Programmeermethoden

Algoritmen

Walter Kosters en Jonathan Vis

week 13: 6–10 december 2021

www.liacs.leidenuniv.nl/~kosterswa/pm/

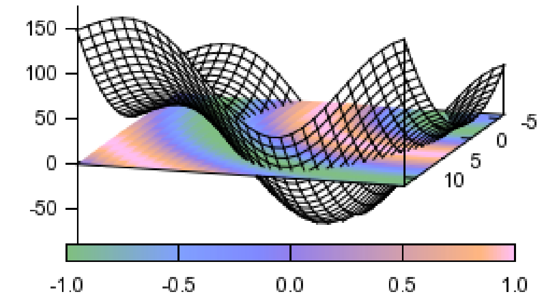
Koffiesweeper programmeren we als volgt:

- week 1: pointerpracticum, opgave lezen
- week 2: klassen, pointerstructuur aanleggen, elementair spelen
- week 3: finesses, recursie, stapel
- week 4: experiment (`gnuplot`), verslag

www.liacs.leidenuniv.nl/~kosterswa/pm/op4pm.php

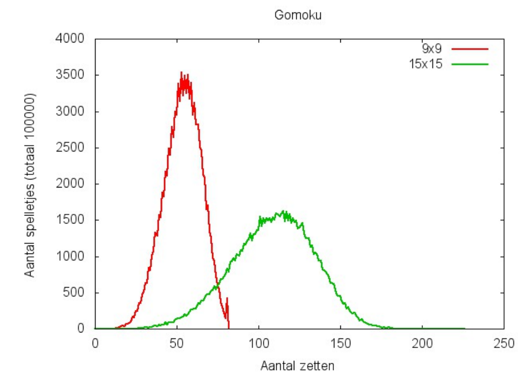
Gebruik gnuplot om een eenvoudige grafiek te maken, ook in Windows:

```
gnuplot> plot "stats.txt" with lines
```



Hierbij is de file `stats.txt` zoiets als:

```
1  7
2  12
3  14
```



Zie www.gnuplot.info.

Op allerlei colleges en in allerlei boeken en artikelen worden algoritmen behandeld, bijvoorbeeld bij de volgende Informatica-colleges in Leiden (semester tussen haakjes):

- Programmeermethoden (1), Algoritmiek (2)
- Datastructuren (3), Kunstmatige intelligentie (4)
- . . .

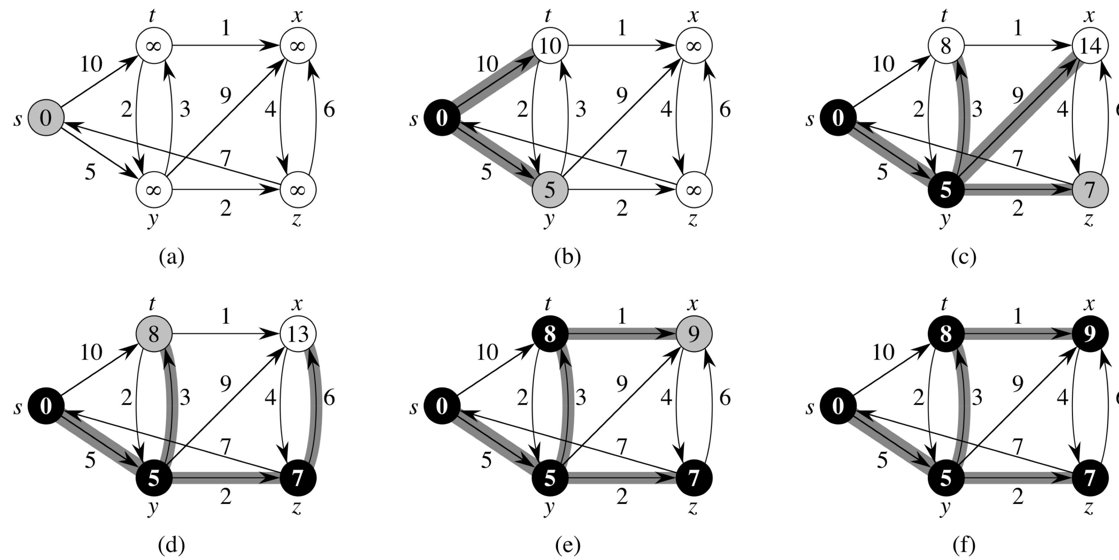
Je kunt algoritmen op allerlei manieren rubriceren, bijvoorbeeld met behulp van de volgende begrippen:

verdeel en heers, numerieke wiskunde, graafalgoritmen,
dynamisch programmeren, patroonherkenning, adversary,
data mining, geometrisch modelleren, backtracking,
benaderende algoritmen, kunstmatige intelligentie,
neurale netwerken, evolutionaire algoritmen, P$\leftrightarrow$NP,
gretige algoritmen, snelle Fouriertransformatie,
...

We bekijken er een paar van.

Gegeven een graaf G, met afstanden op de takken, en twee knopen a en b. Gevraagd: kortste pad van a naar b.

Oplossing: het algoritme van Dijkstra.



Als n een priemgetal is, geldt dat $a^{n-1} - 1$ deelbaar is door n voor $a = 1, 2, \ldots, n - 1$. Het omgekeerde is bijna waar.

Dit suggereert het volgende algoritme dat "bepaalt" of n een priemgetal is: Als $2^{n-1} - 1$ niet deelbaar is door n is n zeker geen priemgetal, en anders (misschien) wel. Dit gaat fout bij 341, 561, . . . , maar dat is te verbeteren: probeer andere a; echter, 561, een Carmichael-getal, blijft lastig. (Uiteindelijk: Miller-Rabin.)

Het algoritme is een randomized algoritme, en wel een Monte Carlo algoritme: het ene antwoord is altijd juist, het anders soms niet. Bij Las Vegas algoritmen zijn de antwoorden altijd juist — maar het duurt soms lang.

En hoe maak je een willekeurige permutatie, dat wil zeggen, een random volgorde van de getallen $1, 2, \ldots, n$?

```
// stop random permutatie van 0,1,...,n-1 in array A
void maakpermutatie (int A[ ], int n) {
  int i;  // array-index
  int r;  // random array-index
  for ( i = 0; i < n; i++ ) A[i] = i;
  for ( i = n-1; i >= 0; i-- ) {
    r = rand ( ) % ( i+1 );  // 0 <= r <= i, random
    wissel (A[i],A[r]);
  }//for
}//maakpermutatie
```

`rand ( )` geeft een random-getal; `srand (42)` zet het "seed".

Een gretig (greedy) algoritme neemt beslissingen door één stap vooruit te kijken; het zijn meestal benaderende algoritmen.

We bekijken het volgende algoritme voor het Common Superstring probleem, dat vraagt naar een (zo kort mogelijke) string die een stel gegeven strings bevat: Neem herhaald de twee meest overlappende strings bij elkaar.

Een voorbeeld. Begin met `TCAGT`, `CATCAG`, `GTG` en `GCA`.
De twee meest overlappende strings zijn `CATCAG` en `TCAGT`; vervang deze door `CATCAGT`, we hebben dan `CATCAGT`, `GTG` en `GCA` over.
Zowel `GTG` als `GCA` hebben een overlap van 2 met `CATCAGT`.
Kies bijvoorbeeld `GTG`, wat `CATCAGTG` en `GCA` oplevert.
De eindoplossing is `GCATCAGTG`, en die is toevallig optimaal.

Nog een voorbeeld: begin met `GCC`, `ATGC` en `TGCAT`.
De twee meest overlappende strings zijn `ATGC` en `TGCAT`; vervang deze door `ATGCAT`, en we houden `ATGCAT` en `GCC` over.
Deze twee strings hebben geen overlap, dus de eindoplossing is hun "concatenatie": `ATGCATGCC` of `GCCATGCAT`, beide van lengte 9. De optimale oplossing, `TGCATGCC`, heeft echter lengte 8!

Het algoritme vindt wel snel een superstring, maar niet altijd een optimale . . .

Algemener: begin met

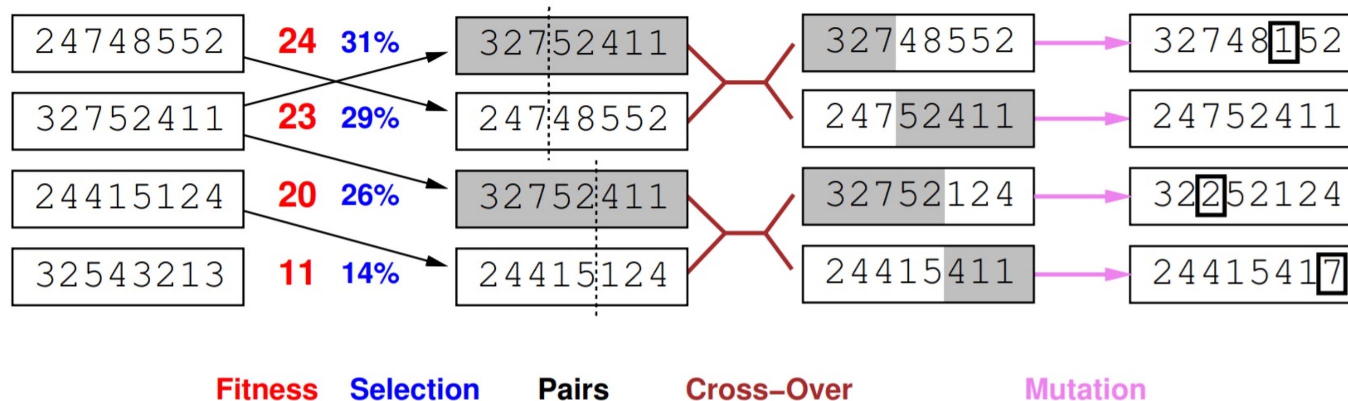
$$\{\mathtt{C}(\mathtt{AT})^k, (\mathtt{TA})^k, (\mathtt{AT})^k\mathtt{G}\}$$

voor een vaste $k \geq 1$, dan levert het algoritme $\mathtt{C}(\mathtt{AT})^k\mathtt{G}(\mathtt{TA})^k$ ter lengte $4k + 2$, terwijl de optimale string $\mathtt{C}(\mathtt{AT})^{k+1}\mathtt{G}$ lengte $2k + 4$ heeft.

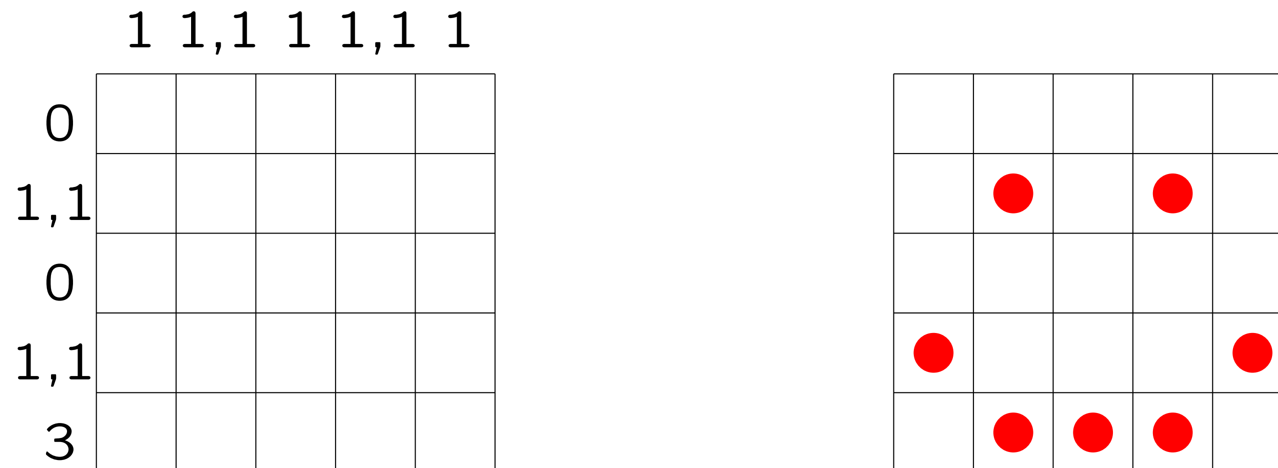
De uitvoer van het gretige algoritme kan dus twee keer zo lang zijn dan de optimale. Maar erger wordt het niet (open problemen).

Dit soort algoritmen wordt gebruikt bij DNA-reconstructie: de “shotgun-methode”.

Genetische algoritmen, of algemener Evolutionaire algoritmen, evolueren een populatie met kandidaat-oplossingen voor een probleem. Van elke kandidaat-oplossing kun je de kwaliteit berekenen met een fitness-functie. De beste individuen gaan door, en met mutatie (willekeurige, kleine veranderingen) en crossover (combineer twee “ouders”) krijg je een nieuwe generatie.



Japanse puzzels (Nonogrammen) zien er zo uit:



Naast iedere rij en boven iedere kolom staan in volgorde de lengtes van aaneengesloten series rode blokjes. Zie de derde programmeeropgave van vorig jaar.

www.liacs.leidenuniv.nl/~kosterswa/nono/

Genetische algoritmen kunnen gebruikt worden om Nonogrammen op te lossen.

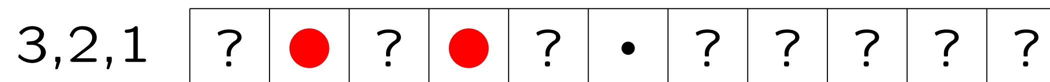
Een individu is hier een string (of array) met 25 bits (algemener, voor een m bij n puzzel, met mn bits), waarbij een 1 rood en een 0 leeg voorstelt. Je kunt er voor kiezen om het aantal enen per rij altijd "goed" te houden.

Mutatie zou bijvoorbeeld in een rij een 1 en een 0 kunnen verwisselen. Als je handig muteert kun je "alles" bereiken!

De fitness-functie is een som over rijen en kolommen. Per rij/kolom geef je aan hoeveel je van de specificatie afwijkt — en dat is nog lastig precies te maken.

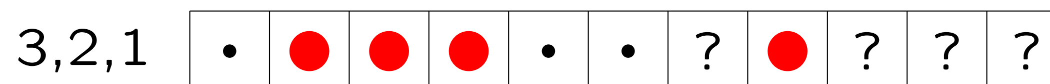
Maar het is wellicht beter om Dynamisch Programmeren te gebruiken om Nonogrammen op te lossen.

De vraag is wat bij een lijn (rij of kolom) kan worden afgeleid, gegeven een deelinvulling:



Een • betekent een zeker leeg vakje, een ● staat voor een zeker gevuld vakje. De rest is nog onbekend.

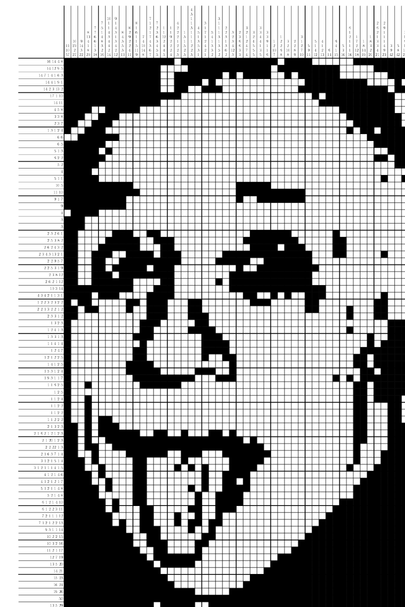
Dan concluderen we:



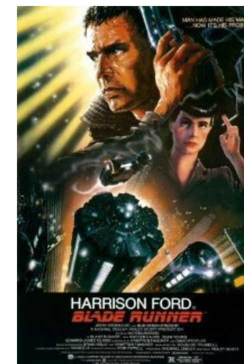
Hoe helpt Dynamisch Programmeren hierbij?

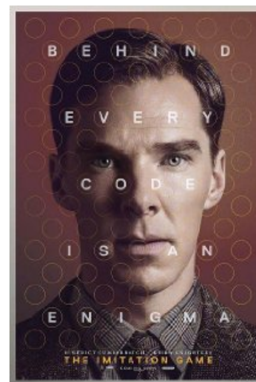
In plaats van alle manieren te bedenken waarmee de lijn kan worden gevuld (en te kijken wat deze gemeenschappelijk hebben) kun je ook tabellen maken om te zien hoe dit zit met deelrijen en deelbeschrijvingen!

We berekenen L_{ij}: kan de (deel)beschrijving $d_1, d_2, \ldots, d_i$ worden gerealiseerd in de (deel)string $s_1, s_2, \ldots, s_j$; en zo ja: wat "moet"? Hierbij: $0 \leq i \leq$ lengte beschrijving en $0 \leq j \leq$ lengte string.



Kunstmatige intelligentie





IN 2013 ROB FORD, MAYOR OF THIS 4th-LARGEST CITY IN N. AMERICA, FIRST SAID HE SMOKED WEED, NOT CRACK...THEN YES, OK, CRACK, TOO

2011

What is Toronto????

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

3	9	8
2	4	6
14	5	2

❶ ❷

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?

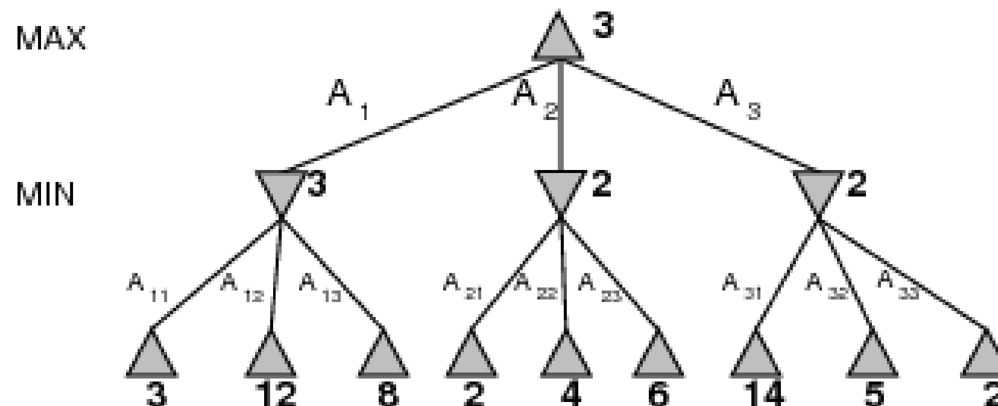
Is Maxi rij 1 kiest, kiest Mini kolom 1 (levert 3); als Maxi rij 2 kiest, kiest Mini kolom 1 (levert 2); als Maxi rij 3 kiest, kiest Mini kolom 3 (levert 2). Dus kiest Maxi rij 1!

3	9	8
2	**?**	**?**
14	5	2

Nu merken we op dat de analyse 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.

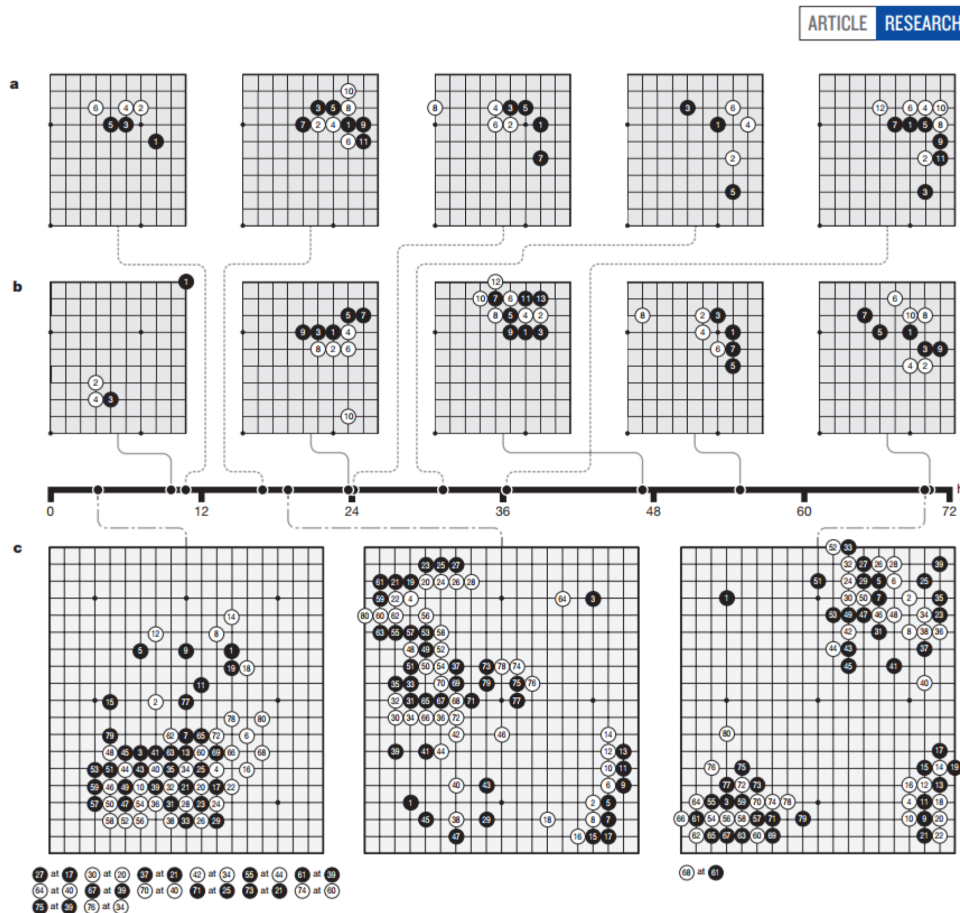
Ieder "oud" schaakprogramma gebruikt deze methode.

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.



breaking news

arXiv:1712.01815v1 [cs.AI] 5 Dec 2017

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver,[1*] 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]

[1]DeepMind, 6 Pancras Square, London N1C 4AG.

[*]These authors contributed equally to this work.

Abstract

The game of chess is the most widely-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. In contrast, the *AlphaGo Zero* program recently achieved superhuman performance in the game of Go, by *tabula rasa* reinforcement learning from games of self-play. In this paper, we generalise this approach into a single *AlphaZero* algorithm that can achieve, *tabula rasa*, superhuman performance in many challenging domains. Starting from random play, and given no domain knowledge except the game rules, *AlphaZero* achieved within 24 hours a superhuman level of play in the games of chess and shogi (Japanese chess) as well as Go, and convincingly defeated a world-champion program in each case.

The study of computer chess is as old as computer science itself. Babbage, Turing, Shannon, and von Neumann devised hardware, algorithms and theory to analyse and play the game of chess. Chess subsequently became the grand challenge task for a generation of artificial intelligence researchers, culminating in high-performance computer chess programs that perform at superhuman level (*9, 13*). However, these systems are highly tuned to their domain, and cannot be generalised to other problems without significant human effort.

A long-standing ambition of artificial intelligence has been to create programs that can instead learn for themselves from first principles (*26*). Recently, the *AlphaGo Zero* algorithm achieved superhuman performance in the game of Go, by representing Go knowledge using deep convolutional neural networks (*22, 28*), trained solely by reinforcement learning from games of self-play (*29*). In this paper, we apply a similar but fully generic algorithm, which we

1

Silver et al., Nature 550 (2017) 354–359: Deep learning, TPU's, . . .

Nature, oktober 2019: breaking news

Article

Grandmaster level in StarCraft II using multi-agent reinforcement learning

https://doi.org/10.1038/s41586-019-1724-z

Received: 30 August 2019

Accepted: 10 October 2019

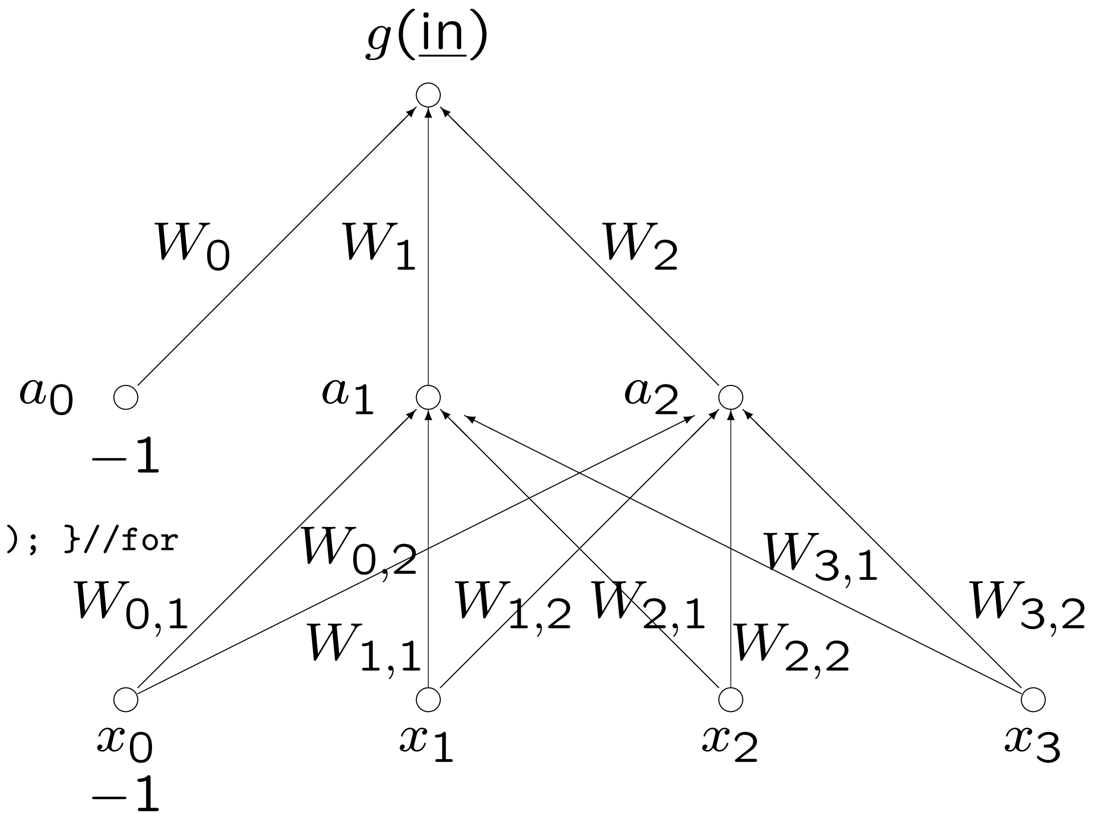
Published online: 30 October 2019

Oriol Vinyals[1,3*], Igor Babuschkin[1,3], Wojciech M. Czarnecki[1,3], Michaël Mathieu[1,3], Andrew Dudzik[1,3], Junyoung Chung[1,3], David H. Choi[1,3], Richard Powell[1,3], Timo Ewalds[1,3], Petko Georgiev[1,3], Junhyuk Oh[1,3], Dan Horgan[1,3], Manuel Kroiss[1,3], Ivo Danihelka[1,3], Aja Huang[1,3], Laurent Sifre[1,3], Trevor Cai[1,3], John P. Agapiou[1,3], Max Jaderberg[1], Alexander S. Vezhnevets[1], Rémi Leblond[1], Tobias Pohlen[1], Valentin Dalibard[1], David Budden[1], Yury Sulsky[1], James Molloy[1], Tom L. Paine[1], Caglar Gulcehre[1], Ziyu Wang[1], Tobias Pfaff[1], Yuhuai Wu[1], Roman Ring[1], Dani Yogatama[1], Dario Wünsch[2], Katrina McKinney[1], Oliver Smith[1], Tom Schaul[1], Timothy Lillicrap[1], Koray Kavukcuoglu[1], Demis Hassabis[1], Chris Apps[1,3] & David Silver[1,3*]

Many real-world applications require artificial agents to compete and coordinate with other agents in complex environments. As a stepping stone to this goal, the domain of StarCraft has emerged as an important challenge for artificial intelligence research, owing to its iconic and enduring status among the most difficult professional esports and its relevance to the real world in terms of its raw complexity and multi-agent challenges. Over the course of a decade and numerous competitions[1–3], the strongest agents have simplified important aspects of the game, utilized superhuman capabilities, or employed hand-crafted sub-systems[4]. Despite these advantages, no previous agent has come close to matching the overall skill of top StarCraft players. We chose to address the challenge of StarCraft using general-purpose learning methods that are in principle applicable to other complex domains: a multi-agent reinforcement learning algorithm that uses data from both human and agent games within a diverse league of continually adapting strategies and counter-strategies, each represented by deep neural networks[5,6]. We evaluated our agent, AlphaStar, in the full game of StarCraft II, through a series of online games against human players. AlphaStar was rated at Grandmaster level for all three StarCraft races and above 99.8% of officially ranked human players.

Kunnen computers denken?

```
for ( j = 1; j <= hs; j++ ) {
  s[j] = -ItH[0][j];
  for ( k = 1; k <= ip; k++ )
    s[j] += ItH[k][j] * I[k];
  HtA[j] = g (s[j]); }//for
for ( i = 0; i < os; i++ ) {
  sO = -HtO[0][i];
  for ( j = 1; j <= hs; j++ )
    sO += HtO[j][i] * HtA[j];
  O[i] = g (sO);
  dO[i] = gp (sO) * ( T[i] - O[i] ); }//for
for ( j = 1; j <= hs; j++ ) {
  d[j] = 0;
  for ( i = 0; i < os; i++ )
    d[j] += HtO[j][i] * dO[i];
  d[j] *= gp (sum[j]); }//for
for ( j = 0; j <= hs; j++ )
  for ( i = 0; i < os; i++ )
    HtO[j][i] += a * HtA[j] * dO[i];
for ( k = 0; k <= ip; k++ )
  for ( j = 1; j <= hs; j++ )
    ItH[k][j] += a * I[k] * d[j];
```



Aan een spel als Tetris kleven allerlei vragen:

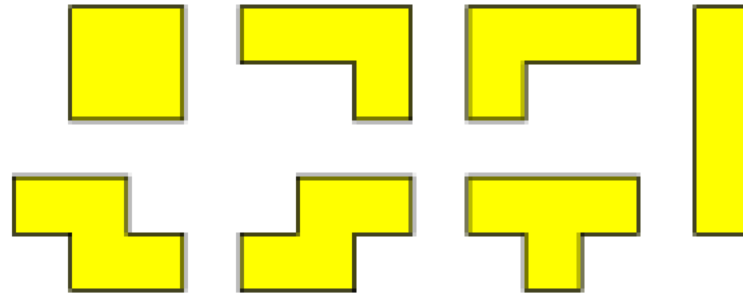
- Hoe speel je het zo goed mogelijk?
  (AI = Kunstmatige intelligentie)

- Hoe moeilijk is het? (complexiteit)

- Wat kan er allemaal gebeuren?

Zo is bijvoorbeeld bewezen dat sommige Tetris-vragen NP-volledig zijn (gezamenlijk werk met mensen van MIT), dat je bijna alle configuraties kunt bereiken, maar dat niet alle problemen “beslisbaar” zijn.

De 7 Tetris-stukken:



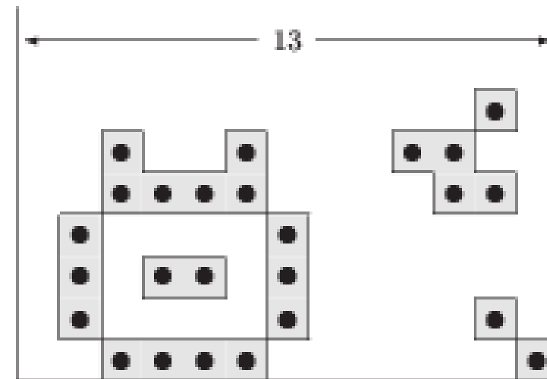
De vraag “Kun je met een gegeven serie (inclusief volgorde) van deze stukken een deels al gevuld bord helemaal leeg spelen?” is NP-volledig.
Als iemand het bord leeg speelt kun je dat eenvoudig controleren. Als het *niet* kan, kan men (tot nu toe) niks beters verzinnen dan alle mogelijkheden één voor één na te gaan!

P en NP zijn “klassen” van beslissingsproblemen. Het probleem of een graaf samenhangend is, zit in P. Het probleem of een graaf een Hamilton-circuit heeft, zit in NP, en is zelfs NP-volledig; je kunt het “eenvoudig”, maar niet efficiënt, oplossen met bruteforce technieken.

P is de klasse van beslissingsproblemen die door een “deter-mistische Turing-machine” in “polynomiale tijd” kunnen worden opgelost. NP is de klasse van beslissingsproble-men die door een “niet-determistische Turing-machine” in “polynomiale tijd” kunnen worden opgelost: je mag “gokken”.

Open probleem: geldt P = NP?

Een “willekeurige” configuratie:



Deze kan gemaakt worden door 276 geschikte Tetris-stukken op de juiste plaats te laten vallen.

Claim: op een bord van oneven breedte kan elke configuratie bereikt worden!

www.liacs.leidenuniv.nl/~kosterswa/tetris/

Opgave 1b van het tentamen van 6 januari 2020:

In het array `int A[n]` staan `n` (een `const int` ≥ 3) verschillende gehele getallen.
Schrijf een C++-functie `int stijg (A,b,n)` die de lengte van een langste stijgende aaneengesloten deelrij van `A` geeft, en diens begin-index in `b` oplevert. Als er meerdere deelrijen het langste zijn: de kleinste `b`. Dus array `2 7 4 5 6 1 3 8` geeft 3, met `b` = 2 (deelrij `4 5 6`, even lang als `1 3 8`).

```
b. int stijg (int A[ ], int & b, int n ) {
     int i, langste = 1, lang = 1, begin = 0; b = 0;
     for ( i = 1; i < n; i++ )
       if ( A[i] > A[i-1] ) {
         lang++;
         if ( lang > langste ) { langste = lang; b = begin; }//if
       }//if
       else {
         begin = i; lang = 1;
       }//else
     return langste;
   }//stijg
```

Opgave 2 van het tentamen van 6 januari 2015:

a. Bij een functie kun je te maken hebben met *call by value* en *call by reference*, en ook met *locale* en *globale* variabelen. Verder onderscheiden we ook nog *formele* en *actuele* parameters. Leg deze zes begrippen duidelijk uit.

b. Gegeven een C++-programma met daarin de volgende twee functies:

```
int ludo (int a, int b, int n) {
  int i = 42; for ( i = 0; i < n; i += 2 ) { b += a; i--; }//for
  return b; }//ludo
int jeanine (int a, int b) {
  a = ludo (a,b,a); cout << a << "," << b << endl;
  a = ludo (a,b,a); cout << a << "," << b << endl;
  return a; }//jeanine
```

Verder zijn de globale variabelen `x` en `y` gegeven (van type `int`). Wat is dan de uitvoer van het volgende stukje programma (leg je antwoord duidelijk uit):

```
x = 2; y = 2; y = jeanine (x,y); cout << x << "," << y << endl;
```

c. Als **b**, maar nu met een `&` ("ampersand") bij de vijf parameters van de functies.

d. Geef een eenvoudige uitdrukking voor de return-waarde van een aanroep `ludo (a,b,n)`, uitgedrukt in diens parameters. Het maakt niet uit of er `&`'s bij de parameters staan.

e. Als **d**, maar nu voor `jeanine (a,b)`. Neem aan dat er bij alle vijf parameters een `&` staat, net als bij **c**.

f. Als in de functie `ludo` ergens `a = jeanine (a,2*n);` staat, compileert het programma dan nog? Onderscheid gevallen met en zonder `&`.

- werk aan de vierde programmeeropgave — de deadline is op maandag 13 december 2021
- volgende week laatste college, over Talen als Python & oude tentamens
- www.liacs.leidenuniv.nl/~kosterswa/pm/