Samenvatting Introduction Computational Science

Lucas Riedstra

20 februari 2019

Inhoudsopgave

1 Introductie

Definitie 1.1. Een *systeem* is simpelweg een onderscheidbaar deel van het universum, of een potentiële bron van data. Variabelen die door de omgeving buiten het systeem worden "gemaakt" en gedrag van het systeem beïnvloeden noemen we *inputs*, variabelen die door het systeem worden "gemaakt" en het gedrag van de omgeving beïnvloeden noemen we *outputs*.

Een *experiment* is een proces waarbij data uit een systeem wordt verkregen, waarbij "gespeeld" kan worden met de inputs.

Een *model* M voor een experiment E bij een systeem S kan alles zijn waarbij E kan worden toegepast om een vraag over S te beantwoorden. Merk op dat een model zelf ook een systeem is. We onderscheiden twee soorten modellen: *wiskundige* en *natuurkundige* modellen.

Wiskundige modellen zijn modellen die gerepresenteerd kunnen worden door een verzameling vergelijkingen, die soms analytisch kunnen worden opgelost maar waarvoor meestal een *simulatie* nodig is - een experiment uitgevoerd op een wiskundig model. Bij natuurkundige modellen wordt iets getest "in de echte wereld", d.w.z. door een opstelling te bouwen.

Validatie is het proces van controleren dat een model accuraat genoeg is, *verificatie* is controleren dat er geen fouten in het model zitten (bv. bugs bij een simulatie).

Een veelgemaakte fout bij modellen is geen rekening houden met het *domein* van het model: de verzameling parameters waarbij de output van een model nog accuraat is. Hierdoor kunnen resultaten aangenomen worden uit het model die niet kloppen.

Bij dit vak houden we ons bezig met wiskundige modellen.

Definitie 1.2. We onderscheiden drie soorten wiskundige modellen:

- *Continue-tijd-modellen* zijn modellen waarbij de staat van het systeem continu verandert als functie van de tijd;

- *Discrete-tijd-modellen* zijn modellen waarbij de staat van het systeem in discrete tijdstappen verandert;
- *Discrete-gebeurtenis-modellen* zijn modellen waarbij de staat van het systeem verandert bij bepaalde gebeurtenissen in de tijd.

Voorbeeld 1. Er zijn veel verschillende redenen om een simulatie te gebruiken in plaats van het systeem zelf: soms is het systeem niet beschikbaar, of is het experiment te gevaarlijk of te duur. De tijd bij bepaalde gebeurtenissen in het systeem kan te kort (picosecondes) of te lang (miljoenen jaren) zijn. Inputs kunnen onbewerkbaar zijn (bv. het weer), en het systeem kan te veel ruis hebben.7

Definitie 1.3. We kunnen een systeem zien als een "functie" $f_S(I) = O$, met S de staat van het systeem, I de inputs en O de outputs. In de gewone situatie zijn I en S bekend, en willen we O bepalen. Als het omgekeerde het geval is, d.w.z. de outputs en de staat zijn bekend en we willen weten wat de inputs waren, spreken we van een *invers probleem*. Als zowel de inputs I als de outputs O bekend zijn, maar we kennen de structuur of de staat van het systeem niet, spreken we van een *structuur- of staatidentificatieprobleem*.

2 Cellulaire automaten

2.1 Definitie en gedrag

Definitie 2.1. Een *cellulaire automaat* (CA) is een n-dimensionaal grid van cellen ($n \geq 1$), waarbij dat grid meestal carthesisch is (vierkantjes/lijnstukjes), maar bv. ook hexagonaal o.i.d. kan zijn. Elke cel heeft een eindig aantal staten, en de nieuwe staat van een cel hangt af van (a) de huidige staat van die cel en (b) de staat van cellen in een bepaalde omgeving van die cel. *De nieuwe staat van een cel hangt dus niet af van de tijd*, en dus vertoont een CA deterministisch gedrag. Een CA is een voorbeeld van een discrete-tijd-model. Bij een *synchrone* CA wordt de staat van elke cel op hetzelfde moment bijgewerkt, bij een *asynchrone* CA niet.

Een CA heeft veel nut:

- De dynamica van een CA is simpel, maar geeft alsnog complexe patronen;
- Echte fysische processen kunnen worden weergegeven met CA's (botsen van deeltjes o.i.d.);
- We kunnen discrete dynamische systemen simuleren met CA's;
- De implementatie van een CA is makkelijk paralleliseerbaar, zeker als de CA synchroon is.

In het 1D-geval bepalen we een CA met het aantal mogelijke staten k van een cel, en de "straal" r van een cel: een cel met straal 2 bekijkt voor het bepalen van een nieuwe staat zijn twee linkerburen en zijn twee rechterburen, en zichzelf. Voor het bepalen van de nieuwe staat van een cel is dus een input nodig van $2r+1$ cellen. Voor die $2r+1$ cellen zijn er k staten per cel, dus in totaal zijn er k^{2r+1} mogelijke configuraties.

Merk op dat we een probleem hebben aan de rand: daar hebben we geen $2r+1$ buren. Hiervoor zijn twee mogelijkheden: ofwel *blocking* gedrag, waarbij we doen alsof er aan de rand oneindig veel nullen zijn, ofwel *cyclic*, waarbij de buurman van de linkercel de meest rechtercel is en zo verder.

Een *regel* is een functie die aan elk van de k^{2r+1} mogelijke configuraties een staat toekent (en in totaal zijn er dus $k^{k^{2r+1}}$ mogelijke regels). Het "alfabet" van een CA is simpelweg de verzameling mogelijke configuraties. Met k en r vast kunnen we een regel weergeven als string met k^{2r+1} staten: de output bij elke configuratie (waarbij we de configuraties op aflopende grootte nummeren). Deze string kunnen we ook zien als base-k-getal, en door deze te converteren naar base 10 kunnen we regels nummeren in ons eigen getallensysteem.

Een 1D-CA wordt weergegeven door de nieuwe rij cellen direct onder de huidige rij te zetten, dit geeft dus een 2D-grid dat naar beneden uitbreidt. We kunnen de verschillende regels nu in het geval $k = 2, r = 1$ classificeren (met configuraties van lengte $2r + 1 = 3$, en dus regels van lengte $k^{2r+1} = 2^3 = 8$, en in totaal $k^{k^{2r+1}} = 2^{2^3} = 256$ mogelijke regels), dit geeft de zogeheten *Wolfram-klasses*:

1. Regels die na zekere tijd homogeen gedrag veroorzaken (dus alleen maar 0 of alleen maar 1);
2. Regels die na zekere tijd stabiel of oscillerend gedrag veroorzaken (bijvoorbeeld een lijn, of even grote driehoekjes onder elkaar);
3. Regels die chaotisch, pseudo-random gedrag veroorzaken;
4. Regels die complexe structuren genereren. Eventueel kunnen deze structuren na lange tijd weer klasse 2-structuren worden (dus stabiel). Het typische voorbeeld is regel 110, die een universele turingmachine kan simuleren.

Definitie 2.2. In het geval 1D-geval definiëren we bij een regel de *transient path* als het aantal stappen totdat het systeem een regelmatige staat bereikt (dit getal is vaak groot bij klasse 4-regels), en de *cycle length* als het aantal stappen in een zichzelf herhalende structuur, als zoiets aanwezig is (dit getal is vaak groot bij klasse 3-regels). Logischerwijs hangt de *transient path* af van de start-seed, meestal wordt deze voor een regel uitgerekend door van alle mogelijke start-seeds dit getal uit te rekenen en vervolgens het gemiddelde te nemen.

Een *Garden-of-Eden*-staat bij een bepaalde regel en start-seed is een staat waar je niet meer naar kan terugkeren. Simpel voorbeeld: als we beginnen met een rij van zwarte cellen en regel 0 toepassen, is de eerste staat een Garden-of-Eden-staat (want daarna wordt alles wit).

2.2 Classificatie van complexiteit

Elke Wolfram-klasse van CA's is nuttig: klasses 1 en 2 voor een deterministische berekening bij een bepaalde input, klasse 3 voor pseudo-random getallen, en klasse 4 voor bijvoorbeeld Turingmachines. We hebben nu twee doelen: het maken van regels met een bepaalde complexiteit, en het classificeren van een gegeven regel.

We beginnen met ons eerste doel:

Definitie 2.3. Bij een eendimensionale CA definiëren we de *Langtonparameter* van een regel Δ als volgt: we kiezen een *nulstaat* s_q. Vervolgens tellen we het aantal configuraties waaraan deze staat s_q toekent, dit noemen we n. We definiëren $\lambda(\Delta)$ nu als volgt:

$$\lambda(\Delta) = \frac{k^{2r+1} - n}{k^{2r+1}},$$

dus het aantal configuraties waaraan Δ níét s_q toekent gedeeld door het totaal aantal configuraties.

Als λ 0 of 1 is, is de regel duidelijk klasse 1. Als alle staten even vaak voorkomen in Δ geldt $\lambda = 1 - k^{-1}$, hierbij hoort meestal chaotisch gedrag.

Stappenplan 2.4. We kennen twee manieren om, gegeven een zekere $\lambda \in [0, 1]$, een regel Δ te maken zodat $\lambda(\Delta) = \lambda$:

1. *Random table*: Voor elke plek i in de tabel van Δ genereren we een willekeurig getal g uit de uniforme verdeling op $[0, 1]$. Als $g > \lambda$ stellen we $i = s_q$, anders kennen we aan i een willekeurige andere staat toe.

   Deze methode geeft dus, bij een bepaalde λ, een regel Δ waarvoor $\lambda(\Delta)$ met hoge waarschijnlijkheid in de buurt ligt van λ.

- **i.e., given the state *s(t)*, count neighborhood configurations**

Neighborhood	Count	p_i	$p_i \log_2 p_i$
111	1	1/12	-0.298746875
110	3	3/12	-0.5
101	1	1/12	-0.298746875
100	2	2/12	-0.430827083
011	3	3/12	-0.5
010	0	0/12	NA
001	2	2/12	-0.430827083
000	0	0/12	NA

$$H(X) = -\sum_{i=1}^{k} p_i \log p_i = 2.459$$

Figuur 1: Een simpel voorbeeld van Shannonentropie bij een 1D-CA met $k = 2, r = 1$.

2. *Table walk-through*: Deze methode geeft een manier om de Langtonparameter van Δ in "stapjes" te verhogen. Initialiseer alle staten van Δ op s_q. We beginnen dus met $\lambda(\Delta) = 0$.

   Vervolgens kiezen we de volgende λ' die we willen hebben. Om van λ naar λ' te komen kiezen we $(\lambda' - \lambda) \cdot k^{2r+1}$ staten met s_q willekeurig, en vervangen we s_q daar door een willekeurige andere staat.

Overal waar staat "ken een willekeurige andere staat toe", moet dit ook echt willekeurig gedaan worden: als we drie staten $\{0, 1, 2\}$ hebben, we hebben $s_q = 0$, en we steeds 1 toe als we een 0 moeten vervangen, krijgen we geen complex gedrag. Hierdoor is λ geen zeer goede maatstaaf voor classificatie van een gegeven regel.

Voor $\lambda = 0$ wisten we al dat "saai" gedrag vertoond wordt. Nu blijkt dat bij verhogen van λ dit verandert in periodiek gedrag. Verhogen we λ nog meer, verschijnt er complex gedrag en als λ nóg meer verhoogd wordt krijgen we chaotisch gedrag (met een piek op $\lambda = 1 - k^{-1}$). In de taal van Wolframklasses gaan we dus van 1 naar 2 naar 4 naar 3.

We gaan verder met ons tweede doel: het classificeren van de complexiteit van een gegeven regel Δ. Een simpele manier om de complexiteit te classificeren is door de lengte van de *transient path* te meten: deze is vaak lang bij chaotische (klasse 3-)systemen, en blijkt exponentieel te groeien met de grootte van het systeem bij $\lambda = 1 - k^{-1}$ (de meest chaotische λ).

Een andere mogelijkheid is door middel van *Shannonentropie*: een manier om informatie te kwantificeren.

Definitie 2.5. Zij X een stochast op een zekere eindige verzameling Z met kansmassafunctie p, dan definiëren we de *entropie* van X als

$$H(X) = -\sum_{z \in Z} p(z) \log_2 p(z).$$

Bij een CA hebben we geen "random events", maar we kunnen bij een staat s voor elke configuratie c (bijvoorbeeld '111' of '010') de "waarschijnlijkheid" van c in s definiëren als het aantal keer dat deze voorkomt, gedeeld door de grootte van de CA. Zie figuur 1.

2.3 2D-CA: The game of life

Definitie 2.6. Een 2D-CA hebben we al correct gedefinieerd, op de omgeving na. Bij een carthesisch grid (vierkantjes) hebben we twee bekende omgevingen:

- Als een cel wordt beïnvloed door de vier direct aanliggende cellen, heet dit de *Von Neumann-omgeving*.
- Als een cel wordt beïnvloed door de acht omringende cellen, heet dit de *Moore*-omgeving.7

Natuurlijk is ook een hexagonaal grid mogelijk.

Bij een 2D-CA hebben we opnieuw het probleem van de rand. Weer hebben we twee oplossingen: *blocking*, waarbij we doen alsof er om ons grid heen oneindig veel nullen zijn, en *periodic*, waarbij we één of andere wrap-around doen.

Definitie 2.7. Eén specifieke 2D-CA is zo beroemd dat hij een eigen naam heeft: de *Game of Life*, of korter, *Life*. Deze CA met twee staten ('levend' en 'dood') gebruikt een *Moore*-omgeving met blocking gedrag aan de rand, en de volgende regels:

- Een levende cel sterft, tenzij hij 2 of 3 buren heeft.
- Een dode cel komt tot leven als deze precies 3 buren heeft.

Life is geboren uit een simpele vraag: kunnen simpele structuren zelf complexere structuren genereren? Het antwoord is ja.

Stelling 2.8. *The game of life kan een universele turingmachine simuleren.*

Bewijs. Ten eerste hoeven we alleen rekening te houden met binair geheugen en binaire berekeningen, aangezien alles ook binair kan. Een turingmachine heeft een aantal dingen nodig:

- *Geheugen*: dit kunnen we representeren met *gliders*: stabiele structuren die elke tijdstap dezelfde stap zetten (bv. een aantal plekken omlaag).
- *Clock*: we moeten objecten met een vaste snelheid kunnen bewegen om de "tape head" te verplaatsen. Bewegende structuren hebben we (*gliders*, *spaceships*, *cars*), we moeten ze ook kunnen genereren. Dit kan met *Glider guns*.
- Elke functie moet berekend kunnen worden. Omdat we z.v.v.a. binair rekenen moeten we elke Booleaanse functie kunnen berekenen. En aangezien elke Booleaanse functie te schrijven is als e.o.a. samenstelling van NAND-gates, hoeven we alleen maar de NAND-gate te kunnen simuleren. Hiervoor moeten we dus een NOT-gate en een AND-gate maken.

  Volgende filmpje legt dit erg goed uit: `https://www.youtube.com/watch?v=vGWGeund3eA`.

We concluderen dit "bewijs". □

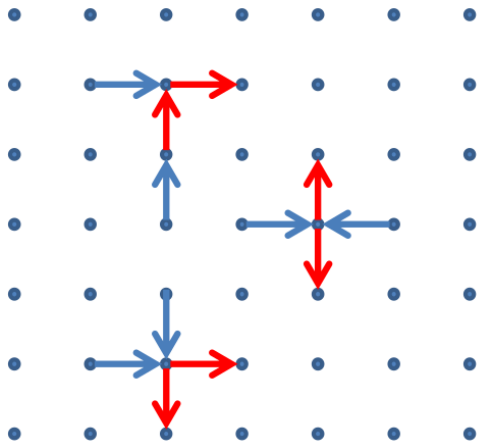
2.4 Voorbeelden van grid-based modellen

2.4.1 Bosbranden

Bij bosbranden zijn de belangrijke vragen *hoe* en *hoe snel* het vuur zich verspreidt - daarbij horen natuurlijk ook vragen als hoe de brand het best kan worden aangepakt.

Het model is een simpele CA met een Von Neumann-grid (elke cel wordt beïnvloed door zichzelf en de vier direct aanliggende), en elke cel representeert een boom. De vier staten zijn: niets (geen boom), levende boom, brandende boom, of dode boom. De rand heeft de vorm van een torus (3D-cirkelring) Er zijn maar twee regels:

- Een boom gaat in de fik als een buurman in brand staat;
- Een brandende boom gaat de volgende tijdstap uit (dus vuur duurt één stap).



Figuur 2: Transities in een gasmodel (blauwe pijlen zijn huidige richting, rood is richting in volgende stap).

Een belangrijk resultaat is dat er een grote piek zit in de percentage van het bos dat afbrandt: tot ongeveer 50% bezetting van de cellen blijft het bos gewoon leven (1 à 2 procent verwoesting), maar rond 60% schiet dit percentage ineens naar meer dan 75 procent. Hoe groter we het percentage vervolgens maken, hoe sneller het bos afsterft.

Dit soort modellen kunnen natuurlijk veel ingewikkelder met bv. stochastische processen zoals bliksem of veel meer staten om ook de leeftijd van de boom en/of de brandbaarheid mee te rekenen.

2.4.2 Gasmodellen

De staten in een gasmodel zijn (a) er is geen gasdeeltje of (b) er is een deeltje, waarbij de richting ook in de staat verwerkt is. Bij elke transitie hebben we twee mogelijkheden voor een deeltje: of het beweegt door in de huidige richting, of het botst met een ander deeltje. Bij een botsing blijven massa en momentum behouden. Zie figuur 2.

Het probleem is dat dit model zich niet volledig natuurlijk gedraagt omdat het een carthetisch (vierkant) grid gebruikt, het is een beter idee om een hexagonaal grid te gebruiken. Dan moeten er wel stochastische keuzes gemaakt worden bij botsingen, terwijl we in het carthetisch grid steeds één mogelijkheid hebben.

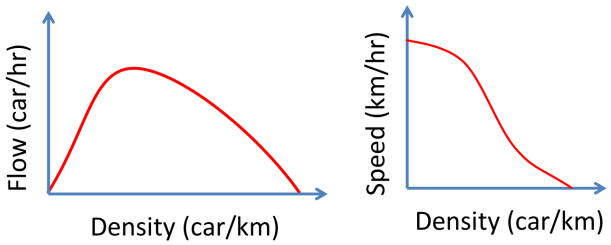
2.4.3 Kankergroeimodellen en andere

Bij een kankergroeimodel hebben we drie staten: “gezonde cel”, “kankercel”, of “dode cel” (een van de grote problemen met kanker/tumoren is dat het cellen doodmaakt). De kankercellen beginnen ergens geklonterd (zeg in de vorm van een cirkel) en verspreiden zich naar buiten, dode cellen achterlatend. Dit soort modellen zijn erg belangrijk om kanker goed te kunnen behandelen.

Er zijn veel andere voorbeelden te bedenken zoals verspreiding van malaria - dit is dan niet meer volledig CA-based omdat er ook *agents* in het spel zijn, en idealiter ook stochastische processen.

2.5 1D-CA-Verkeersmodel

In het 1D-model stellen we ons een rechte weg voor, waarbij voertuigen simpelweg vooruit willen (wat we dan voorstellen als naar rechts of links bewegen). In het simpelste model, ervan uitgaand dat auto’s naa rrechts willen, kan een auto vooruit als het vakje ervoor leeg is, dus bijvoorbeeld $010 \rightarrow 0$ (want



Figuur 3: *The fundamental diagram of traffic flow.*

de auto kan vooruit), maar 011 $\rightarrow$ 1 (want de auto blijft stilstaan). De regel die dit doet is regel 184 (als de auto's naar rechts bewegen). Deze simpele regels kunnen al complex gedrag genereren.

Zolang de dichtheid van auto's minder dan 50% is, zal de situatie zich stabiliseren (alle auto's "vinden hun plek op de weg"), zodra het meer wordt zal er file ontstaan en *stop-and-go*-verkeer.

Het belangrijke principe bij verkeer, dat idealiter terugkeert in de modellen, is *the fundamental diagram of traffic flow* (zie figuur 3), die stelt dat hoe dichter de auto's op elkaar zitten, hoe langzamer ze gaan, en dat de *flow* of *throughput* een piek kent.