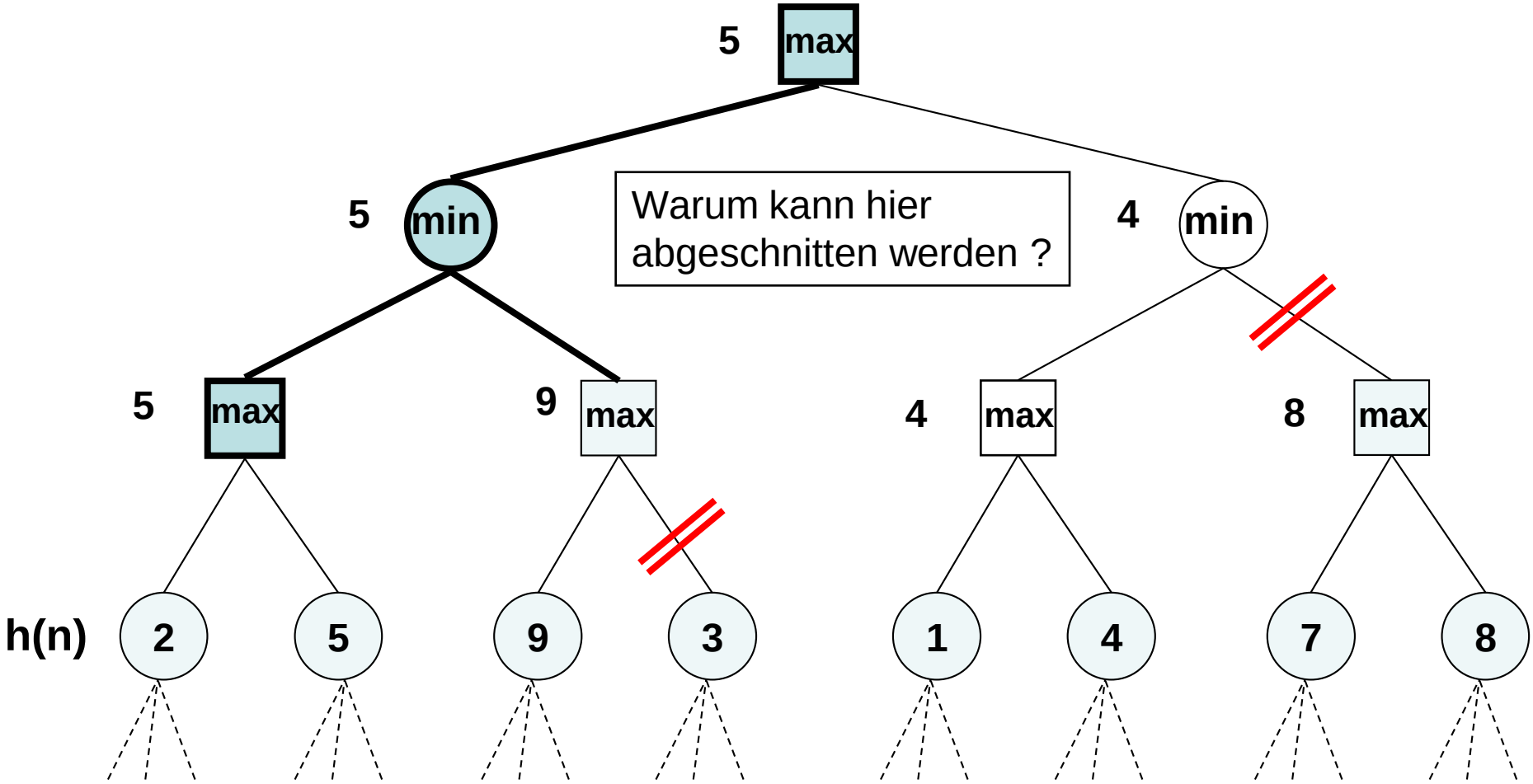
minmax(n) =	h(n) , falls n in maximaler Tiefe max{ minmax(s)| s ist Nachfolger von n} bei eigenem Zug min{ minmax(s)| s ist Nachfolger von n} bei gegnerischem Zug

Anwendbarkeit

- Komplexität ist ein Problem
  - Reale Spiele haben riesige Suchräume
    - Dame: 10^{78}, Schach 10^{120}, Go 10^{761}
  - Beispiel Schach:
    - ca 40 Zugvarianten auf beiden Seiten
    - D.h. ca 2 Mio Knoten, wenn man 2 Züge im Voraus planen will
    - Selbst Anfänger planen 3 bis 4 Züge, Profis bis zu 10
- Lösung: Einschränkung (Pruning) des Suchraums
  - Idee: Äste, die keine bessere Lösung mehr enthalten können, werden nicht mehr durchsucht
  - Variante von Branch&Bound Verfahren

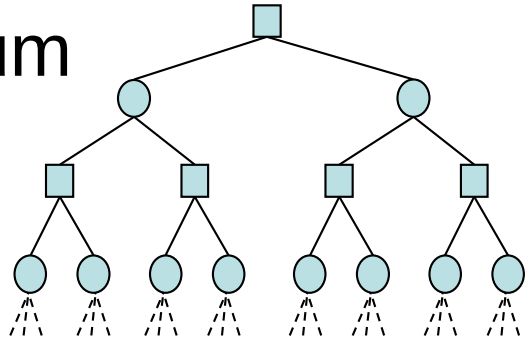
Pruningmöglichkeiten

(Erläuterung via Whiteboard)



Einfluss der Sortierung

- Suchbaum



Pruning wird maximiert, wenn immer zuerst die aus der Sicht des jeweiligen Spielers besten Züge betrachtet werden

- Vielversprechende Züge zuerst betrachten, erlaubt es mehr Äste zu prunen!

- An der Tafel:
  - Reihenfolge 3, 4, 1, 2, 7, 8, 5, 6
  - Reihenfolge 5, 6, 7, 8, 1, 2, 3, 4

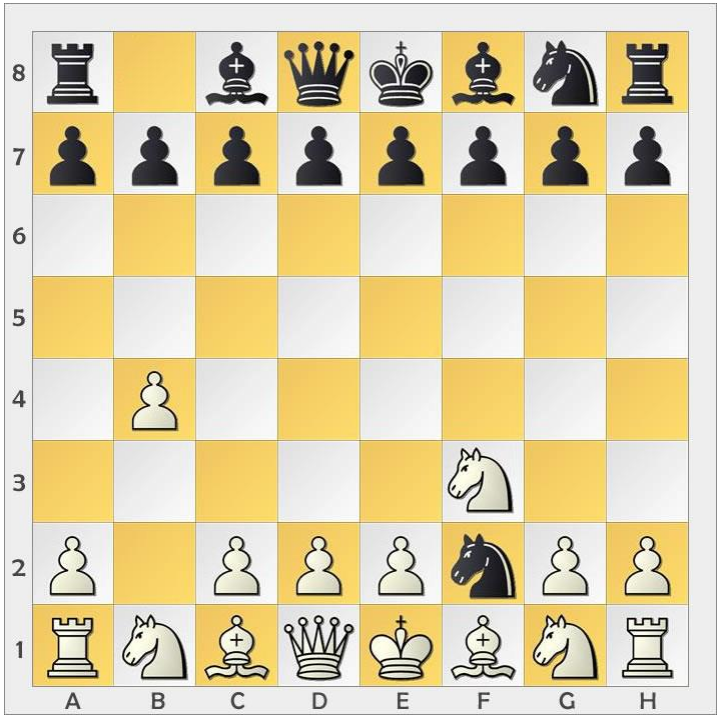
Optimierungen / Erweiterungen

- Vorsortierung der Züge
  - Führe erst vielversprechende Züge aus
  - Verschiedene Ansätze um dies umzusetzen

- Iterative Tiefensuche
  - Verwendung zur besseren Vorsortierung
  - Ergebnisse Level n bestimmt Reihenfolge auf Level n+1
- Spielspezifische Heuristik
  - Schach: Erst Züge betrachten, die Figuren schlagen
  - Bohnenspiel: Erst Züge betrachten die bedrohte Felder sichern
- Killer Heuristik
  - Guter Zug in einem Teilbaum = guter Zug in anderem Teilbaum

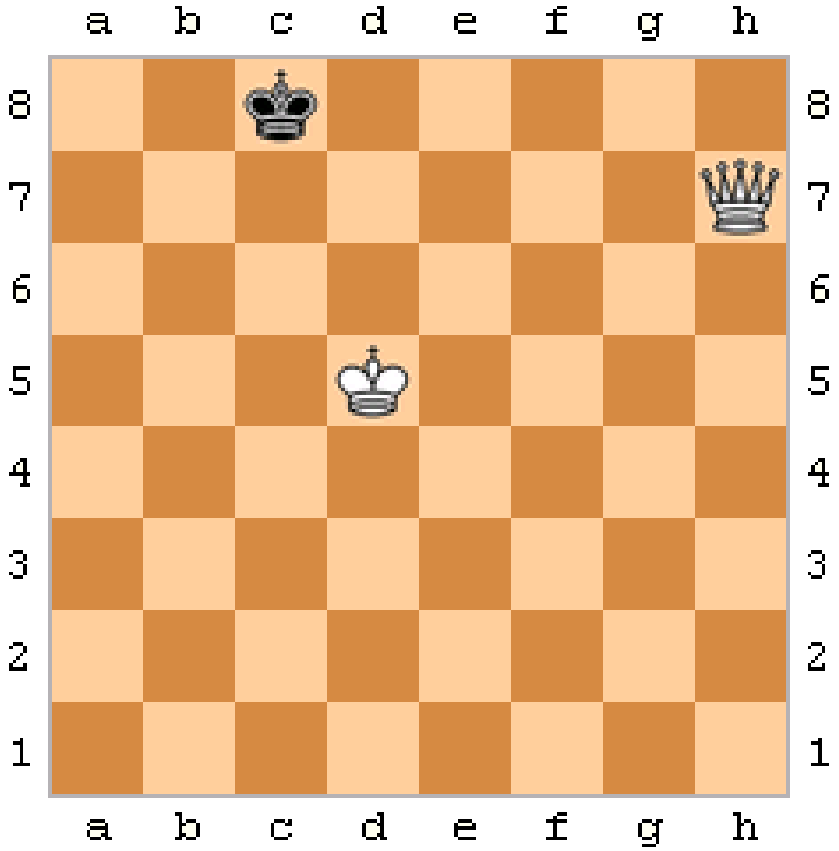
Killer Heuristik

- Weiss am Zug!
- Egal welchen Zug weiss macht, für schwarz ist es immer gut, die weiße Dame zu schlagen
- Wenn man dies für einen weißen Zug durchgerechnet hat, dann sollte man für jeden weiteren weißen Zug stets diesen Zug zuerst betrachten
- Ergibt deutlich mehr Pruning!



Endgame Tablebases

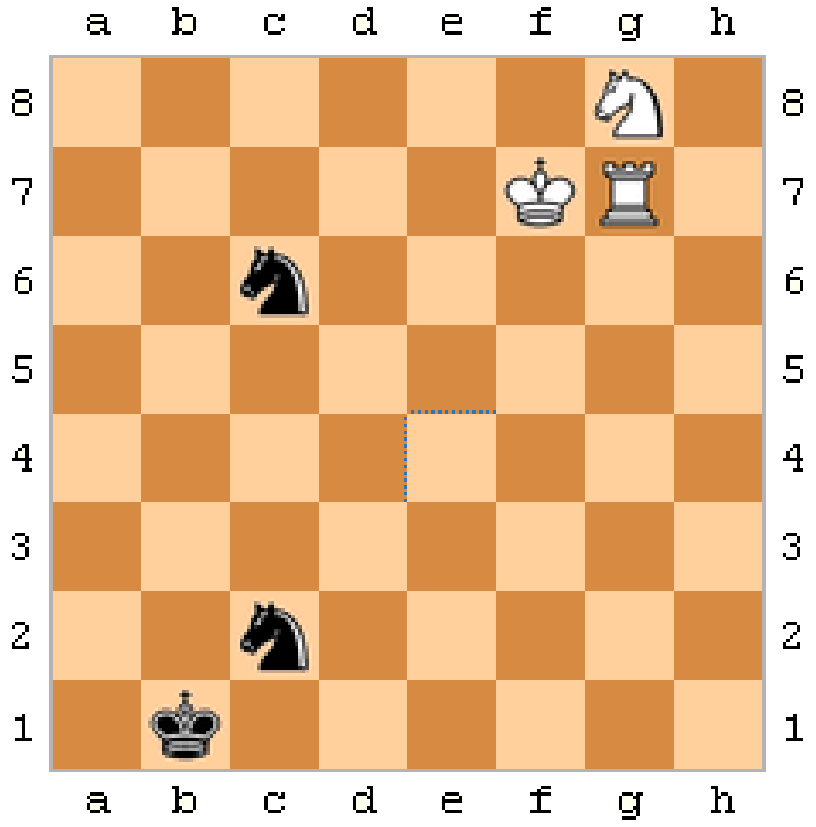
- Datenbank mit allen Positionen, für die das Ergebnis bei optimalem Spiel feststeht.
- Ist eine solche Position erreicht, muss nicht weitergesucht werden
- Für Schach gibt es eine komplette Endgame Datenbank für alle Endgames mit bis zu 6 Spielfiguren
- Variante: Positionen, die sicher zum schlagen einer Figur führen. Züge bis zum Schlagen als Bewertung



Weiss am Zug, matt in 3 Halbzügen

Endgame Tablebases

- Datenbank mit allen Positionen, für die das Ergebnis bei optimalem Spiel feststeht.
- Ist eine solche Position erreicht, muss nicht weitergesucht werden
- Für Schach gibt es eine komplette Endgame Datenbank für alle Endgames mit bis zu 6 Spielfiguren
- Variante: Positionen, die sicher zum schlagen einer Figur führen. Züge bis zum Schlagen als Bewertung



matt in 262 Zügen !!

Endgame Algorithmen

- Für Schach sind exakte Verfahren bekannt, wie man in bestimmten Situationen gewinnt
  - Zum Beispiel: Mit König und Turm (oder König, Springer, Läufer) kann man immer einen einzelnen König mattsetzen
- Erkennen (oder herbeiführen durch Suchen) einer solchen Ausgangslage
  - Kennzeichnen eines solchen Zustandes als Terminalzustand
- Deaktivieren des Suchverfahrens und aktivieren des „hart-gecodeten“ Algorithmus

- Muss problem(=spiel)-spezifisch gelöst werden

Probleme

- Automatische Erzeugung vollständiger Endgame Tables ist extrem aufwendig
  - Allerdings können hierfür die beschriebenen Suchverfahren verwendet werden
- Der Speicherbedarf steigt auch bei guter Kompression extrem an:
  - Alle Endgames mit 5 Figuren: 7 GB
  - Alle Endgames mit 6 Figuren: 1.2 Terabyte
  - 7 Figuren Endgames werden in näherer Zukunft nicht speicherbar sein
    - ob das noch stimmt ...
- Effiziente Suchverfahren können die meisten Endgames schneller selbst analysieren als sie aus der Tablebase geladen werden können!
  - Ausnahme: Endgames wie das auf der vorletzen Folie

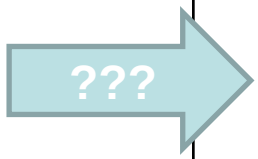
Eröffnungen

- Werden in der Regel nicht über „on-the-fly“ Suchverfahren bestimmt
  - Können vorab berechnet werden mittels bekannter Suchverfahren
  - Können aus „bekannten Partien“ entnommen werden

- Standardsuchverfahren aber geänderte Heuristik
  - „Springer am Rande bringt Unglück und Schande“

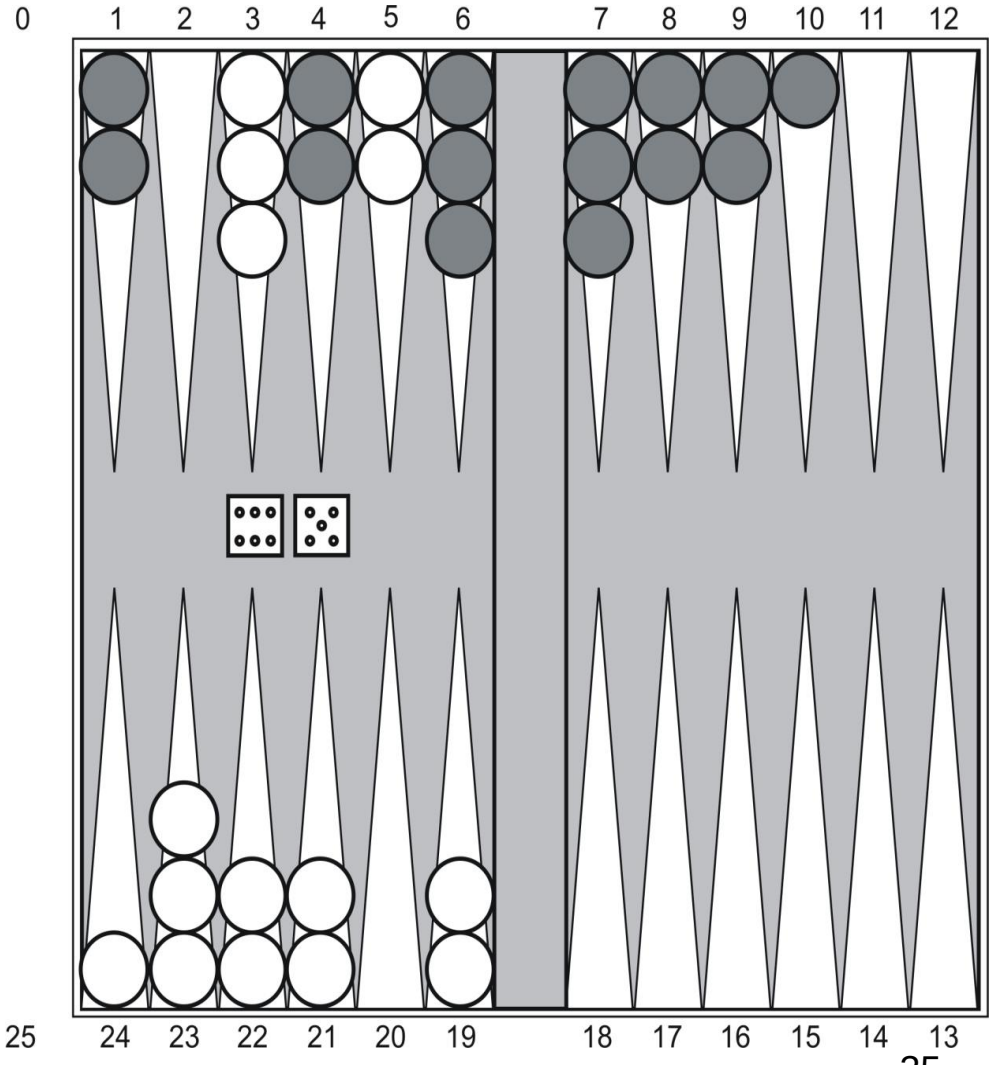
Spieltypen

	Deterministic	Stochastic
Fully Observable	Schach, Go	Backgammon, Monopoly
Partially Observable	Schiffe Versenken	Bridge, Poker



Nicht-Deterministische Spiele

- Beispiel: Backgammon
- Mögliche Züge abhängig vom Würfelergebnis
- Minimax-Baum kann nicht aufgebaut werden



Erwartetes Minimax

(Expectimax)

- Würfelaktionen werden als zusätzliche Knoten dargestellt

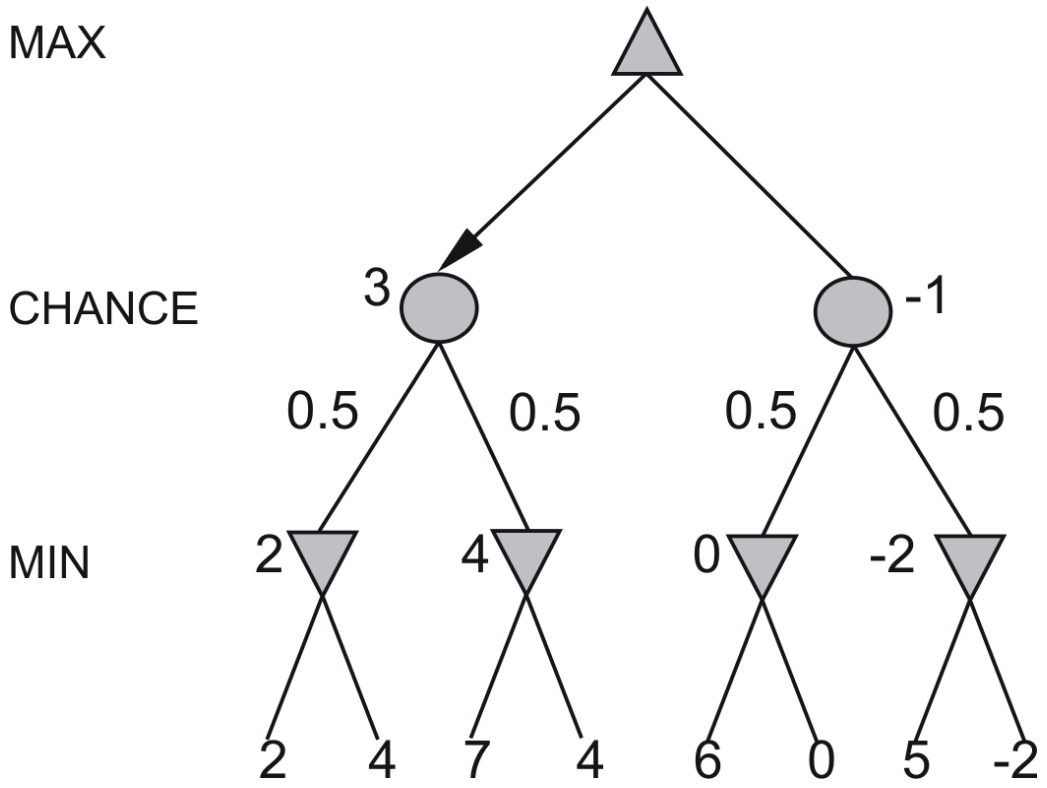
- Minimax-Werte für verschiedene Würfelergebnisse werden gemäß ihrer Wahrscheinlichkeit berücksichtigt

ExpectMM(n) =

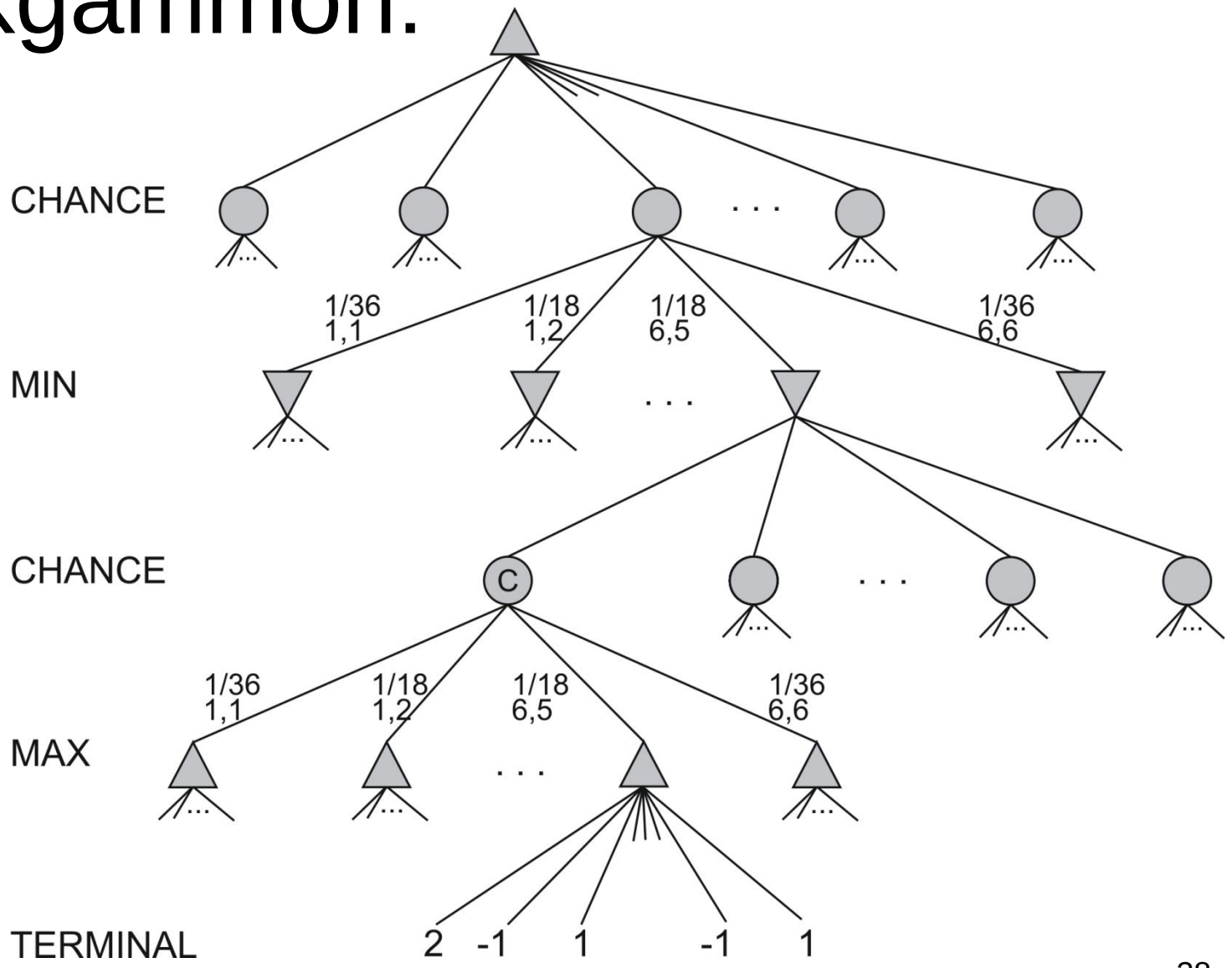
Utility(n) , falls n TERMINAL
max{ExpectMM(s) | s ist Nachfolger von n} bei eigenem Zug
min{ExpectMM(s) | s ist Nachfolger von n} bei gegnerischem Zug
sum{P(s) * ExpectMM(s) | s ist Nachforlger von n} bei Würfelaktionen

Ein einfaches Beispiel

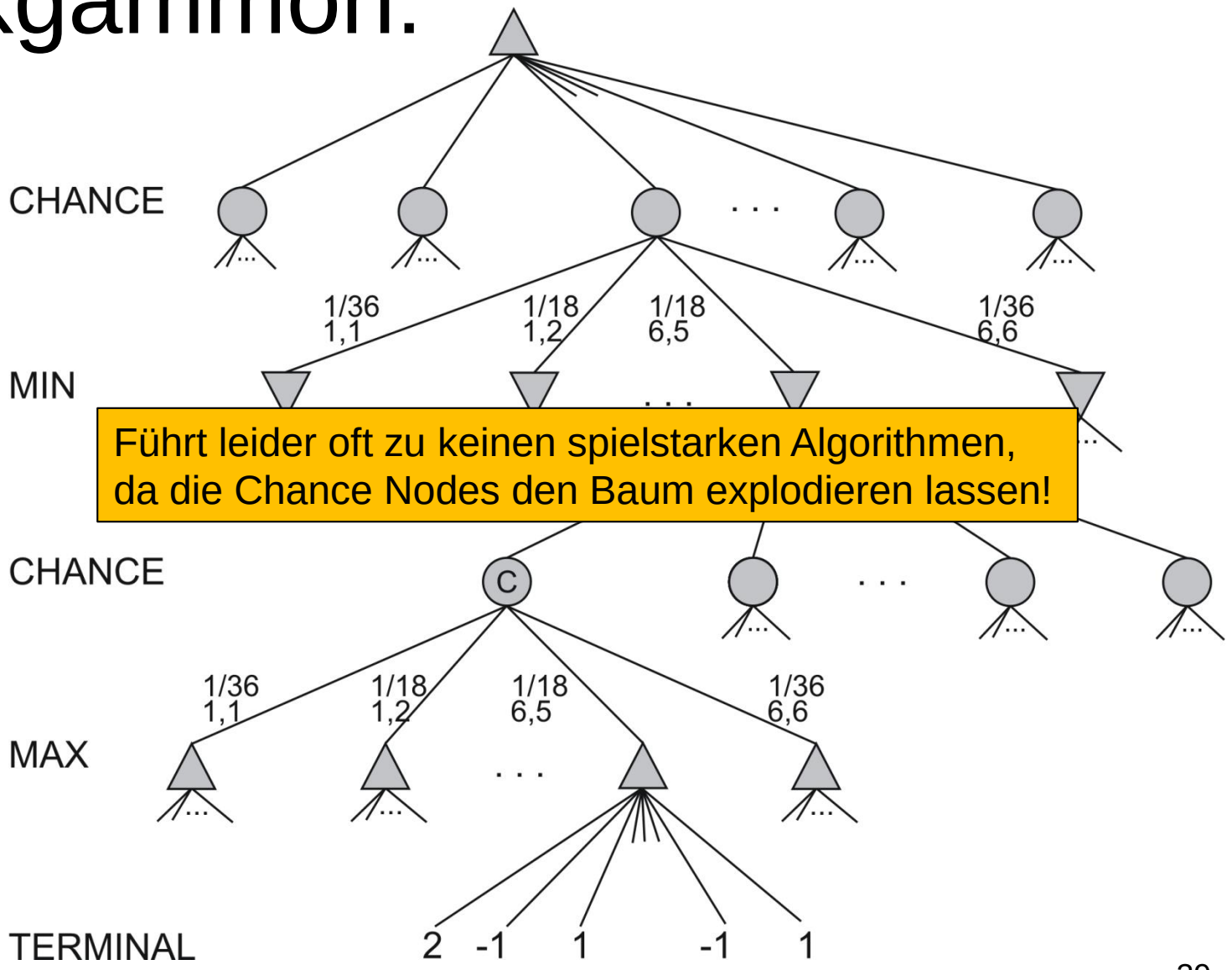
- Spiel mit Münzwurf:



Backgammon:

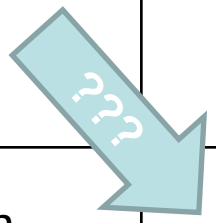

CHANCE
1/36
1,1
1/18
1,2
1/18
6,5
1/36
6,6
MIN
CHANCE
C
1/36
1,1
1/18
1,2
1/18
6,5
1/36
6,6
MAX
TERMINAL
2
-1
1
-1
1

Backgammon:


CHANCE
1/36
1,1
1/18
1,2
1/18
6,5
1/36
6,6
MIN
Führt leider oft zu keinen spielstarken Algorithmen,
da die Chance Nodes den Baum explodieren lassen!
CHANCE
C
1/36
1,1
1/18
1,2
1/18
6,5
1/36
6,6
MAX
TERMINAL
2
-1
1
-1
1

Spieltypen

	Deterministic	Stochastic
Fully Observable	Schach, Go	Backgammon, Monopoly
Partially Observable	Schiffe Versenken	Bridge, Poker



Kartenspiele

- Möglicher Ansatz:
  - Eigene Karten bekannt; (Variante von) Min-Max für alle möglichen Verteilungen berechnen
  - Mittelwert bilden und danach für einen Zug entscheiden
- Problem: Kann unmöglich für alle Verteilungen berechnet werden

- Lösung: Monte Carlo Simulation, Berechnung für einige zufällige Verteilungen

Monte-Carlo Tree Search

- Alternatives Verfahren: Monte-Carlo Tree Search
- Kann nicht nur auf Kartenspiele angewendet werden
  - Sehr erfolgreich bei Back-Gammon!
  - Aber auch sehr erfolgreich bei GO (vollständig beobachtbar)!
- Im allgemeinen anwendbar auf Spiele mit einem sehr hohen Branching Faktor
- Grundidee: Simuliere eine große Anzahl an vollständig durchgespielten Parteien und schließe aus den Ergebnissen auf den besten Zug
  - Die Simulationen sind zunächst nahezu vollständig zufällig
  - Mit jeder weiteren Iteration lernt man welche Züge sinnvoll sind
  - Folge-Iterationen konzentrieren sich auf sinnvolle Zuabfolgen

Monte-Carlo Tree Search

- Alternatives Verfahren: Monte-Carlo Tree Search
- Kann nur auf Kartenspiele angewendet werden
  - Sehr gut bei Back-Gammon!
  - Aber auch sehr gut bei GO (vollständig beobachtbar)!
- Im allgemeinen anwendbar mit einem sehr hohen Branching Faktor
- Grundidee: Simuliere eine große Anzahl an vollständig durchgespielten Parteien und schließe aus den Ergebnissen auf den besten Zug
  - Die Simulationen sind zunächst nahezu vollständig zufällig
  - Mit jeder weiteren Iteration lernt man welche Züge sinnvoll sind
  - Folge-Iterationen konzentrieren sich auf sinnvolle Zuabfolgen

Nächste Vorlesung mehr dazu!

Zusammenfassung

- Heute:
  - Spiele als Suchprobleme
  - Deterministisch und vollständig beobachtbar
    - Oder/Und Bäume
    - Heuristiken und Min-Max Bäume
    - Alpha-Beta Pruning
    - Ansätze um Alpha-Beta Pruning effektiver zu machen
    - Eröffnung und Endspiel
  - Nicht deterministische Spiele
  - Partiell beobachtbare Spiele

- Nächste Vorlesungseinheit: MCTS als Gegenstück zu dem “klassischen” Verfahren