Nachfolger durch „genetische“ Kombination

Genetische Algorithmen

Effiziente, stochastische Erzeugung von Knoten-Nachfolgern nach „Evolutions“-Prinzipien

Starte mit zufällig erzeugter Knotenmenge („Population“) und mach bis zum harten Abbruch:

- **Selektion:**
  Wähle für Erzeugung von Nachfolgern mit höherer W‘keit solche Knoten, deren VALUE („Fitness“) höher ist; **dann**:
  - **Mutation:** Verändere stochastisch eine einzelne „Stelle“ des Knotens
  - **oder** ***Crossover***: Erzeuge Nachfolger von zwei Knoten durch Kombination (Konkatenation) von Stücken ihrer Zustandsbeschreibungen

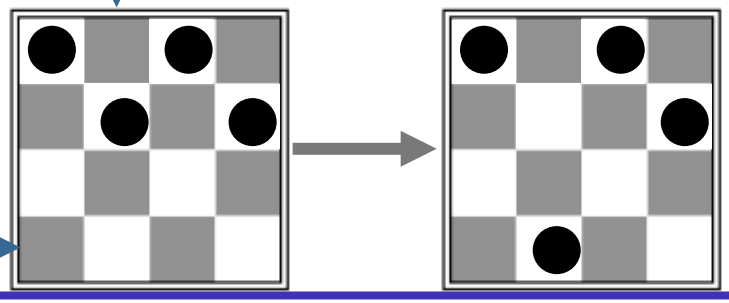
Beispiele im n-Damen-Problem

Mutation

Gegeben 1 Knoten in d. Population, …

… wähle zufällig zu verändernde Spalte

… wähle zufällig Zielwert

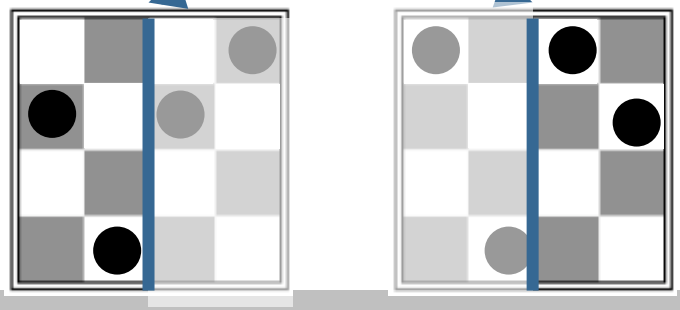


In String-Darstellung:
[1**2**12] ⇒ [1**4**12]

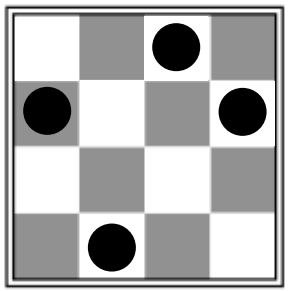
Crossover

Gegeben 2 Knoten in d. Population, …

… wähle zufällig Schnittlinie



… kombiniere beide „Eltern-Stücke“



[**24**21]+[14**12**] ⇒ [2412]

Eigenschaften Genetischer Algorithmen

- ☺ Speicherbedarf: k (Populationsgröße)
- 😐 Zeitbedarf: ??? (Zeit- oder Fitness-Schranke)
- 😐 Asymptotisch optimal
- 😐 Asymptotisch vollständig

Auf der Suche nach optimalen Lösungen

Suchprobleme im Allgemeinen sind

- unentscheidbar (unendliche Suchbäume) oder
- i.a. nicht mit polynomiellem Aufwand lösbar (NP-vollständig)

Algorithmen, die gleichzeitig effizient, optimal und vollständig sind, sind nicht zu erhoffen!

In KI und außerhalb eine Fülle weiterer Arbeiten dazu, z.B.

- online-Suche (R/N 4.5; „Lern“-Kapitel)
- Suche in Spielbäumen (Ertel 6.4; s. 3.3)
- Suche in kontinuierlichen Zustandsräumen (R/N 4.4)

➥ Optimierung (➥ Sigrid Knust)

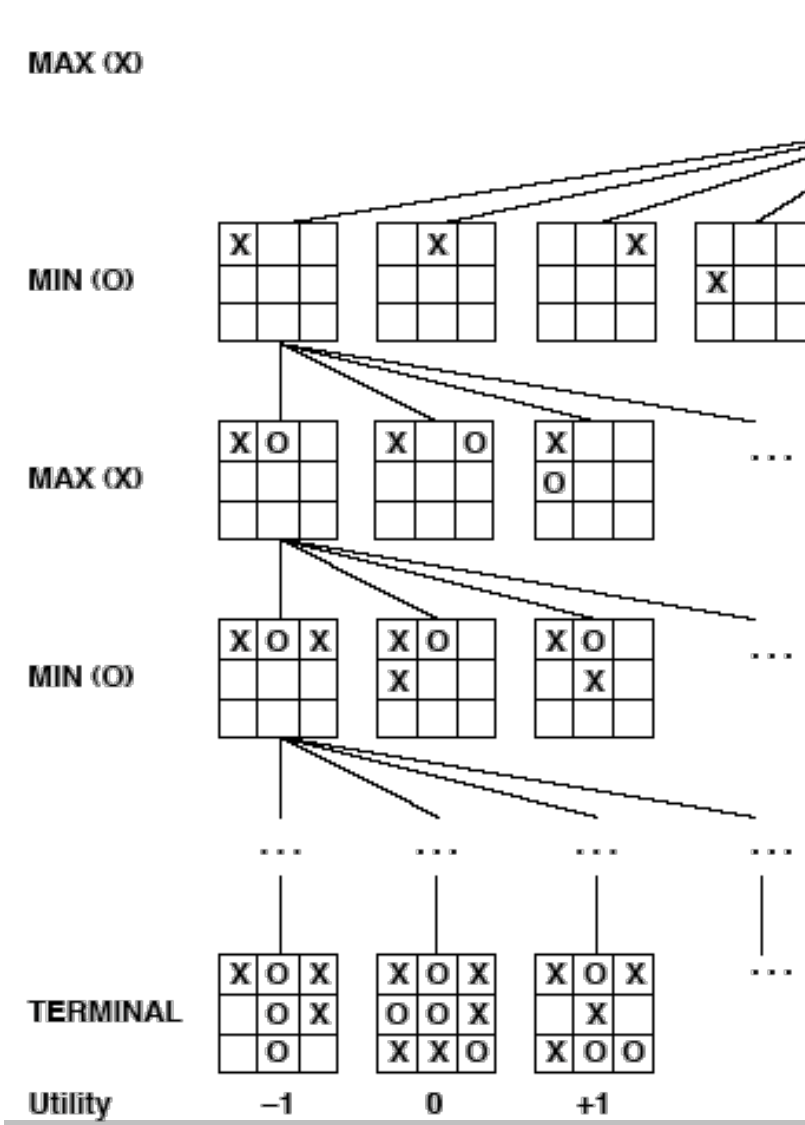
3.3 Suche in Spielbäumen

- Spiele gegen Gegner schaffen Umgebungen für Agenten
- Spiele lassen Qualität einer Agentenfunktion leicht messen
  („wer gewonnen hat, war besser“)
- Spiele kommen in vielen relevanten Strukturvarianten
  (vollständige/unvollst. Information, mit/ohne Zufall, Roboterspiele)
- Spiele können kombinatorisch komplex sein
  (Verzweigungsfaktor Schach ~35; Go anfangs 361)
- Theoriemodell weit übertragbar
  (Spieltheorie v.Neumann/Morgenstern, 1944
  in Wirtschaftswissenschaften, Handel: Nash, 1950)
- Spielprogrammierung „fasziniert“
  (Zuse, Wiener, Turing, McCarthy, Simon, …)

John v.Neumann, 1903-1957

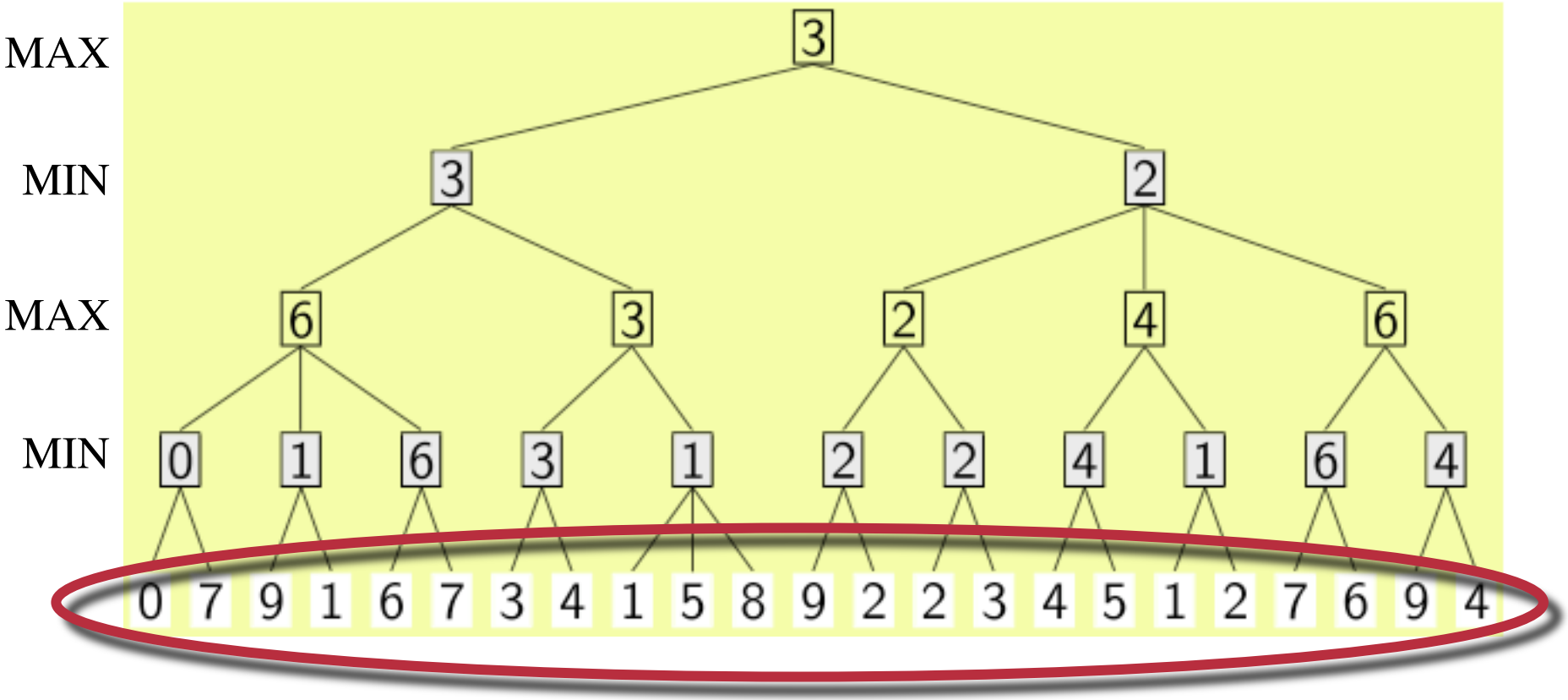
Auf der Suche nach dem nächsten Zug …

… aus Sicht von X



- Zug finden durch Suche im **Spielbaum**
- Spieler ziehen *abwechselnd*; Annahme: Beide wollen gewinnen
- Hier **Nullsummenspiele**: Was der eine gewinnt, verliert d. andere
- Agent sucht Lösung, die eigenen **Nutzen maximiert**, Gegnernutzen **minimiert**

Minimax-Suche – Das Prinzip



Terminal-Situationen,
Nutzen (*utility*) aus Sicht von MAX

- Nutzenwerte im Baum „hochgerechnet“ aus Nutzenwerte an den Blättern!

MINIMAX

```
function MINIMAX-DECISION(state, game) returns an action
    action, state ← the a, s in SUCCESSORS(s_state)
                such that MINIMAX-VALUE(s, game) is maximized
    return action

function MINIMAX-VALUE(s_state, game) returns a utility value
    if TERMINAL-TEST(state) then
        return UTILITY(s_state)
    else if MAX is to move in state then
        return the highest MINIMAX-VALUE of SUCCESSORS(state)
    else
        return the lowest MINIMAX-VALUE of SUCCESSORS(s_state)
```

Eigenschaften des MINIMAX-Verfahrens

Für maximale Suchtiefe (Vorausschau oder Spiel-Ende) m

- ☹ Zeitbedarf: $O(b^m)$ (expandiere kompletten Spielbaum)
- ☺ Speicherbedarf: $O(bm)$ (Tiefen-Traversierung)
- ☺ Vollständig (bei endlichem Baum – z.B. Schach, Go, … 😐)
- ☺ Optimal gegen einen optimal spielenden Gegner
- 😐 Nicht notwendig optimal gegen suboptimalen/anders bewertenden Gegner

Beispiel Schach

- im Schnitt b~35
- Suchtiefe m≥100 (Halbzüge, *ply*)

Merke: $35^{100} \approx 3\text{x}10^{154}$

- 1 Jahr hat ~ $3\text{x}10^8$sec
- bearbeite 10^{12} Stellungen/sec

↪ brauche 10^{134} Jahre für 1 Zug!

MINIMAX für Realisten

- Entwickle Spielbaum nur bis zu realistischem **Horizont**
- Verwende **Bewertungsfunktion** für Nicht-Terminalzustände
- Ggf. variiere Horizont (CUTOFF-TEST statt TERMINAL-TEST): Unterbrich Baumentwicklung nur in **ruhenden** (*quiescent*) Zuständen: Solchen, wo die Bewertung nur wenig schwankt

(Heuristische!) Bewertungsfunktionen

- Gewichtete Summe aus leicht erhebbaren Einzelmerkmalen
- Beispiel Schach:
  - Materialdifferenz selber/Gegner
  - Grad der Feldkontrolle (evtl. Zentrum höher wichten)
  - Beweglichkeit der eigenen Figuren
  - Bedrohung von Königs-Nachbarfeldern
  - ...

Wie weit ist ein realistischer Horizont?

- **Annahmen**:
  - 10 sec Bedenkzeit pro Zug erlaubt
  - Bewertung dominiert Berechnungszeit
  - 100.000 Bewertungen/sec möglich

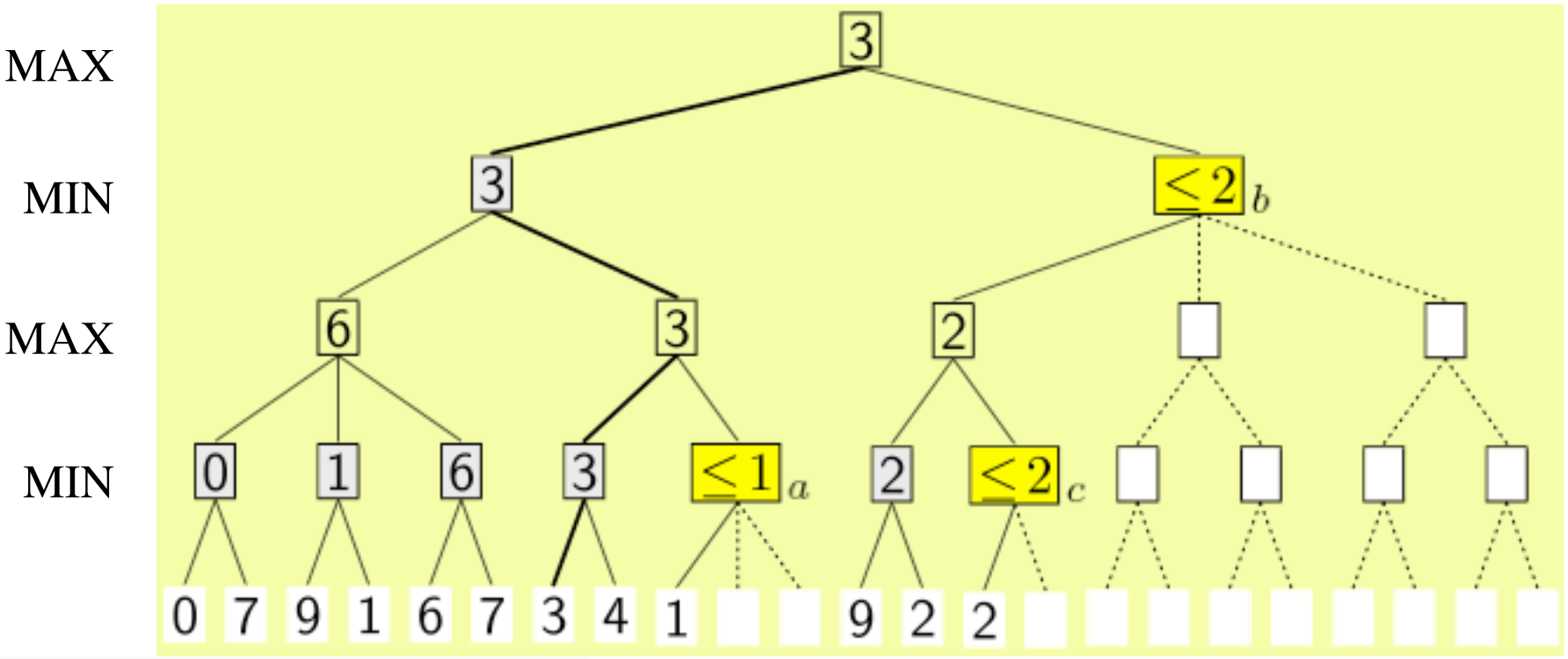
↪ 1.000.000 Zustände pro Zug analysierbar

- **Beispiel Schach** (b~35): 35^m = 1.000.000

↪ m = 3,8858

↪ In 10 sec kann man ~4 Halbzüge voraus analysieren
  - entspricht Anfänger-Spielstärke
  - starker Spieler: ~8 Halbzüge
  - Kasparow, Deep Blue (1997): ~12 Halbzüge

α–β-Schnitte, Beispiel

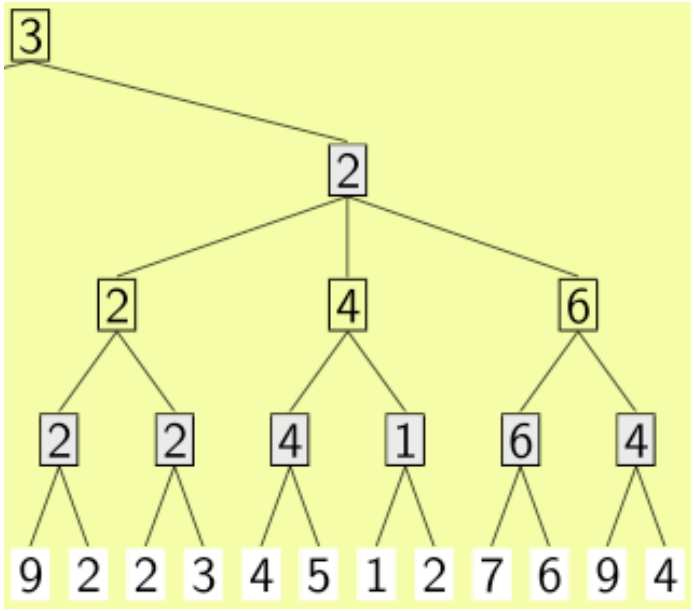


Horizont,
nicht notwendig Terminal!
Nutzen aus Sicht von MAX

Spare weitere Expansion & Bewertung

- in MIN-Knoten, wenn Wert ≤MAX darüber
- in MAX-Knoten, wenn Wert ≥MIN darüber

Was kann das bringen?



- Da die MIN-Knoten im „rechten Unterbaum“ des Beispiels „günstig“ geordnet sind:
- Nur „2“-Knoten muss generiert & evaluiert werden (danach Schnitt)!
- Nicht heuristisch, sondern korrekt!

Schnitte werden maximiert, wenn in jedem Knoten ein lokal bester Nachfolger (größter bei MAX-Knoten, kleinster bei MIN) zuerst generiert wird!

- Perfekte Reihenfolge reduziert Zeit bis auf $O(b^{m/2})$
  (Perfektion praktisch nicht erreichbar!)

➥ Horizont wird nahezu verdoppelt! (Im Schach-Beispiel auf 8)