Kunstmatige Intelligentie (AI)

Hoofdstuk 5 van Russell/Norvig = [RN]

Spel(l)en

voorjaar 2024

College 6 en 7, 13 en 20 maart 2024

www.liacs.leidenuniv.nl/~kosterswa/AI/spellen.pdf

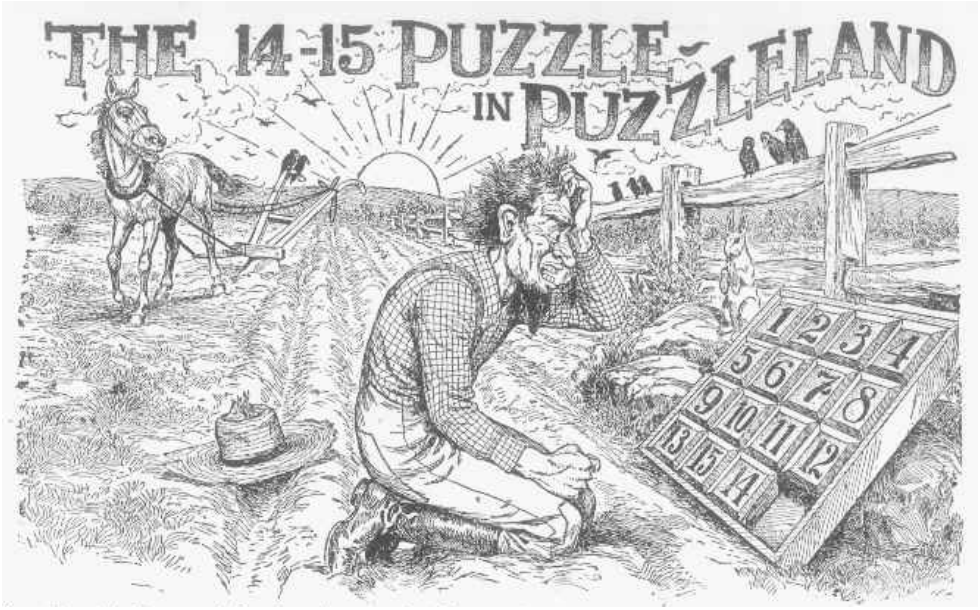
Spellen geven aanleiding tot zeer complexe zoekproblemen, bijvoorbeeld bij schaken, Go, vier-op-een-rij. Extra problemen zijn contingency (onzekerheid), kansen, . . .
We hadden evaluatie-functies nodig om niet-eindtoestanden te beoordelen, bijvoorbeeld bij schaken: pion 1, paard/loper 3, toren 5, koningin 9, veiligheid koning, . . .

We willen prunen: niet alles doorrekenen. We bekijken met name het minimax-algoritme van Von Neumann (1928) en het α-β-algoritme uit 1956/58.

Nu: deep learning, . . .

Leuk om te lezen: H.J. van den Herik, J.W.H.M. Uiterwijk en J. van Rijswijck, Games solved: Now and in the future, Artificial Intelligence 134 (2002) 277–311.

De derde programmeeropgave gaat over schuifpuzzels als de 15-puzzel die we met A* en IDA* in C++ willen programmeren.

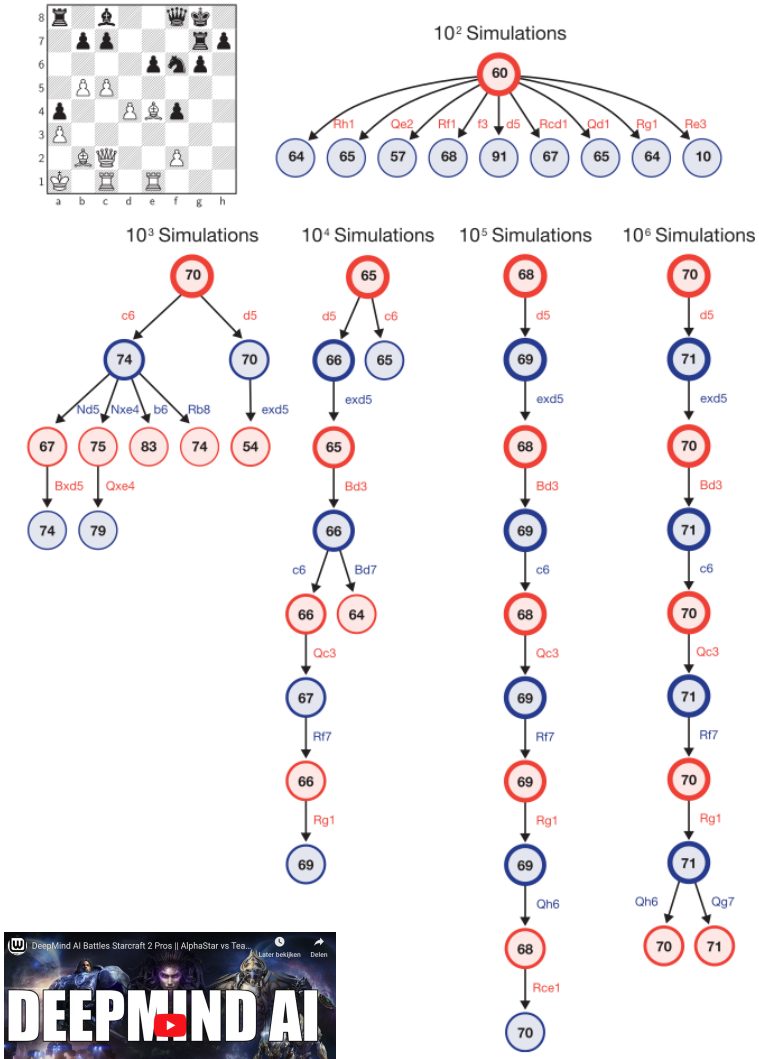


www.liacs.leidenuniv.nl/~kosterswa/aster2024.html

Vergeet de sommen niet, zoals opgave 8e over A* en IDA*.

Deep Blue vs. Garry Kasparov, 1997

Marion Tinsley (1927–1995) was de beste menselijke checkers-speler (dammen op een schaakbord) ooit.
In 2007 werd door Jonathan Schaeffer bewezen dat de beginspeler altijd minstens remise kan halen.



December 2018

AlphaZero

Silver et al.

Science 362, 1140–1144

RESEARCH

COMPUTER SCIENCE

A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play

David Silver[1,2*†], Thomas Hubert[1*], Julian Schrittwieser[1*], Ioannis Antonoglou[1], Matthew Lai[1], Arthur Guez[1], Marc Lanctot[1], Laurent Sifre[1], Dharshan Kumaran[1], Thore Graepel[1], Timothy Lillicrap[1], Karen Simonyan[1], Demis Hassabis[1†]

The game of chess is the longest-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been refined by human experts over several decades. By contrast, the AlphaGo Zero program recently achieved superhuman performance in the game of Go by reinforcement learning from self-play. In this paper, we generalize this approach into a single AlphaZero algorithm that can achieve superhuman performance in many challenging games. Starting from random play and given no domain knowledge except the game rules, AlphaZero convincingly defeated a world champion program in the games of chess and shogi (Japanese chess), as well as Go.

The study of computer chess is as old as computer science itself. Charles Babbage, Alan Turing, Claude Shannon, and John von Neumann devised hardware, algorithms, and theory to analyze and play the game of chess. Chess subsequently became a grand challenge task for a generation of artificial intelligence researchers, culminating in high-performance computer chess programs that play at a superhuman level (1, 2). However, these systems are highly tuned to their domain and cannot be generalized to other games without substantial human effort, whereas general game-playing systems (3, 4) remain comparatively weak. A long-standing ambition of artificial intelligence has been to create programs that can instead learn for themselves from first principles (5, 6). Recently, the AlphaGo Zero algorithm achieved superhuman performance in the game of Go by representing Go knowledge with the use of deep convolutional neural networks (7, 8), trained solely by reinforcement learning from games of self-play (9). In this paper, we introduce AlphaZero, a more generic version of the AlphaGo Zero algorithm that accommodates, without special casing, a broader class of game rules. We apply AlphaZero to the games of chess and shogi, as well as Go, by using the same algorithm and network architecture for all three games. Our results demonstrate that a general-purpose reinforcement learning algorithm can learn, tabula rasa—without domain-specific human knowledge or data, as evidenced by the same algorithm succeeding in multiple domains—superhuman performance across multiple challenging games.

A landmark for artificial intelligence was achieved in 1997 when Deep Blue defeated the human world chess champion (1). Computer chess programs continued to progress steadily beyond human level in the following two decades. These programs evaluate positions by using handcrafted features and carefully tuned weights, constructed by strong human players and programmers, combined with a high-performance alpha-beta search that expands a vast search tree by using a large number of clever heuristics and domain-specific adaptations. In (10) we describe these augmentations, focusing on the 2016 Top Chess Engine Championship (TCEC) season 9 world champion Stockfish (11); other strong chess programs, including Deep Blue, use very similar architectures (1, 12).

In terms of game tree complexity, shogi is a substantially harder game than chess (13, 14): It is played on a larger board with a wider variety of pieces; any captured opponent piece switches sides and may subsequently be dropped anywhere on the board. The strongest shogi programs, such as the 2017 Computer Shogi Association (CSA) world champion Elmo, have only recently defeated human champions (15). These programs use an algorithm similar to those used by computer chess programs, again based on a highly optimized alpha-beta search engine with many domain-specific adaptations.

AlphaZero replaces the handcrafted knowledge and domain-specific augmentations used in traditional game-playing programs with deep neural networks, a general-purpose reinforcement learning algorithm, and a general-purpose tree search algorithm.

Instead of a handcrafted evaluation function and move-ordering heuristics, AlphaZero uses a deep neural network $(\mathbf{p}, v) = f_\theta(s)$ with parameters θ. This neural network $f_\theta(s)$ takes the board position s as an input and outputs a vector of move probabilities $\mathbf{p}$ with components $p_a = \Pr(a|s)$ for each action a and a scalar value v estimating the expected outcome z of the game from position s, $v \approx \mathbb{E}[z|s]$. AlphaZero learns these move probabilities and value estimates entirely from self-play; these are then used to guide its search in future games.

Instead of an alpha-beta search with domain-specific enhancements, AlphaZero uses a general-purpose Monte Carlo tree search (MCTS) algorithm. Each search consists of a series of simulated games of self-play that traverse a tree from root state s_{root} until a leaf state is reached. Each simulation proceeds by selecting in each state s a move a with low visit count (not previously frequently explored), high move probability, and high value (averaged over the leaf states of

*DeepMind, 6 Pancras Square, London N1C 4AG, UK. †University College London, Gower Street, London WC1E 6BT, UK.
*These authors contributed equally to this work.
†Corresponding author. Email: dsilver@google.com (D.S.); dhcontact@google.com (D.H.)

Fig. 1. Training AlphaZero for 700,000 steps. Elo ratings were computed from games between different players where each player was given 1 s per move. (**A**) Performance of AlphaZero in chess compared with the 2016 TCEC world champion program Stockfish. (**B**) Performance of AlphaZero in shogi compared with the 2017 CSA world champion program Elmo. (**C**) Performance of AlphaZero in Go compared with AlphaGo Lee and AlphaGo Zero (20 blocks over 3 days).

DeepNash

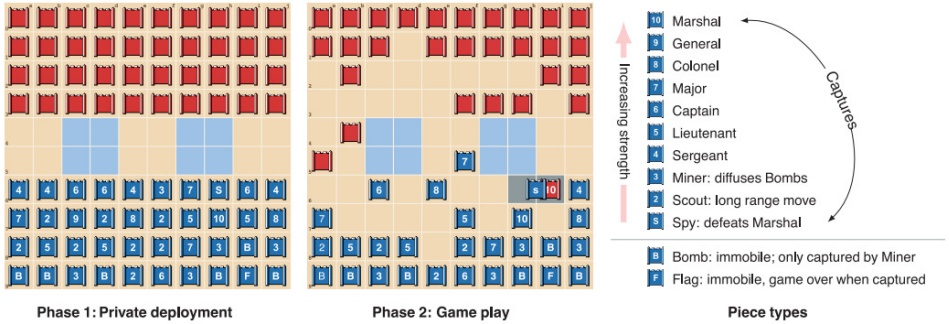
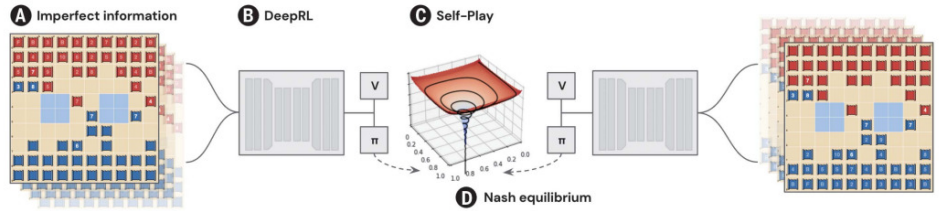


Fig. 1. Stratego is a two-player board game in which players try to capture the opponent's flag. Initially, the players secretly deploy 40 pieces of diverse strengths on the board. Then, they take turns moving pieces, possibly encountering an opponent piece that reveals both piece identities, and then the weaker piece is removed. Two lakes (indicated in blue) cannot be crossed by any piece. The complete rules are defined by the International Stratego Federation.



Replicator dynamics: $\frac{d}{d\tau}\pi^i_\tau(a^i) = \pi^i_\tau(a^i)\left[Q^i_{\pi_\tau}(a^i) - \sum_{b^i}\pi^i_\tau(b^i)Q^i_{\pi_\tau}(b^i)\right]$

Reward transformation: $r^i(\pi^i, \pi^{-i}, a^i, a^{-i}) = r^i(a^i, a^{-i}) - \eta\log\left(\frac{\pi^i(a^i)}{\pi^i_{\text{reg}}(a^i)}\right) + \eta\log\left(\frac{\pi^{-i}(a^{-i})}{\pi^{-i}_{\text{reg}}(a^{-i})}\right)$

breaking news

↓ link

RESEARCH

MACHINE LEARNING

Mastering the game of Stratego with model-free multiagent reinforcement learning

Julien Perolat*†, Bart De Vylder*†, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer‡, Paul Muller, Jerome T. Connor, Neil Burch, Thomas Anthony, Stephen McAleer, Romuald Elie, Sarah H. Cen, Zhe Wang, Audrunas Gruslys, Aleksandra Malysheva, Mina Khan, Sherjil Ozair, Finbarr Timbers, Toby Pohlen, Tom Eccles, Mark Rowland, Marc Lanctot, Jean-Baptiste Lespiau, Bilal Piot, Shayegan Omidshafiei, Edward Lockhart, Laurent Sifre, Nathalie Beauguerlange, Remi Munos, David Silver, Satinder Singh, Demis Hassabis, Karl Tuyls*†

We introduce DeepNash, an autonomous agent that plays the imperfect information game Stratego at a human expert level. Stratego is one of the few iconic board games that artificial intelligence (AI) has not yet mastered. It is a game characterized by a twin challenge: It requires long-term strategic thinking as in chess, but it also requires dealing with imperfect information as in poker. The technique underpinning DeepNash uses a game-theoretic, model-free deep reinforcement learning method, without search, that learns to master Stratego through self-play from scratch. DeepNash beat existing state-of-the-art AI methods in Stratego and achieved a year-to-date (2022) and all-time top-three ranking on the Gravon games platform, competing with human expert players.

Perolat et al., Science 378 (2022) 990–996: Mastering the game of Stratego . . .

Deep neural nets, reinforcement learning, selfplay.

Kunnen computers denken? Diplomacy!

Spellen kunnen als volgt worden ingedeeld:

	deterministisch	kans
perfecte informatie	schaken, dammen, checkers, Go, othello	monopoly, backgammon
onvolledige informatie	zeeslag, mastermind	bridge, poker, scrabble

En dan is er nog onderscheid in het aantal spelers.
Met name over schaken is veel gepubliceerd, waaronder allerlei anecdotes.
De ultieme uitdaging is/was het spel Go. En Diplomacy?

Er zijn verschillende strategieën om (een benadering van) de "speltheoretische waarde" van een spel (met perfecte informatie = volledig observeerbaar; nulsom = zero-sum: goed voor de één is even slecht voor de ander) te bepalen.

Shannon (1950) onderscheidt drie types:

type A reken alles tot en met zekere diepte door, en gebruik daar een evaluatie-functie

type B reken soms verder door (als het onrustig is: "quiescence"); gebruik heuristische functie om dit te sturen

type C doelgericht menselijk zoeken

A en B zijn $\approx$ "brute-force", C is meer "knowledge-based".

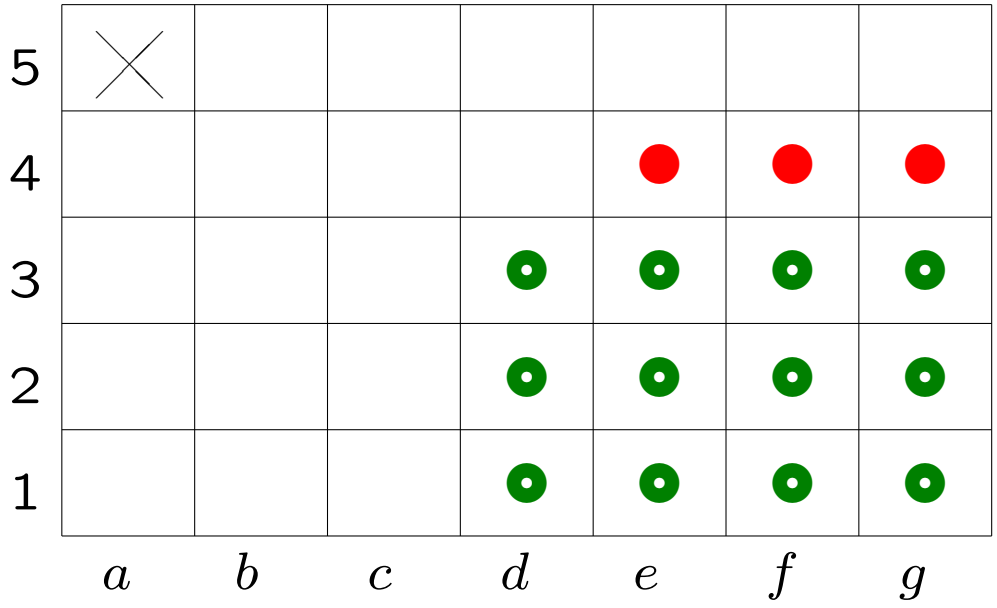
Er worden soms drie soorten oplossingen van spellen onderscheiden:

ultra-zwak de speltheoretische waarde van de beginstand is bekend: "je kunt vier-op-een-rij winnen"

zwak idem, en een optimale strategie is bekend (begin in middelste kolom, . . . , zie later)

sterk in elke legale positie is een optimale strategie bekend

Het spel Chomp wordt gespeeld met een rechthoekige reep chocola, waar de spelers om de beurt een stuk rechtsonder afhappen (een blokje en alles hier onder/rechts van). Wie het (vergiftigde) blokje linksboven eet, heeft verloren.



eerste hap: $d3$

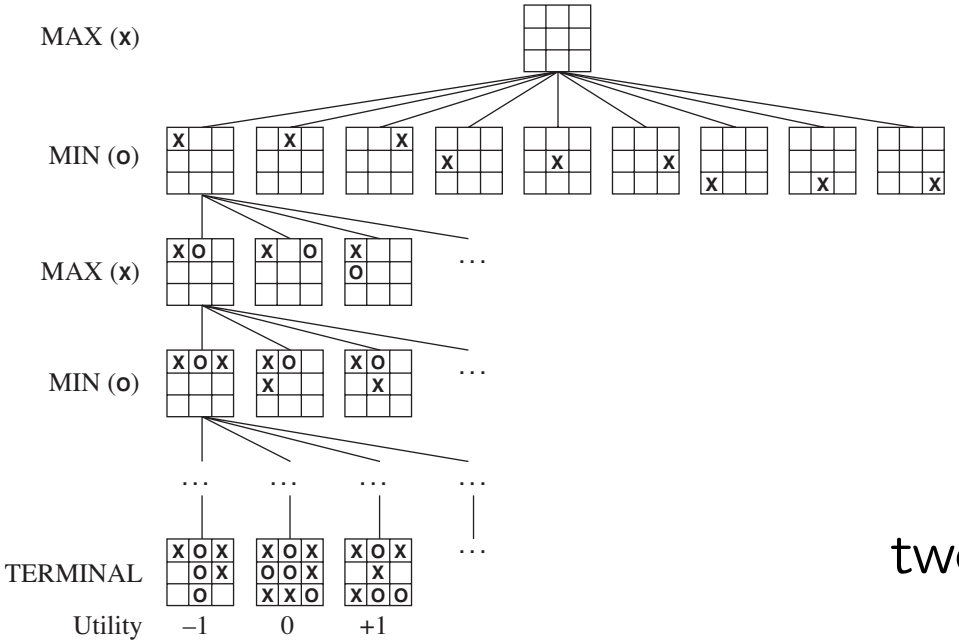
tweede hap: $e4$

Bewering: de beginnende speler bij Chomp kan altijd winnen! Immers, als je door het blokje rechtsonder te nemen kunt winnen is het goed. Als dat niet zo is, heeft de tegenstander blijkbaar een voor hem winnende "tegen-hap". Die kun je dan zelf als eerste doen, en dus daarmee winnen! Dit argument heet strategy stealing.

Kortom: het spel Chomp is ultra-zwak opgelost, de echte winnende zet weten we niet . . .

Voor bijvoorbeeld 2×2 en 2×3 Chomp "wint" het blokje rechtsonder (algemener voor vierkanten: neem het vakje rechts onder het vergiftigde, en dan "spiegelen"); bij 3×4 Chomp het blokje uit de middelste rij, derde kolom.

Voorbeeld spelboom



Boter, kaas en eieren

twee spelers, om en om:

MAX (X) en MIN (O); X begint

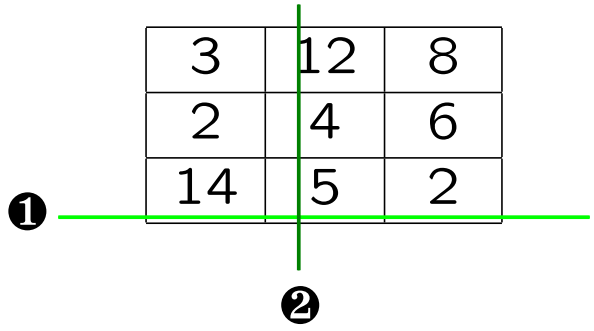
5478 toestanden

Boter, kaas en eieren is een voorbeeld van een tweepersoons deterministisch nulsom-spel met volledige informatie, waarbij de spelers om de beurt een "legale zet" doen. MAX moet een strategie vinden die tot een winnende eindtoestand leidt, ongeacht wat MIN doet. De strategie moet elke mogelijke zet van MIN correct beantwoorden.

Een utility-functie (= payoff-functie) geeft de waarde van eindtoestanden. Hier is dat: $-1/0/1$; bij backgammon is dat: $-192 \ldots +192$.

Uit symmetrie-overwegingen kunnen veel toestanden = posities worden wegbezuinigd. En een "actie" heet nu een "zet".

Maxi en Mini spelen het volgende eenvoudige spel: Maxi wijst eerst een (horizontale) rij aan, en daarna kiest Mini een (verticale) kolom:



Bijvoorbeeld: Maxi ❶ kiest rij 3, daarna kiest Mini ❷ kolom 2; dat levert einduitslag 5.

Maxi wil graag een zo groot mogelijk getal, Mini juist een zo klein mogelijk getal.

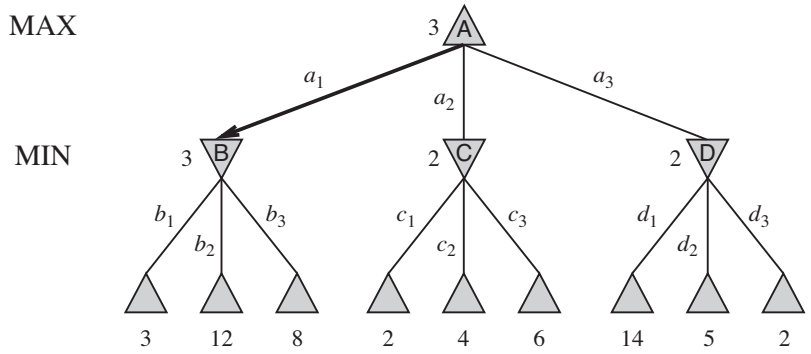
Hoe spelen we dit spel zo goed mogelijk?

Als Maxi rij 1 kiest, kiest Mini kolom 1 (levert 3); als Maxi rij 2 kiest, kiest Mini kolom 1 (levert 2); als Maxi rij 3 kiest, kiest Mini kolom 3 (levert 2). Dus kiest Maxi rij 1!

3	12	8
2	**?**	**?**
14	5	2

Nu merken we op dat de analyse (het minimax-algoritme) hetzelfde verloopt als we niet eens weten wat onder de twee vraagtekens zit. Het α-β-algoritme onthoudt als het ware de beste en slechtste mogelijkheden, en kijkt niet verder als dat toch nergens meer toe kan leiden (zie verderop).

In boomvorm:



Het minimax-algoritme is "recursief": neem in bladeren de evaluatie-functie, in MAX-knopen het maximum van de kinderen, in MIN-knopen het minimum van de kinderen. MAX- en MIN-knopen wisselen elkaar af.

Bovenstaande boom is één zet (= move) diep, oftewel twee ply.

function $MaxWaarde(toestand)$
    if $eindtoestand$ **then return** $Utility(toestand)$
    $waarde \leftarrow -\infty$
    for s **in** $Opvolgers(toestand)$ **do**
        $waarde \leftarrow \max(waarde, MinWaarde(s))$
    return $waarde$

function $MinWaarde(toestand)$
    if $eindtoestand$ **then return** $Utility(toestand)$
    $waarde \leftarrow +\infty$
    for s **in** $Opvolgers(toestand)$ **do**
        $waarde \leftarrow \min(waarde, MaxWaarde(s))$
    return $waarde$

Eigenschappen van Minimax

Compleet: als de boom eindig is (bij schaken dankzij speciale regels)

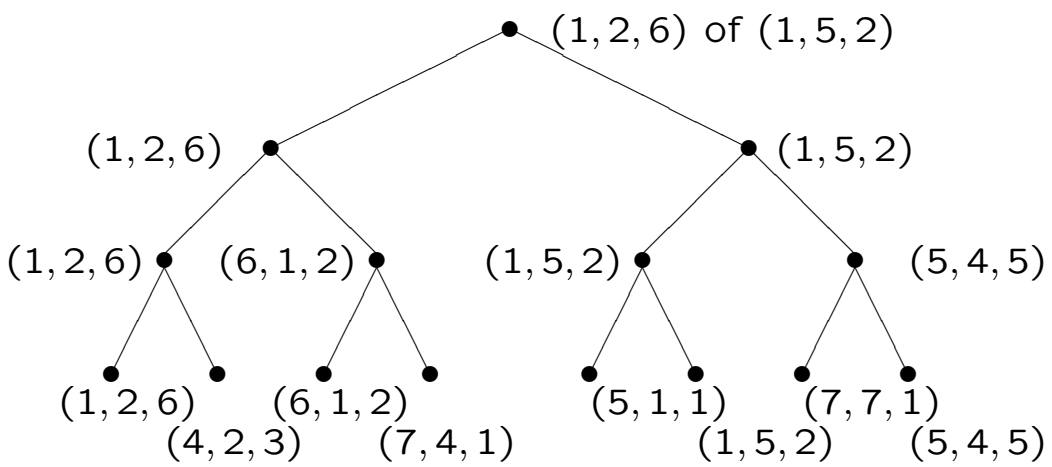
Optimaal: tegen een "optimale" tegenstander

Tijdscomplexiteit: $O(b^m)$ (b is vertakkingsgraad, m diepte van de boom)

Ruimtecomplexiteit: $O(bm)$ (bij depth-first exploratie)

Bij schaken: $b \approx 35$, $m \approx 100$; een exacte oplossing is dus heeeeeeeeeel ver weg.

Met drie spelers (A, B en C) heb je per knoop drie evaluatie-waardes (A wil de eerste zo hoog mogelijk, etcetera):



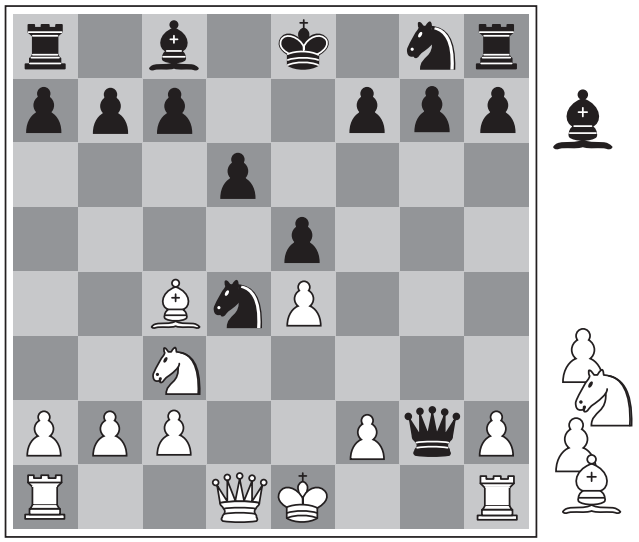
A aan zet

B aan zet

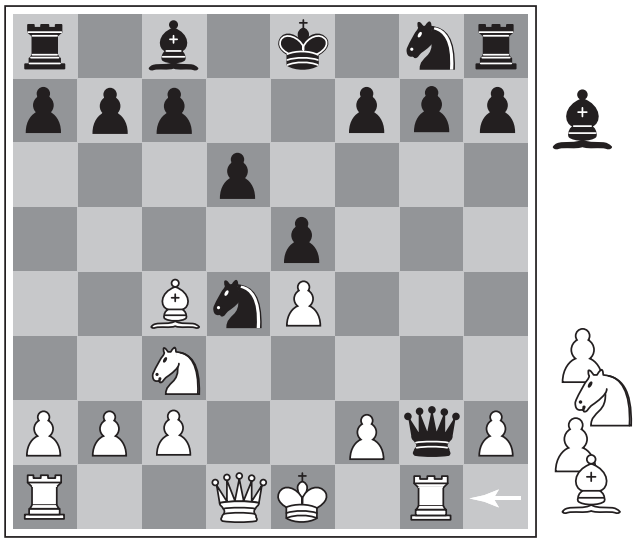
C aan zet

Voor twee spelers is één getal voldoende (bij een nulsomspel is de winst van de een het verlies van de ander).

En hoe zit het bij Diplomacy? En poker?



(a) White to move



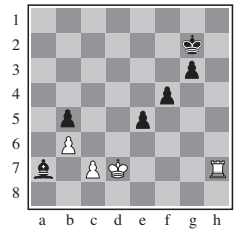
(b) White to move

Voor schaken is de evaluatie-functie meestal een gewogen som van allerlei kenmerken (“features”): 9 maal (aantal witte dames − aantal zwarte dames) + . . .

De functie moet eenvoudig te berekenen zijn, de kans op winnen aangeven, en kloppen met de utility-functie op eindtoestanden.

Bij cut-off search vervang je in het minimax-algoritme de test op eindtoestanden door een "cut-off test" — en een evaluatie-functie aldaar.

Bij schaken, met $b = 35$ en $b^m \approx 10^6$, krijg je $m = 4$. En 4-ply vooruit kijken is hopeloos ($\approx$ amateur). Men denkt: 8-ply $\approx$ PC of schaakmeester, en 12-ply is wereldtop-nivo (computer of mens).



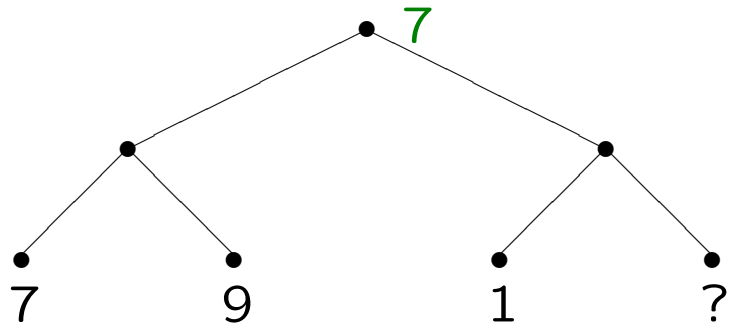
BTM

Je hebt te maken met:

– horizon-effect soms wordt een noodzakelijke slechte zet uit beeld geduwd door extra (minder slechte) zetten;

– quiescence in een onrustige periode moet je langer doorrekenen.

Het basisidee van het α-β-algoritme is het volgende: om de minimax-waarde in de wortel van onderstaande boom te berekenen is de waarde rechts onderin niet van belang — en die subboom kun je prunen (snoeien). Immers, het linker kind van de wortel is 7, en het rechterkind moet dus hoger dan 7 zijn wil het nog invloed uitoefenen. Nu is het (dankzij de 1) hoogstens 1, en zijn we klaar: waarde 7.

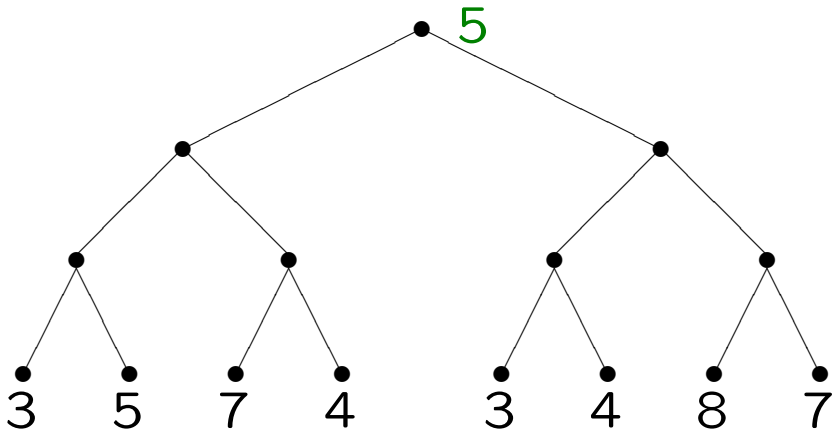


MAX aan zet

MIN aan zet

function $MaxWaarde(toestand, \alpha, \beta)$
  if $toestand$ is eindtoestand **then return** $Utility(toestand)$
    (of cut-off test, en gebruik evaluatie-functie)
  for s **in** $Opvolgers(toestand)$ **do**
    $\alpha \leftarrow \max(\alpha, MinWaarde(s, \alpha, \beta))$
    if $\alpha \geq \beta$ **then return** β
  return α

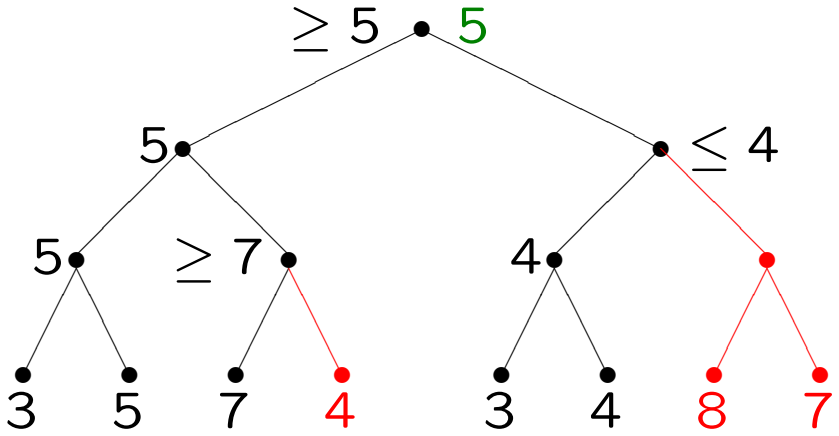
Analoog de functie $MinWaarde$, zie boek (voor een iets andere formulering). Normaal geldt bij aanroep $\alpha < \beta$; α geeft de beste (hoogste) waarde die MAX op het huidige pad kon bereiken, en β voor MIN. De variabelen α en β zijn locaal. Buitenste aanroep: $MaxWaarde(huidigetoestand, -\infty, +\infty)$.



MAX aan zet

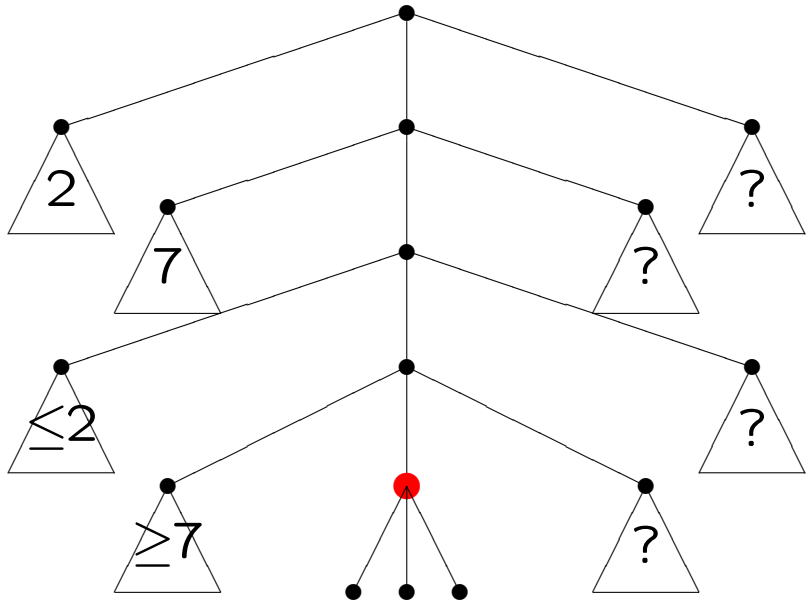
MIN aan zet

MAX aan zet



rood wordt

gepruned



MAX aan zet

MIN aan zet

MAX aan zet

MIN aan zet

MAX aan zet

Bij MAX-knoop ● geldt: $\alpha = 2$ en $\beta = 7$. Op het pad naar die knoop kan MAX al 2 afdwingen, en MIN 7. Als een kind van ● waarde 8 heeft, hoeven de volgende kinderen niet meer bekeken te worden, en krijgt ● waarde 7 (de β).

Er geldt: α-β-pruning levert exact hetzelfde eindresultaat in de wortel als "gewoon" minimax.

De effectiviteit hangt sterk af van de volgorde waarin de kinderen (zetten) bekeken worden.

Met "perfecte ordening" bereik je tijdscomplexiteit $O(b^{m/2})$ (m is de diepte van de boom), dus je kunt effectief de zoekdiepte verdubbelen.

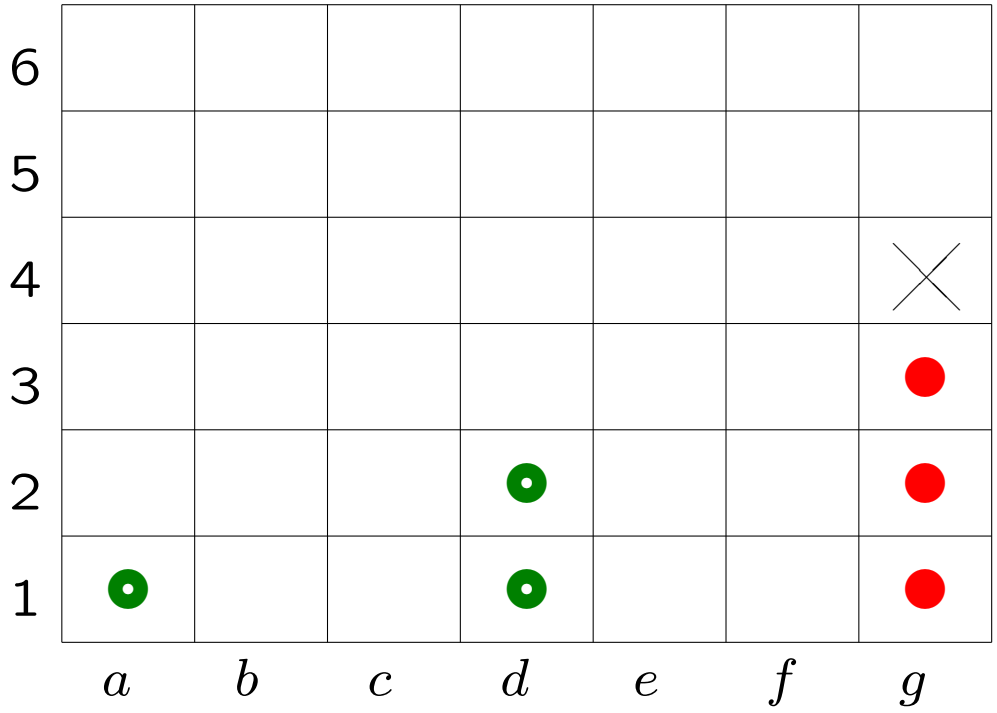
Het is dus van belang heuristieken te hebben om je zetten te ordenen. Enkele voorbeelden:

null-move bekijk eerst voor de tegenstander goede zetten (sla je eigen zet in gedachten even over)

killer als een zet ergens een snoeiing teweeg brengt, doet hij dat elders wellicht ook

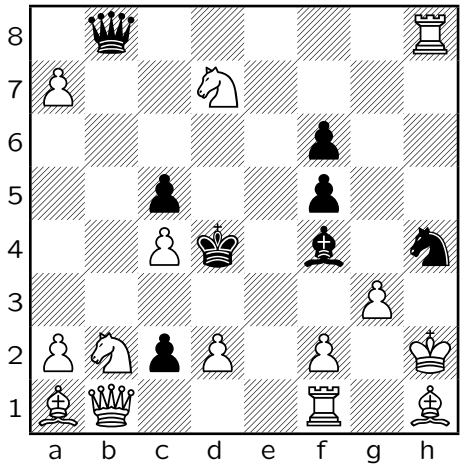
conspiracy-number $\approx$ aantal kinderen dat van waarde moet veranderen (samenzweren) om ouder van waarde te laten veranderen (voor stijgen MAX-knoop is maar één kind nodig, voor stijgen MIN-knoop zijn alle kinderen nodig)

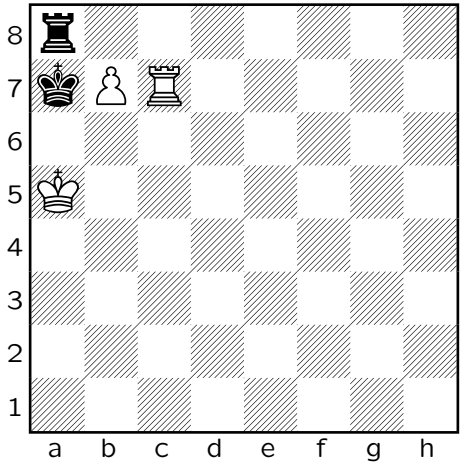
tabu-search onthoud aantal (zeer) slechte zetten

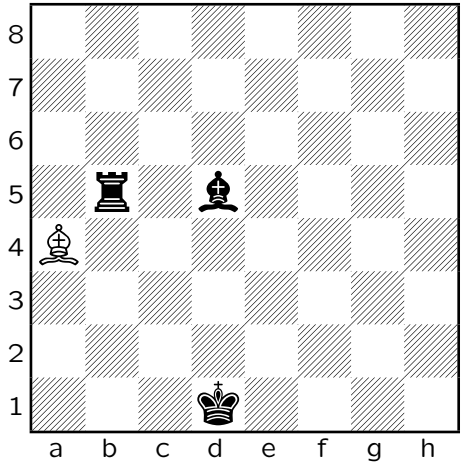


Vier-op-een-rij

Met groen aan de beurt, is $g4$ zowel voor de null-move- als de killer-heuristiek de aangewezen zet.
De voor groen winnende (begin)serie: $d1! - d2 - d3! - d4 - d5! - b1 - b2$ (een ! betekent: unieke winnende zet).





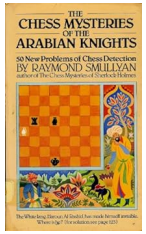


Babson task (R×h4)

Wit: mat in 1

Waar staat witte koning?

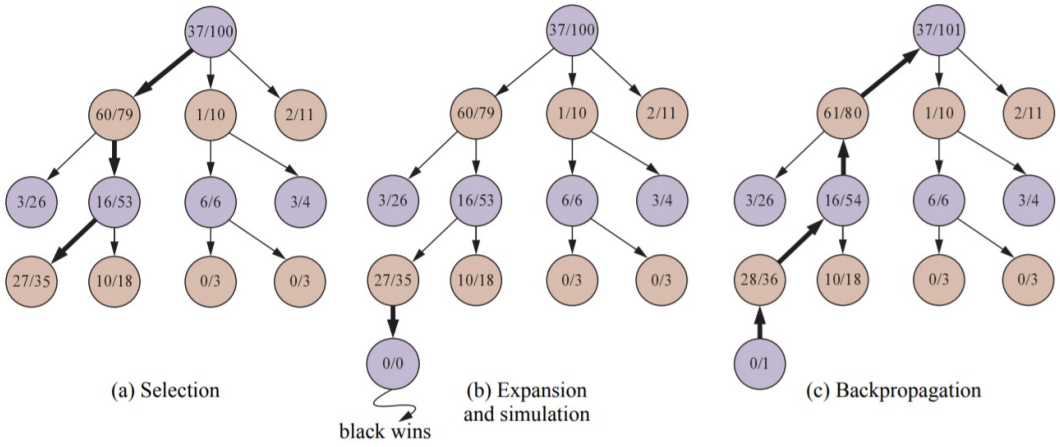
Leonid Yarosh,
Tim Krabbé



Raymond Smullyan
retrograde analyse

Bij pure Monte Carlo search bekijken we voor ieder mogelijke zet een playouts aantal random vervolgpartijen, en kiezen de (= een) zet met de hoogste (gemiddelde) uitkomst. Zie de eerste programmeeropgave.

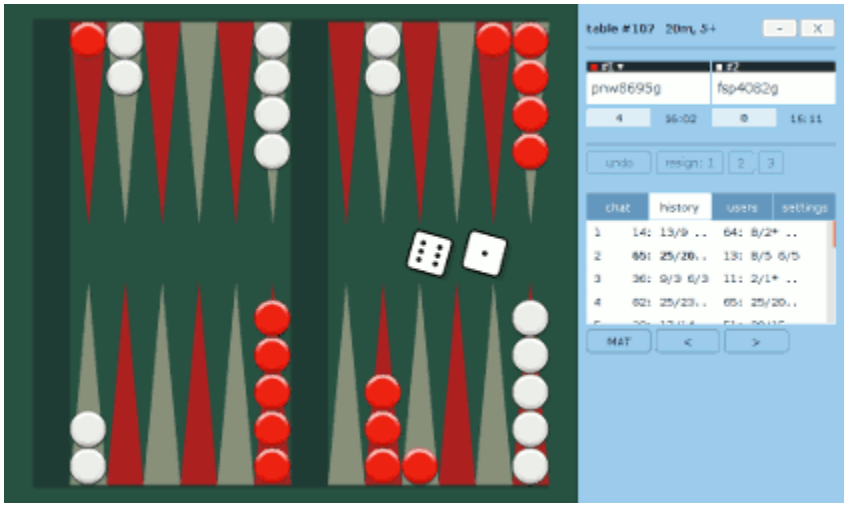
Bij Monte Carlo Tree Search (MCTS) kies je herhaald een kind (a) tot je een nieuwe knoop moet maken (b1), speel dan random uit (b2), en propageer de waardes terug naar de wortel (c). Je kiest hierbij steeds een kind met UCT-selectie (*): "upper confidence bounds applied to trees".



exploitatie exploratie

(*) $UCB1(n) = U(n)/N(n) + C \cdot \sqrt{\log(N(\text{parent}(n)))/N(n)}$ is de kans bij knoop n. Je speelt echt het meest bezochte kind n (hoogste $N(n)$); $U(n)$ is diens utility. Vaak $C = \sqrt{2}$.

Backgammon

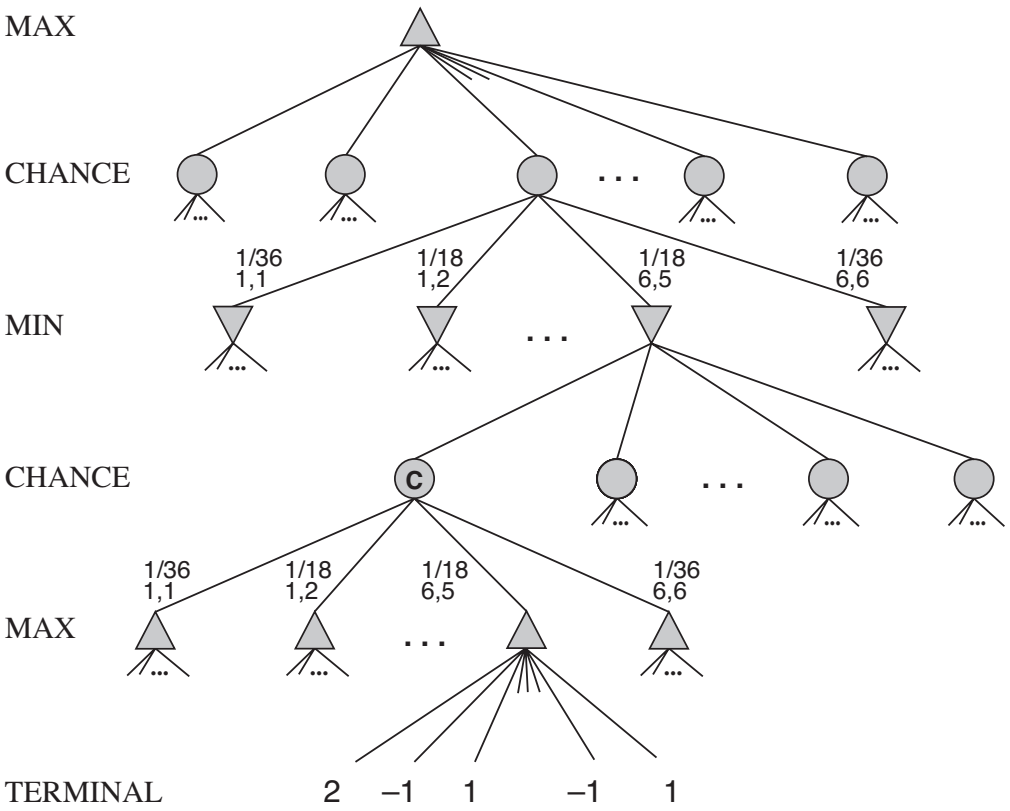


niet-deterministisch

gooi 2 dobbelstenen

en kies toegestane zet

Bij backgammon krijgen we een spelboom als:



MAX kiest

dobbel

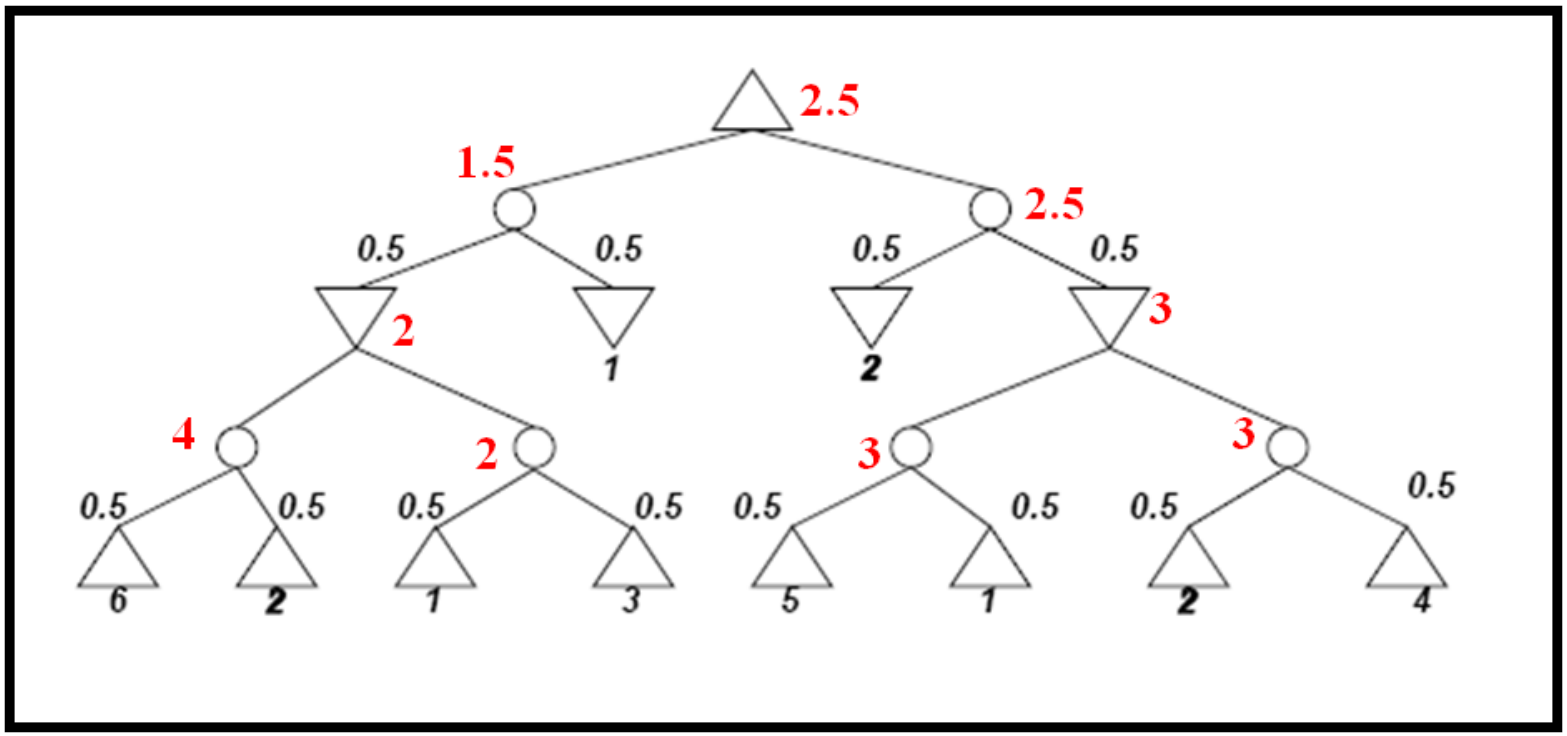
MIN kiest

dobbel

MAX kiest

eindtoestand

Een eenvoudig voorbeeld van een spel met een eerlijke munt:



Expecti-minimax — algemeen

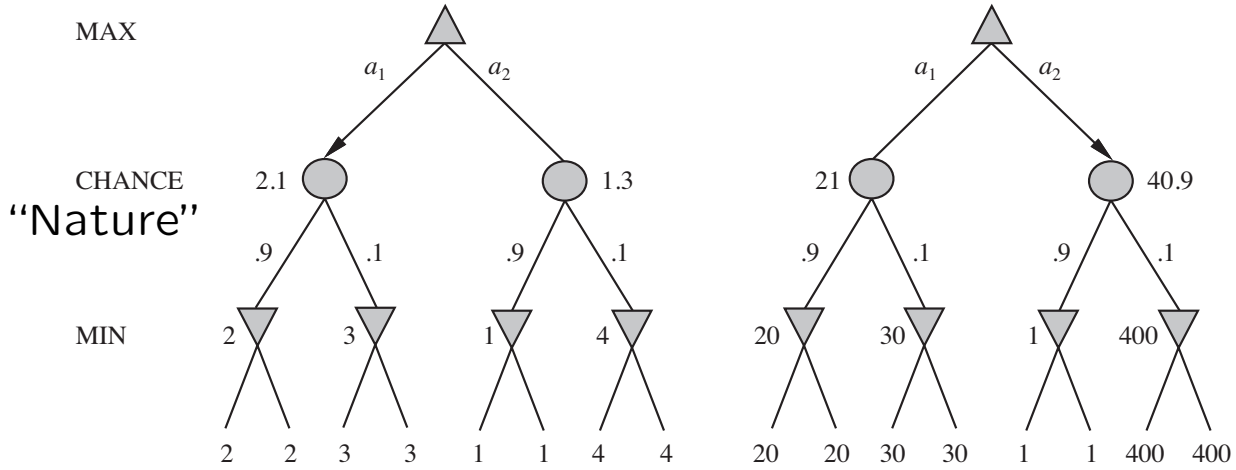
In het algemeen krijg je in een kansknoop n voor de expecti-minimax-waarde $ExpectiMinimax(n)$ een formule als:

$$\sum_{s \in Opvolgers(n)} P(s) \cdot ExpectiMinimax(s),$$

waarbij $P(s)$ de kans op s is.

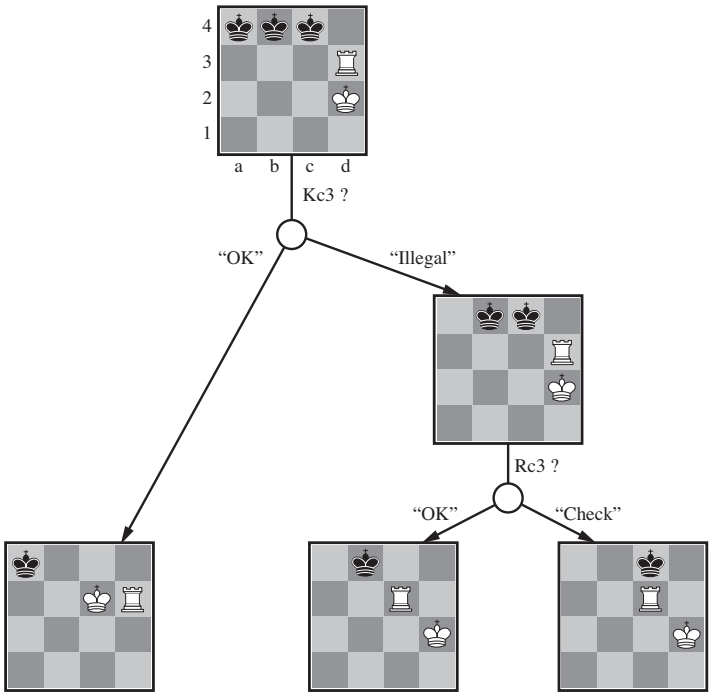
Soms is het gemiddelde "beter" dan de "exacte" minimax-waarde. Stel je voor dat je (= MAX) moet kiezen uit een MIN-knoop met kinderen 99, 1000, 1000 en 1000, en een MIN-knoop met kinderen 100, 101, 102 en 103 ...

In geval van kansknopen moet je beter op de evaluatie-functie letten!



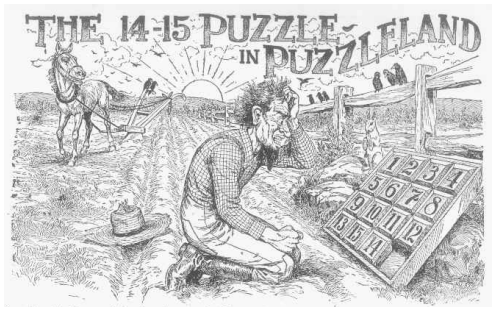
Links, met bladeren 1, 2, 3, 4, is de "linker" zet het beste, rechts, met bladeren 1, 20, 30, 400, de "rechter" zet.
De evaluatie-functie moet proportioneel zijn met de "winst".

Een deels observeerbaar spel is Kriegspiel: schaak waarbij de speler alleen de eigen stukken ziet, en een scheidsrechter zetten beoordeelt (“OK”, “Verboden”, “Schaak”, . . .).



Let ook op de “belief states”.

De derde programmeeropgave gaat over de 15-puzzel die we met A* en IDA* (verbeter!) in C++ programmeren. Mag je ook “schuin” schuiven? En dezelfde getallen?



www.liacs.leidenuniv.nl/~kosterswa/aster2024.html

```
./aster2024 drie.txt 25 50000 1 2 2 100 0 123
```

`drie.txt`: invoerfile, 25: max_lengte, 50000: geheugen, 1: oplosbaarheid, 2: heuristiek, 2: methode (3: IDA*), 100: aantal_spellen, 0: printen, 123: randomseed

De eerstvolgende keer zijn we nog met spellen bezig. Het huiswerk voor de daaropvolgende keer (3 april 2024): lees Hoofdstuk 6, p. 180–199 van [RN] door (in de derde druk p. 202-223) over het onderwerp Constrained Satisfaction Problemen.

Denk aan de tweede opgave: Agenten & Robotica; deadline: 22 maart 2024.
Werk daarna aan de derde opgave: A*.

En voor de liefhebbers: Combinatorial Game Theory.

Vergeet de "opgaven" = "sommen" niet, zie
www.liacs.leidenuniv.nl/~kosterswa/AI/opgaven1.pdf