INF FAKULTÄT FÜR INFORMATIK

Intelligente Systeme

Spiele und Erfüllungsprobleme

Prof. Dr. R. Kruse **C. Braune**

{kruse,cbraune}@iws.cs.uni-magdeburg.de

Institut für Wissens- und Sprachverarbeitung

Fakultät für Informatik

Otto-von-Guericke Universität Magdeburg

Übersicht

1. **Spiele**
   - Perfekte Spiele
   - Glücksspiele

2. **Bedingungserfüllungsprobleme**

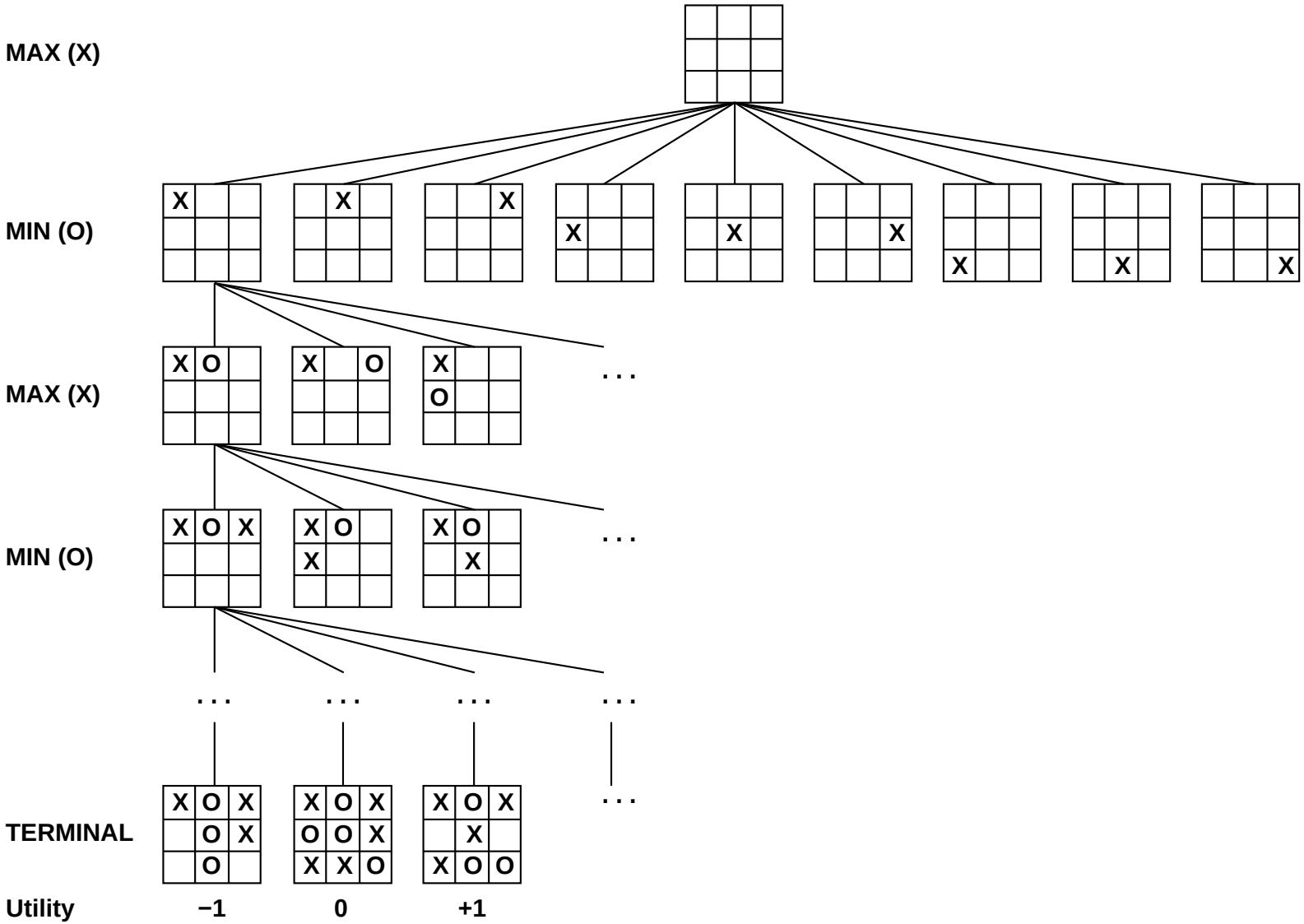
Arten von Spielen

	deterministisch	**Glücksspiel**
vollständige Informationen	Schach, Dame, Go, Othello	Backgammon, Monopoly
unvollständige Informationen	Schiffe versenken, blindes Tic-Tac-Toe	Bridge, Poker, Scrabble, Nuklearer Krieg

Blindes Tic-Tac-Toe:

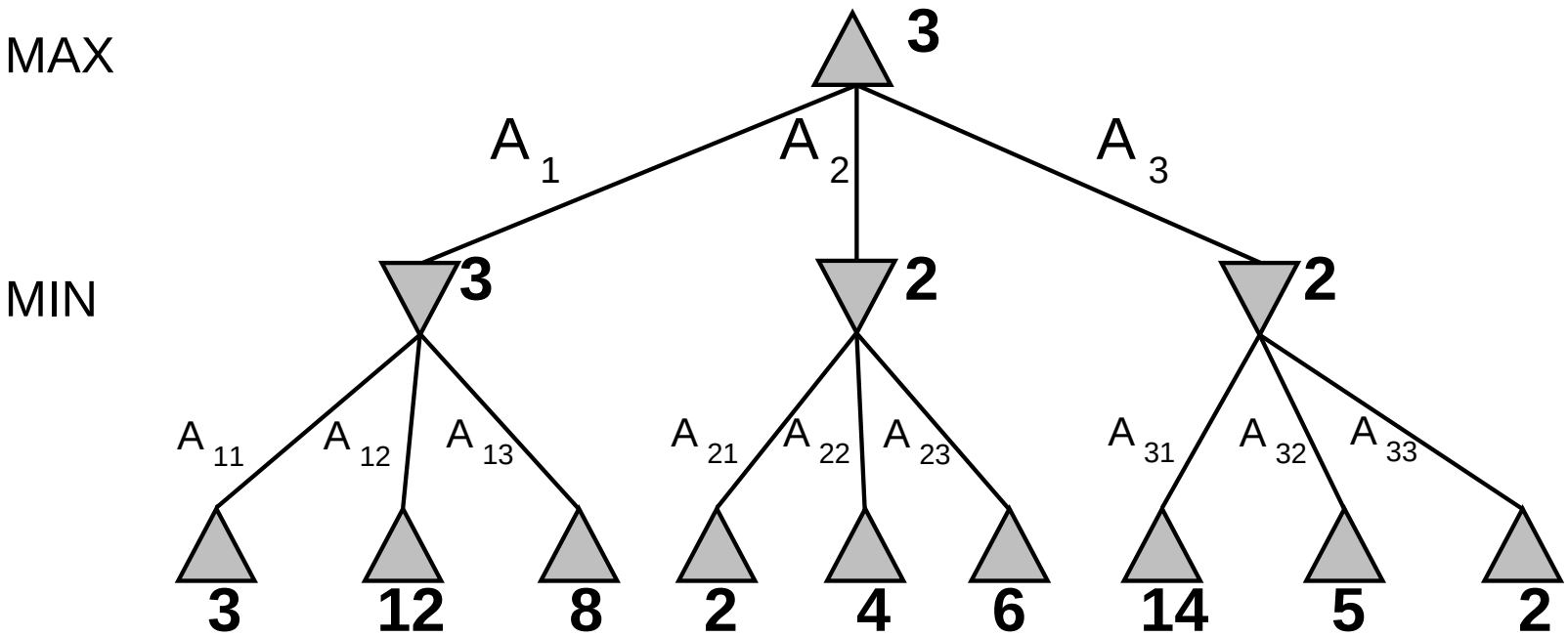
- Unvollständige Variante des Standardspiels
- Jeder Spieler kann X *und* O setzen
- Gegner erfährt nur welches Feld, aber nicht ob X oder O
- Spieler mit erster Linie mit 3 gleichen Zeichen gewinnt

Spielbaum (2 Spieler, deterministisch, Runden)



Minimax-Algorithmus

Perfektes Spiel für deterministische Spiele mit vollständigen Infos
Algorithmus zur Ermittlung der optimalen Spielstrategie für endliche Zweipersonen-Nullsummen-Spiele mit vollständiger Information
z.B. 2-schichtiges Spiel:



MiniMax: Eigenschaften

Zeit- und Speicherkomplexität gemessen anhand von

b: maximalem Verzweigungsfaktor des Suchbaums
m: maximaler Tiefe des Zustandsraums (eventuell ∞)

Vollständig: ja, falls Baum endlich

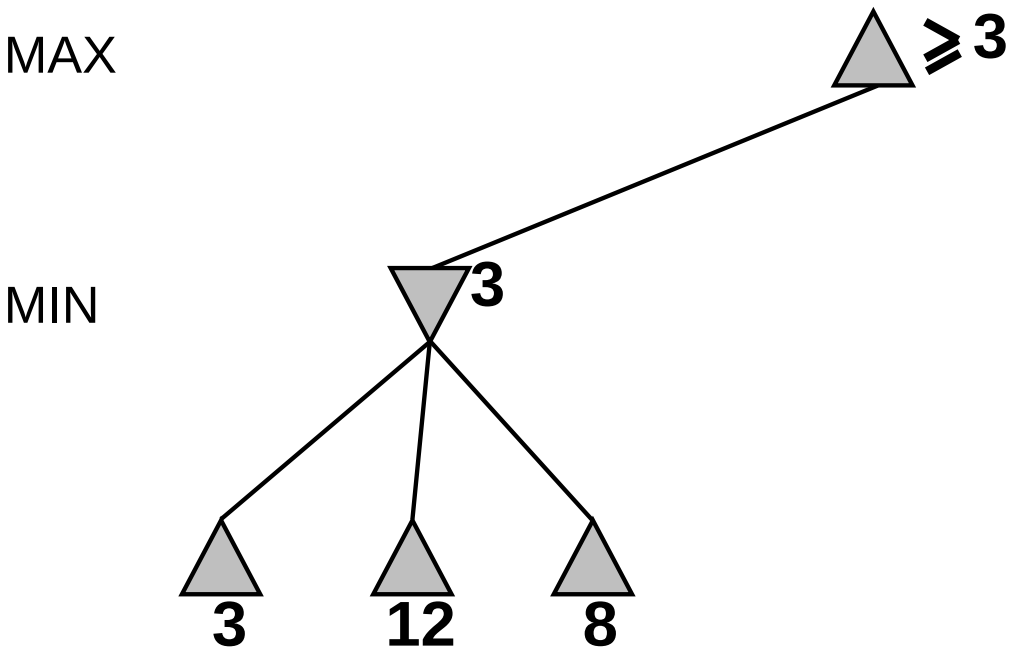
Optimal: ja, gegen optimalen Gegner

Zeit: $O(b^m)$

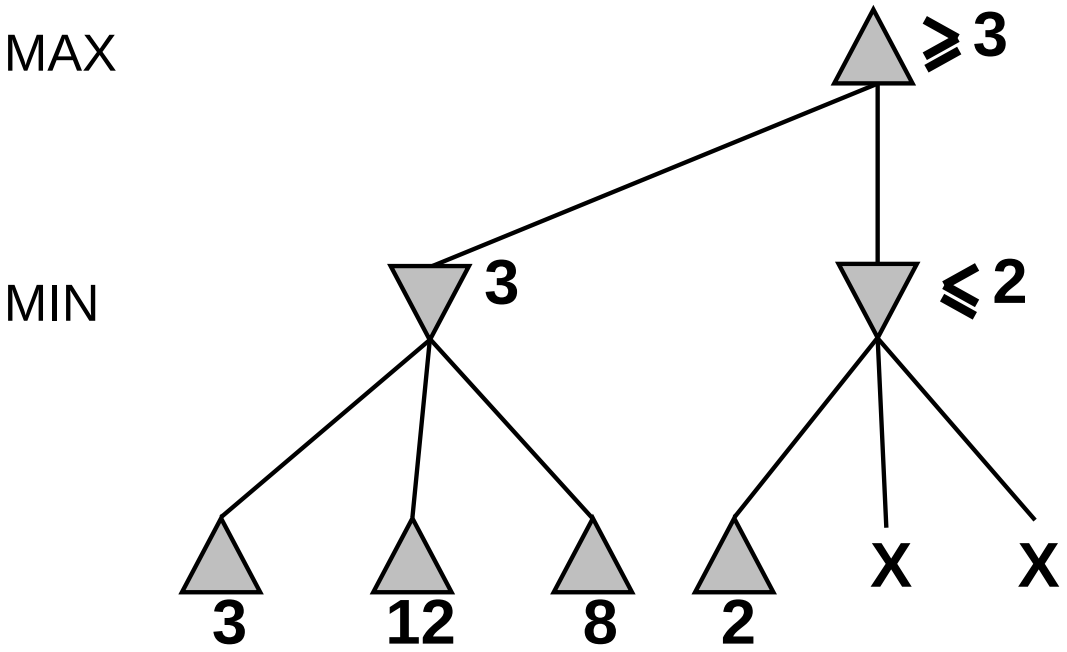
Speicher: $O(b \cdot m)$ (Tiefensuche)

Für Schach: $b \approx 35, m \approx 100$ bei „realistischen“ Spielen.
Somit ist die exakte Lösung absolut nicht berechenbar.
Aber: muss jeder Pfad exploriert werden?

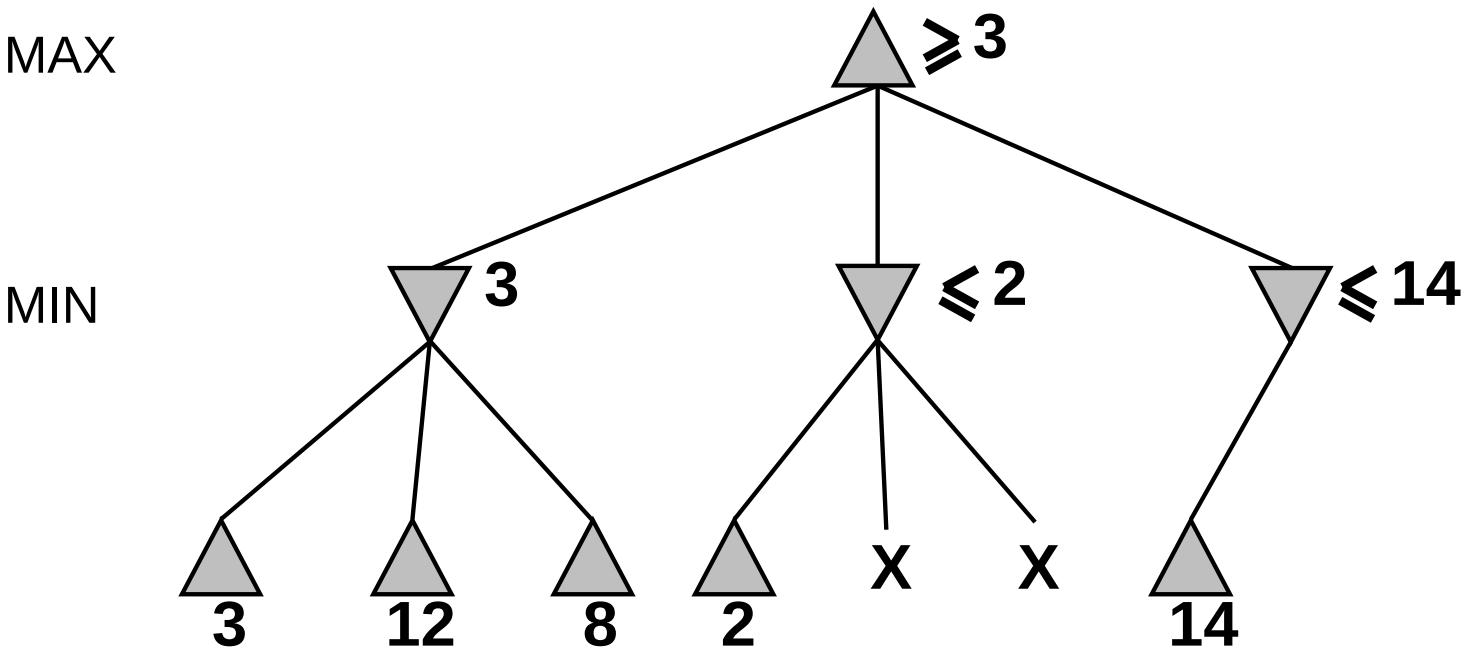
α–β-Stutzen



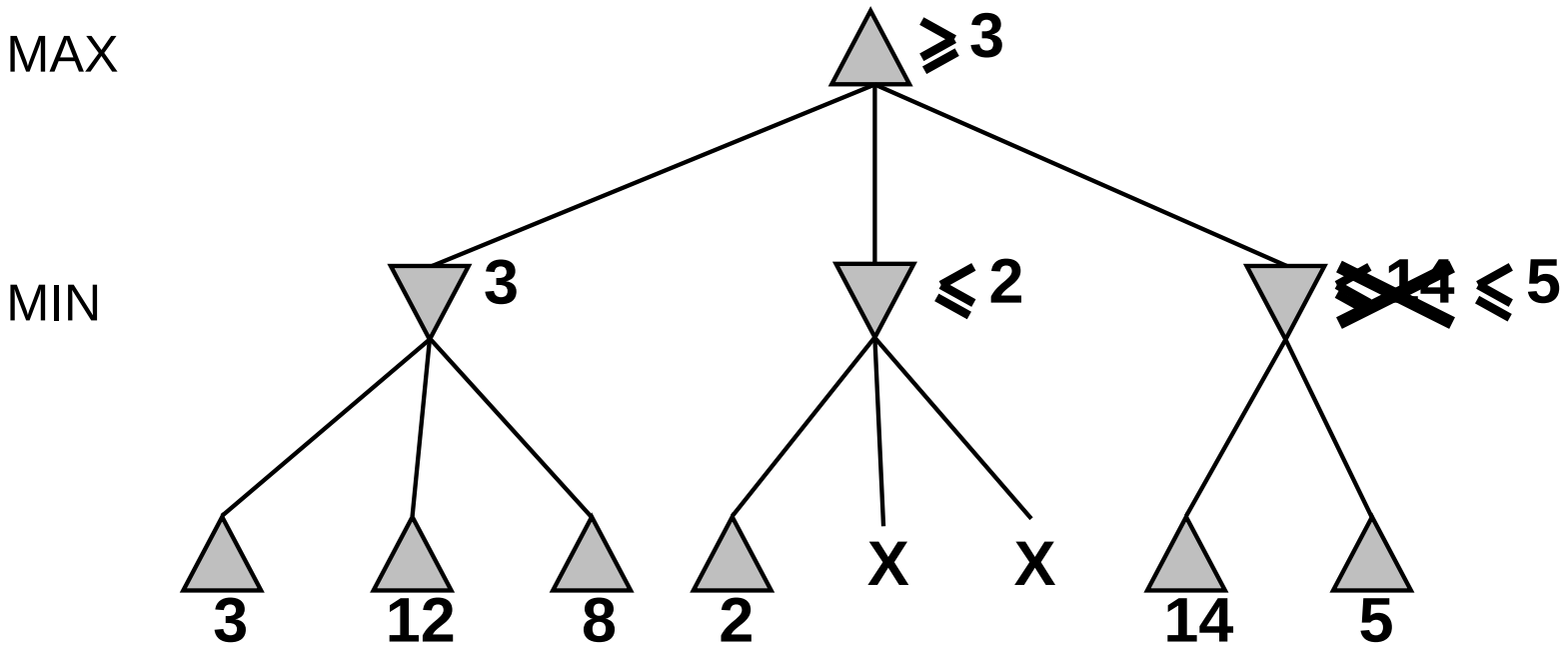
α–β-Stutzen



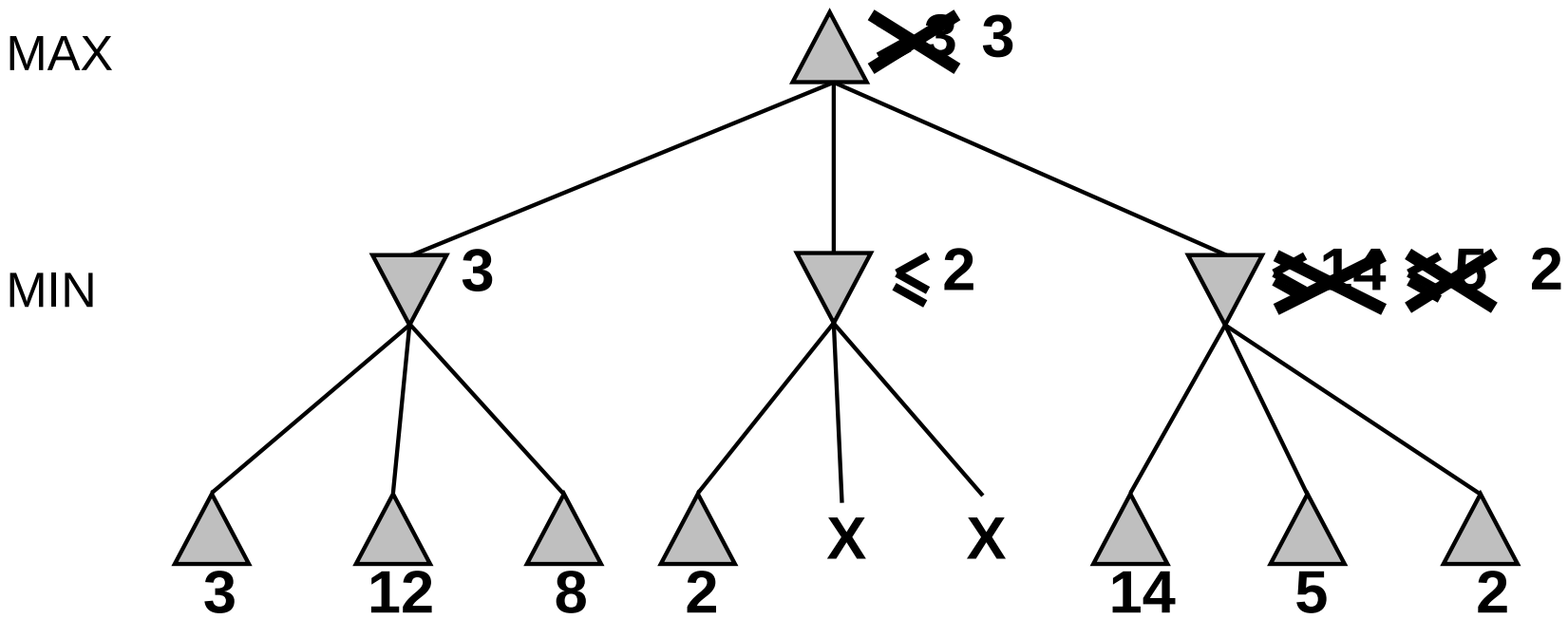
α–β-Stutzen



α–β-Stutzen



α–β-Stutzen



α–β-Algorithmus: Eigenschaften

Stutzen hat **keine** Auswirkung auf Endergebnis

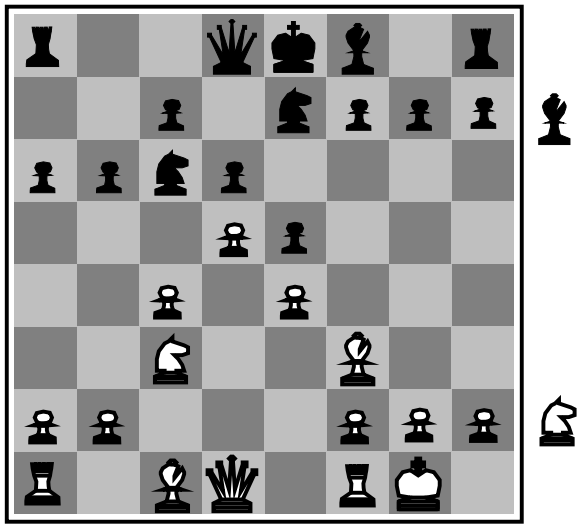
Gute Zugordnung verbessert Effektivität des Stutzens.

Mit „perfekter“ Ordnung, Zeitkomplexität $= O(b^{m/2})$

Somit doppelte Suchtiefe

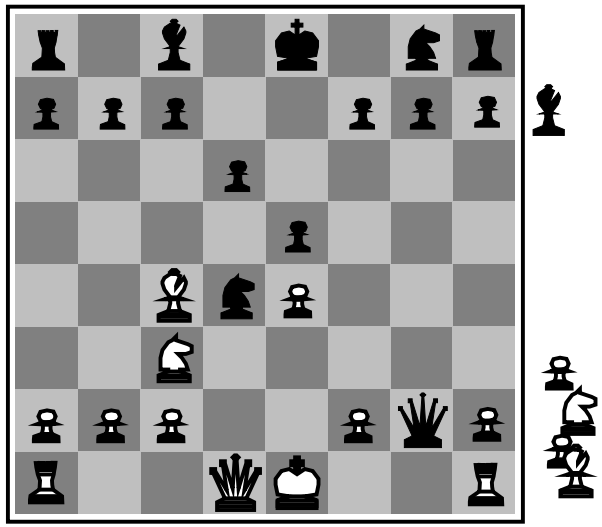
Für Schach: immer noch 35^{50} möglich

Bewertungsfunktionen



Black to move

White slightly better



White to move

Black winning

Für Schach: typischerweise linear gewichtete Summe von n **Merkmalen**

$$\text{Eval}(s) = \sum_{i=1}^{n} w_i \cdot f_i(s)$$

Deterministische Spiele

Dame:

- 1994 beendete Chinook die 40-jährige Herrschaft des Weltmeisters Marion Tinsley
- Endspieldatenbank mit perfekten Spielen aller Stellungen mit ≤ 8 Steinen ($\geq 443 \cdot 10^9$ Stellungen)

Schach:

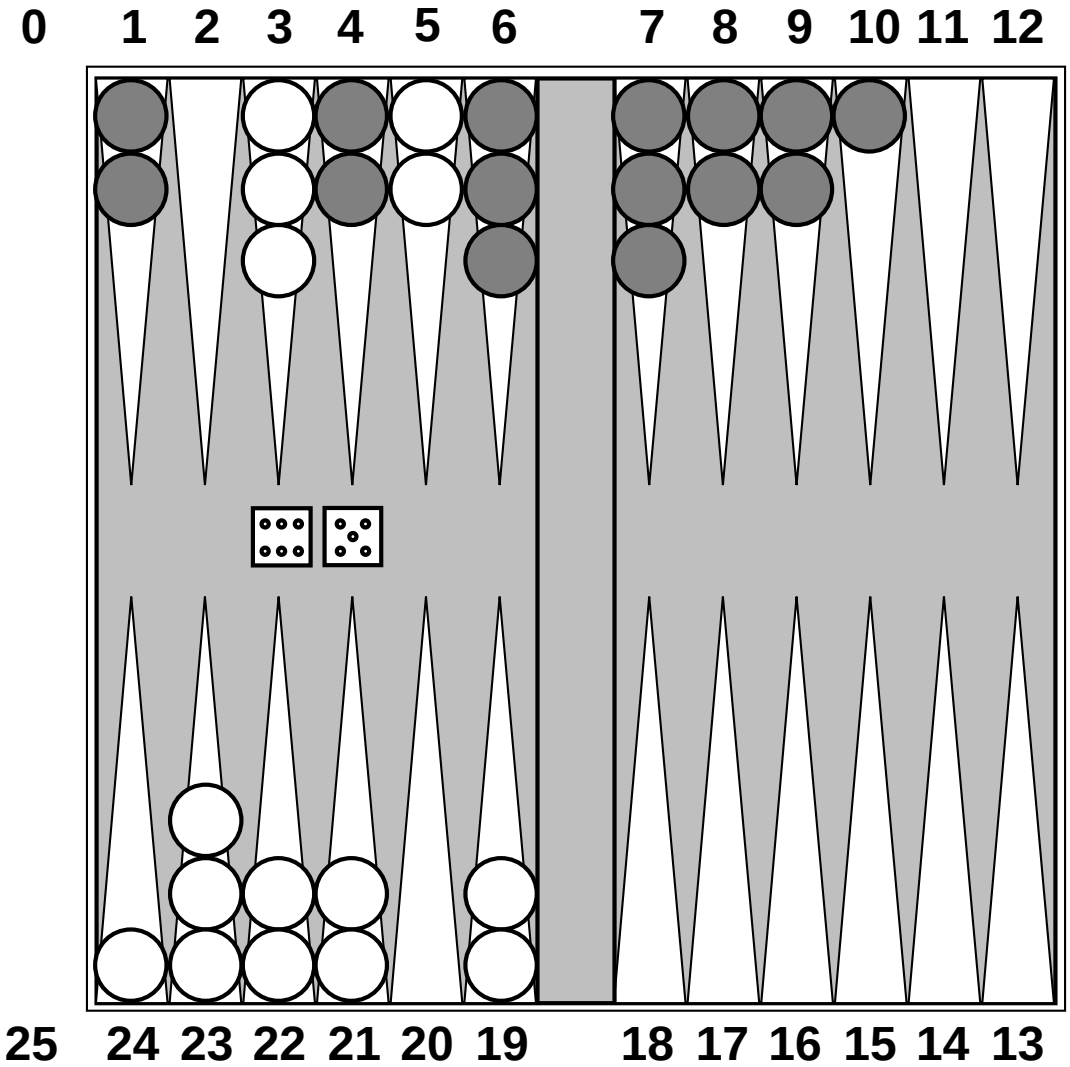
- Deep Blue besiegte 1997 Weltmeister Garri Kasparow in 6 Spielen
- $200 \cdot 10^6$ Stellungen/Sekunde, sehr komplizierte Bewertung
- Bis zu 40 Halbzüge tief (nicht veröffentlichte Methoden)

Othello: Wettbewerb nur unter menschlichen Meistern

Go:

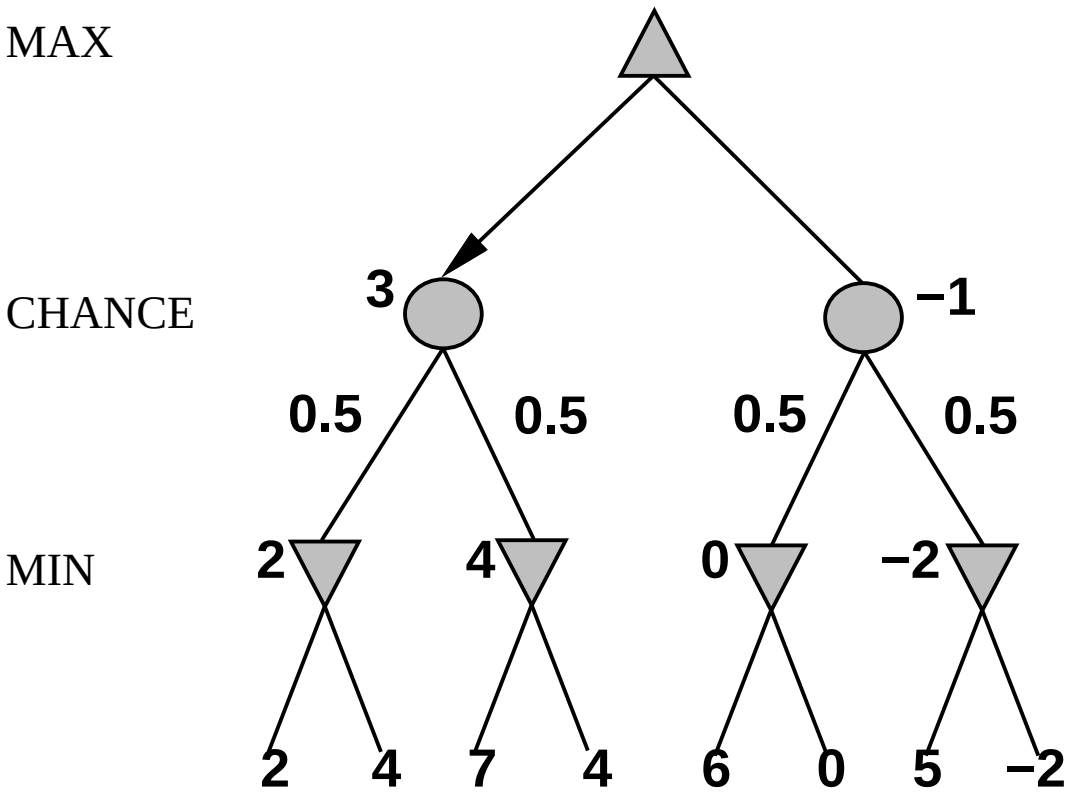
- Wettbewerb nur unter menschlichen Meistern
- $b > 300$, daher Datenbanken mit Mustern für plausible Züge

Glücksspiele: Backgammon



Glücksspiele allgemein

Zufall aufgrund von Würfeln, Mischen von Karten, etc.
Vereinfachtes Beispiel mit Münzwurf:



Algorithmus für nichtdeterministische Spiele

ExpectiMinimax liefert perfektes Spiel

Wie Minimax, nur Zufallsknoten müssen behandelt werden:

```
...
if state is a Max node {
    return highest ExpectiMinimax-Value of Successors(state)
}
if state is a Min node {
    return lowest ExpectiMinimax-Value of Successors(state)
}
if state is a chance node {
    return average of ExpectiMinimax-Value of Successors(state)
}...
```

Nichtdeterministische Spiele in Praxis

Jeder Würfelwurf erhöht Verzweigungsfaktor b: 21 mögliche Würfe mit 2 Würfeln

Backgammon $\approx$ 20 erlaubte Züge (bei einem 1–1 Wurf können es 6000 sein)

$$\text{Tiefe } 4 \rightarrow 20 \cdot (21 \cdot 20)^3 \approx 1,2 \cdot 10^9 \text{ Zustände}$$

Mit steigender Tiefe: Wahrscheinlichkeit sinkt geg. Knoten zu erreichen

Damit wird Wert des Vorgriffs vermindert

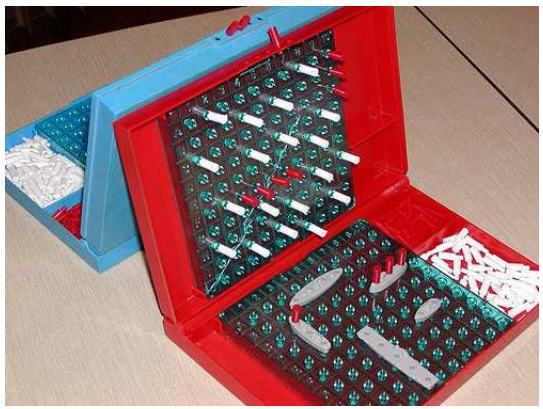
α–β-Stutzen viel weniger effektiv

TDGammon benutzt beschränkte Tiefensuche mit $d = 2 +$ sehr gute Eval: vergleichbar mit menschlichem Weltmeister

Spiele mit unvollständigen Informationen

Die meisten Kartenspiele (wie Bridge, Doppelkopf, Hearts, Mau-Mau, Poker, Siebzehn und vier, Skat) sind Nullsummenspiele mit unvollständigen Informationen.

Auch einige Brettspiele (Schiffe versenken, Kriegspiel-Schach).

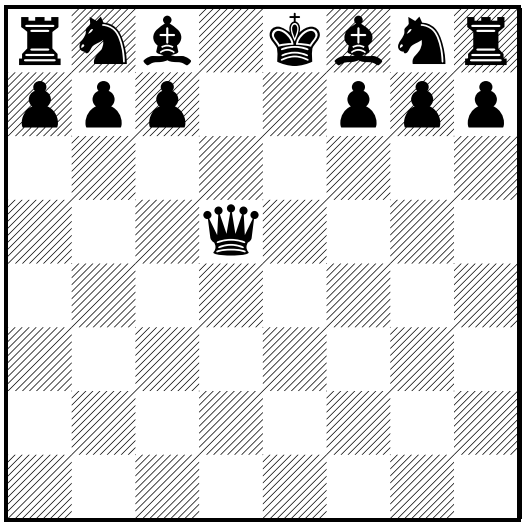


Nullsummenspiel: Spiel, bei dem $\sum$ der Gewinne und Verluste aller Spieler $= 0$

Beispiel: Kriegspiel-Schach

Unvollständige Variante des Schach

- 1824 entwickelt von einem Preußischen Militäroffizier
- Wurde bekannt als Aufgabe fürs Militär-Training
- Vorläufer von modernen Militärkriegsspielen



Ähnlich einer Kombination aus Schach und Schiffe versenken

- Figuren starten in normaler Position, aber man sieht die Züge seines Gegners nicht

Nur wie folgt kommt man an Informationen:

- Man schlägt eine Figur oder verliert eine
- Eigener König wird Schach gesetzt
- Man macht einen ungültigen Zug

Beispiel: Kriegspiel-Schach

Jeder Spieler versucht normale Schachzüge auszuführen

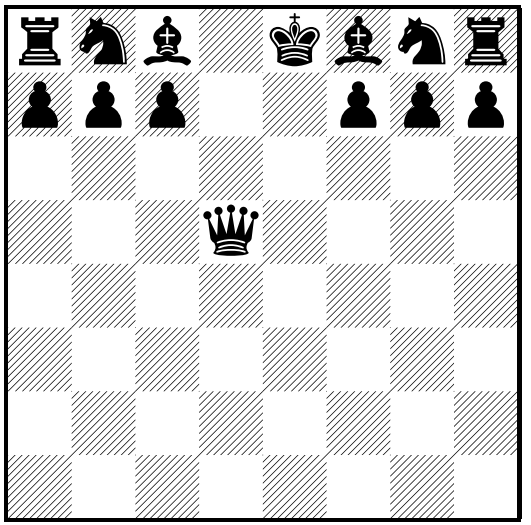
- Falls der Zug ungültig ist, erfährt man, dass man einen anderen Zug versuchen soll

Wenn eine Figur geschlagen wird, erfahren es beide Spieler

- Man erfährt den Quadranten, jedoch nicht die Art der Figur

Wenn ein legaler Zug ein Schach, ein Schachmatt oder einen Patt für den Gegner verursacht, erfahren es beide Spieler

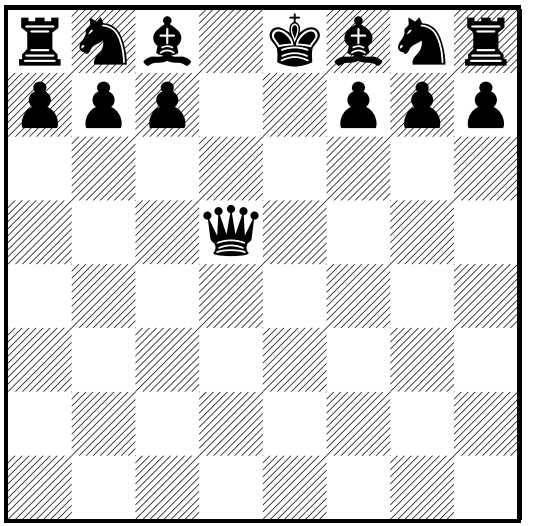
- Man erfährt, ob Schach verursacht wurde durch lange/kurze Diagonale, Reihe, Linie, Springer (oder Kombination derer)



Beispiel: Kriegspiel-Schach

Größe der Informationsmenge (Menge aller Zustände, in denen man sich *möglicherweise* befindet):

Schach:	1
Texas hold'em:	10^3
Bridge:	10^7
Kriegspiel:	10^{14}



Bei Bridge oder Poker resultiert die Unsicherheit durch das zufällige Austeilen der Karten

> Wahrscheinlichkeitsverteilung ist leicht zu berechnen

Beim Kriegspiel resultiert die Unsicherheit daher, dass die gegnerischen Züge nicht sichtbar sind

> Kein einfacher Weg, um eine geeignete Wahrscheinlichkeitsverteilung zu bestimmen

Beispiel: Giraffe - Ein Schachprogramm (1/3)

Motivation:

- Man nimmt an, dass der Spielbaum für Schach eine Größe von ca 10^{123} hat.
- Selbst bei optimaler Anordnung möglicher Züge im Spielbaum müsste immer noch ein Baum der Größe 10^{62} durchsucht werden.
- Die Bewertung der einzelnen Positionen und Züge geschieht semi-heuristisch durch hand-getunete und ausgesuchte Kriterien.
- Hunderte verschiedener Parameter fließen in die Bewertung ein; jahrelange Entwicklung.

Beispiel: Giraffe - Ein Schachprogramm (2/3)

Ansatz:

- Computer lernt Schachspielen, indem er gegen sich selber spielt.
- Mustererkennung (z.B. bestimmte Stellung) und Regeln als Ergebnis eines riesigen, rekurrenten neuronalen Netzes (Deep Learning)
- So gut wie kein Vorwissen einprogrammiert; sämtliche Kriterien werden vom Algorithmus selbst entdeckt.

Beispiel: Giraffe - Ein Schachprogramm (3/3)

Ergebnis:

- Trainingszeit: 72 Stunden auf 20 CPUs verteilt.
- Spielstärke: innerhalb der TOP 2.2% (FIDE International Master)
- Moderne Schachprogramme spielen teilweise auf Großmeisterniveau, sind allerdings über mehrere Jahre hinweg entwickelt worden.

Übersicht

1. Spiele

2. Bedingungserfüllungsprobleme

Was ist ein BEP?

BEP $= < \mathcal{X}, \mathcal{C}, \mathcal{D} >$

- $\mathcal{X}$ ist ein endliche Menge von Variablen $X_1, \ldots, X_n$.
- $\mathcal{C}$ ist eine endliche Menge von Bedingungen $C_1, \ldots, C_m$.
- $\mathcal{D}$ ist eine nicht-leere Menge $D_1, \ldots D_n$ von möglichen Werten für jede Variable X_i.

Ein *Zustand* ist definiert als *Zuweisung* von Werten zu einigen oder allen Variablen.

Eine *konsistente Zuweisung* ist eine Zuweisung, die keine der Bedingungen verletzt.

Was ist ein BEP?

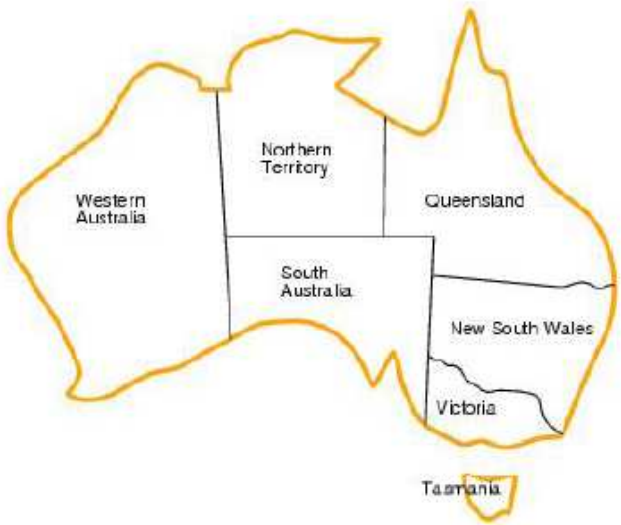
Eine Zuweisung ist *vollständig*, wenn jeder Variablen ein Wert zugewiesen wurde.

Eine Lösung für ein BEP ist eine vollständinge und konsistente Zuweisung.

In einigen Fällen muss durch die Lösung außerdem eine Zielfunktion maximiert werden.

BEP Beispiel: Graphfärbungsproblem

Variablen:
$X_1 = WA, X_2 = NT, X_3 = Q, X_4 = NSW, X_5 = V, X_6 = SA$ und $X_7 = T$



Wertebereiche:
$D_i = \{\text{rot, grün, blau}\}$
für $i = 1, \ldots, 7$.

Bedingungen:
benachbarte Regionen müssen unterschiedliche Farben haben,
z.B.: $WA \neq NT$ oder
$(WA, NT) \in \{(rot, grün), (rot, blau), (grün, rot), \ldots\}$

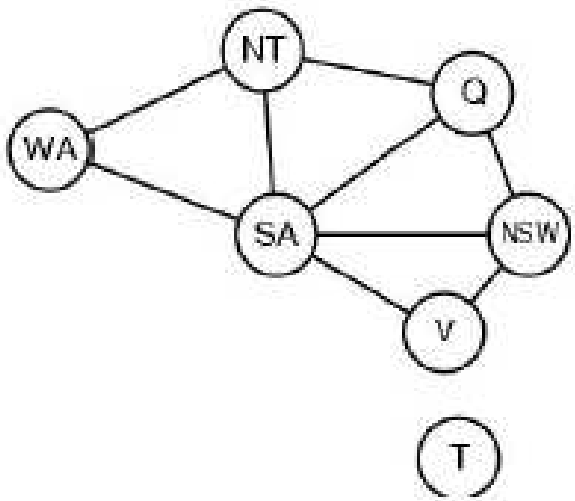
BEP Beispiel Graphfärbungsproblem

Eine Lösung für ein BEP ist eine vollständinge und konsistente Zuweisung, z.B.: $\{WA = red, NT = grün, Q = rot, NSW = grün, V = rot, SA = blau, T = grün\}$.

Bedingungsgraph

In einem binären BEP kommen in jeder Bedingung höchstens zwei Variablen vor.

Solche Probleme lassen sich durch einen Bedingungsgraphen modellieren: Knoten sind Variablen, Kanten zeigen Bedingungen.



Hier: Tasmanien ist ein unabhängiges Teilproblem.

Varianten von BEPs

Diskrete Variablen

- Endlicher Wertebereich der Größe d $\Rightarrow O(d^n)$ mögliche Zuweisungen
  z.B. Bool'sche BEPs, SAT ($\mathbb{NP}$-vollständig)
- Unendlicher Wertebereich (ganze Zahlen, Strings, etc.)
  z.B. Maschinenbelegungsprobleme, Variablen sind Start- bzw. Endzeiten für Aufträge), lineare Bedingungen lösbar, nicht-lineare unentscheidbar

Kontinuierliche Variablen

- Start- und Endzeiten von Beobachtungen des Hubbleteleskops
- Lineare Bedingungen, die in Polynomialzeit durch ganzzahlige Optimierung gelöst werden können

Varianten von Bedingungen

Einstellige Bedingunen beinhalten nur eine Variable
z.B.: $SA \neq grün$

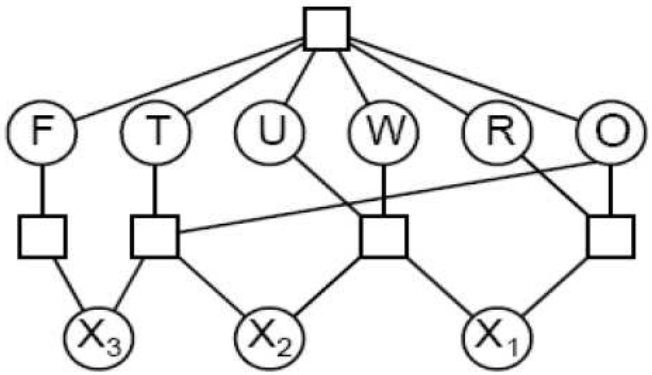
Zweistellige Bedingungen beinhalten Paare von Variablen
z.B.: $SA \neq WA$

Höherstellige Bedingungen beinhalten drei oder mehr Variablen
z.B.: kryptoarithmetische Spaltenbedingungen

Präferenzen (weiche Bedingungen), z.B. “rot ist besser als grün”, sind oft durch Kosten für Variablenzuweisungen darstellbar (Optimierung mit Nebenbedingungen)

Beispiel: Kryptoarithmetik

$$\begin{array}{cccc} & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$



hypergraph

Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Constraints
alldiff(F, T, U, W, R, O)
$O + O = R + 10 \cdot X_1$, etc.

Beispiel: Echte Probleme

Zuweisungsprobleme: Wer gibt welche Vorlesung?

Schedulingprobleme: Welche Vorlesung findet wann und wo statt?

Hardwarekonfiguration

Exceltabellen

Logistische Probleme

BEP als einfaches Suchproblem

Ein BEP kann als einfaches Suchproblem aufgefasst werden

Inkrementelle Formulierung:

- Startzustand: Die leere Zuweisung $\{\}$
- Nachfolgerfunktion: Weise einer bisher nicht belegten Variable einen Wert zu, sodass kein Konflikt entsteht
- Zieltest: Die aktuelle Zuweisung ist vollständig
- Pfadkosten: konstante Kosten für jeden Schritt

BEP als einfaches Suchproblem

Dieses Vorgehen gilt für alle BEPs!

Da die Lösung bei Tiefe n (wenn es n Variablen gibt) liegt, kann Tiefensuche genutzt werden

Pfad ist irrelevant, also kann auch komplette Zustandsrepräsentation genutzt werden

Verzweigungsgrad b des Suchbaums in der Wurzel ist $n \cdot d$

$b = (n - l) \cdot d$ in Tiefe l, also gibt es $n!d^n$ Blätter (aber nur d^n vollständige Zuweisungen)

Grund: BEPs sind kommutativ (die Reihenfolge der Zuweisungen hat keine Auswirkungen auf die Lösung

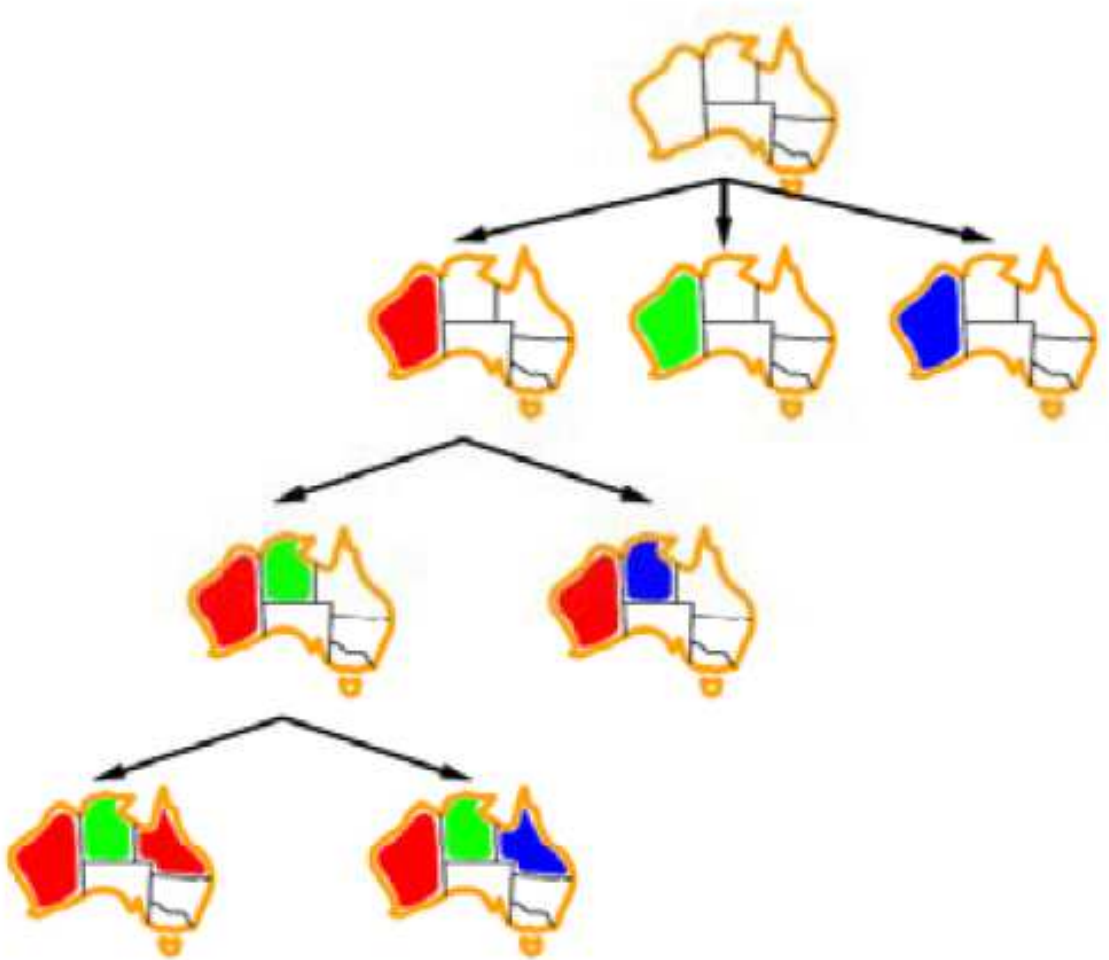
Backtracking

Tiefensuche

Wählt Werte für jeweils eine Variable pro Schritt und geht zurück wenn keine gültigen Schritte mehr möglich sind

Uninformiertes Suchverfahren (im generellen keine gute Performance)

Beispiel: Backtracking



Backtracking Effizienzverbesserung

Drei Hauptfragen:

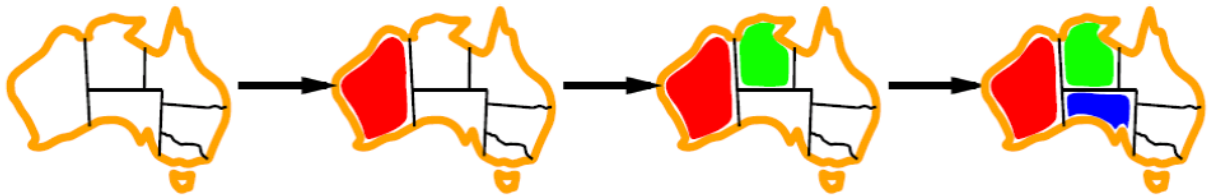
Welche Variable sollte als nächstes belegt werden und in welcher Reihenfolge soll man die Werte durchprobieren?

Welche Implikationen ergeben sich aus einer Belegung für die übrigen, noch nicht belegten Variablen?

Wenn ein Pfad *scheitert*, kann die Suche es vermeiden, diesen Fehler in zukünftigen Pfaden zu wiederholen?

Minimum Remaining Values

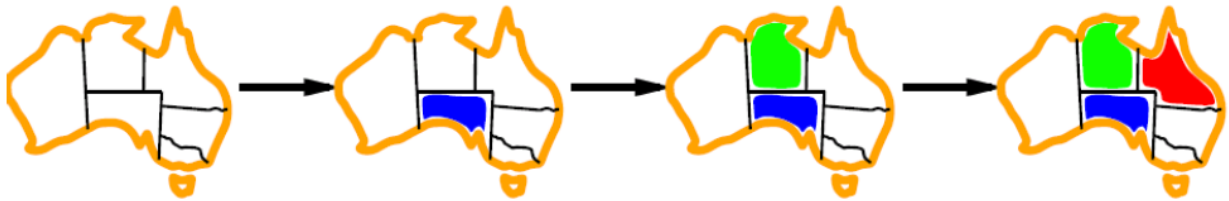
Die *minimum remaining values* Heuristik (am stärksten durch Bedingungen eingeschränkte Variable) belegt die Variable als nächste, bei der die wenigsten Werte noch übrig sind.



Die Heuristik beantwortet die Frage, welche Variable als nächstes belegt werden soll.

Most Constraining Variable

Die *most constraining variable* Heuristik belegt die Variable als nächste, die in den meisten Bedingungen für andere Variable auftaucht.



Die Heuristik beantwortet die Frage, welche Variable als nächstes belegt werden soll.

Least Constraining Value

Die *least constraining value* Heuristik belegt eine gewählte Variable mit dem Wert, der die wenigstens Bedingungen für die übrigen Variablen.

Die Heuristik beantwortet die Frage, welche Wert für eine Variable gewählt werden soll.

Beispiel: Sudoku

Sudoku ist ein BEP

Variablen: $X_{11}, X_{12}, \ldots, X_{99}$

Wertebereiche: $\{1, 2, \ldots, 9\}$

Bedingungen: alle Zeilen, alle Spalten und alle 3×3-Blöcke müssen jede Zahl genau einmal enthalten

Die MRV-Heuristik füllt die Zelle mit den wenigsten möglichen Werten

Die MCV-Heuristik füllt die Zelle, die die meisten anderen, freien Zellen beeinflusst

Die LCV-Heuristik wählt den Wert (für eine gegebene Zelle), der die meisten Möglichkeiten für die übrigen Zellen lässt.

Vorwärtsprüfung

Können wir einen unvermeidlichen Fehler frühzeitig entdecken? Und ihn dann vermeiden?

Vorwärtsprüfung: Führe eine Liste von verbleibenden, möglichen Werten für freie Variablen

Beende die Suche, wenn irgendeine Variable keine gültigen Werte mehr hat.

Funktioniert sehr gut zusammen mit der MRV-Heuristik

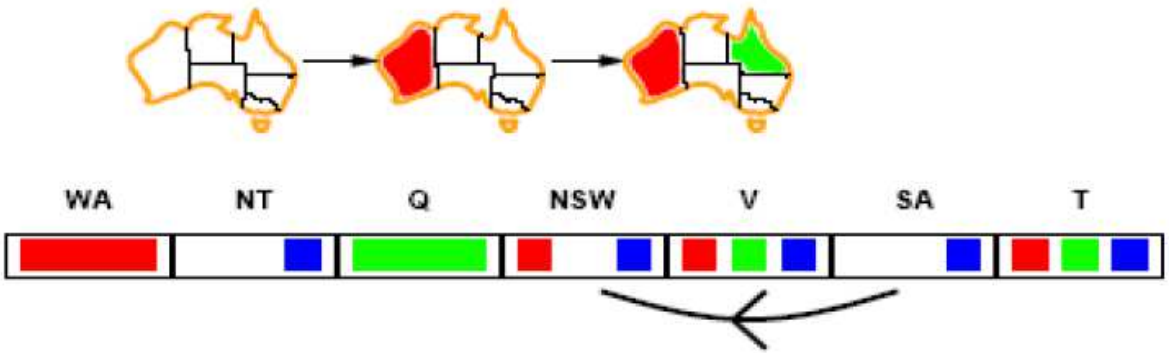
Bedingungsverbreitung

Vorwärtsprüfung findet nicht alle Fehler.

Wenn für zwei Variablen die selben, sich ausschließenden Belegungen verbleiben, wird das von der Vorwärtsprüfung nicht erkannt.

Durch wiederholtes Propagieren der Bedingungen, können diese lokal besser durchgesetzt werden.

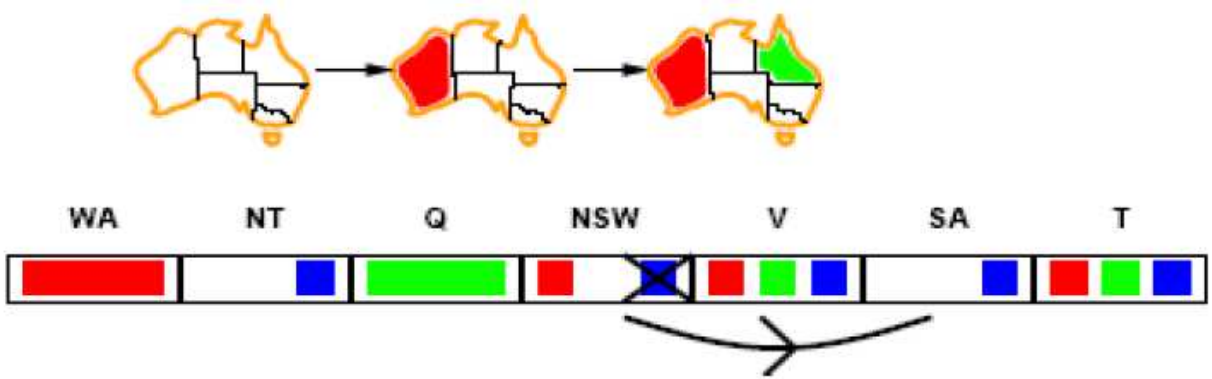
Kantenkonsistenz



Eine Kante $X \rightarrow Y$ ist genau dann konsistent, wenn für jeden Wert x von X mindestens ein Wert y für Y erlaubt ist.

$SA \rightarrow NSW$ ist konsistent $\leftrightarrow SA = \text{blau} \wedge NSW = \text{rot}$

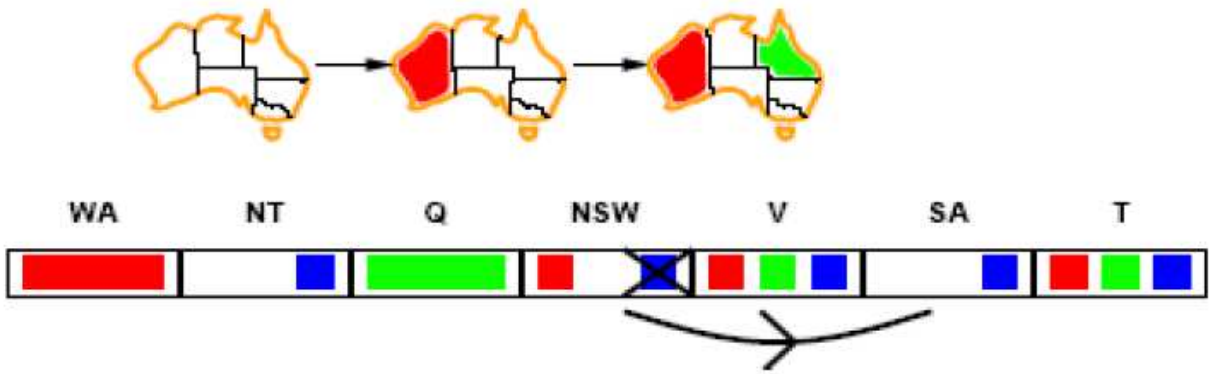
Kantenkonsistenz



Eine Kante $X \rightarrow Y$ ist genau dann konsistent, wenn für jeden Wert x von X mindestens ein Wert y für Y erlaubt ist.

Umgekehrt: $NSW \rightarrow SA$ ist konsistent
$\leftrightarrow NSW = \text{rot} \wedge SA = \text{blau}$ und $NSW = \text{blau} \wedge SA = ???$.

Kantenkonsistenz



Kante kann konsistent gemacht werden, indem *blau* von den möglichen Werten für *NSW* entfernt wird.

Dadurch können weitere Inkonsistenzen entstehen und der Prozess muss für die übrigen Kanten wiederholt werden.

Diese Heuristik erkennt Fehler früher und kann daher nachg jeder Zuweisung als eine Art *Präprozessor* verwendet werden.

k-Konsistenz

Kantenkonsistenz erkennt nicht alle Inkonsistenzen

Stärkere Formen der Verbreitung können durch k-Konsistenzen definiert werden

Ein BEP ist k-konsistent, wenn für jede Menge von $k-1$ Variablen und jede konsistente Zuweisung zu diesen Variablen, einer beliebigen k-ten Variable ein gültiger Wert zugewiesen werden kann.

Formen: 1-Konsistenz (Knotenkonsistenz), 2-Konsistenz (Kantenkonsistenz), 3-Konsistenz (Pfadkonsistenz)

k-Konsistenz

Ein Graph ist *stark k-konsistent*, wenn er für jedes $j \leq k$ j-konsistent ist.

Das ist ideal, denn eine Lösung kann dann in $\mathcal{O}(nd)$ gefunden werden.

Aber: No-Free-Lunch! Jeder Algorithmus der n-Konsistenz herstellt, braucht im schlimmsten Fall mindestens $\mathcal{O}(2^n)$.