Stochastisch Dynamisch Programmeren

Ger Koole
Vrije Universiteit, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

17 juni 2005

Samenvatting

We introduceren de principes van stochastisch dynamisch programmeren en passen het toe op de dobbelsteenduels van het spel Risk. Dit leidt tot een verbluffend eenvoudige optimale politiek.

Inleiding

Veel problemen in ons leven van alledag zijn zowel *stochastisch* als *dynamisch* van aard. Het eerste refereert aan het feit dat we de loop van veel zaken niet kunnen voorspellen. Wiskundig modelleren we deze onzekerheid met kansrekening, ofwel stochastiek. Het tweede heeft betrekking op het feit dat veel beslissingen niet eenmalig zijn, maar dat er meerdere beslissingsmomenten zijn. Een voorbeeld is de aanschaf van een hypotheek. Welke hypotheek het beste bij ons past hangt niet alleen van de huidige, maar ook van toekomstige financiële ontwikkelingen af. Zo wordt de mogelijkheid vervroegd af te lossen interessant bij een salarisverhoging of een erfenis, zeker als de dan geldende rentevoet laag is. Een lage rentevoet kan ook een reden zijn een nieuwe hypotheek met lagere maandelijkse lasten te nemen. De kunst is het een hypotheek te vinden met de juiste balans tussen enerzijds lage maandelijkse lasten en anderzijds de juiste hoeveelheid flexibiliteit, aangaande bijvoorbeeld aflossingen, zodat ook in de toekomst de kosten laag blijven. Het op deze wijze kiezen en onderhouden van een hypotheek is een typisch stochastisch dynamisch programmeringsprobleem. Het dynamische vinden we terug in de zeg jaarlijkse beslissing om vervroegd af te lossen of een andere hypotheek aan te schaffen, het stochastische komt tot uiting in de veranderende economische en persoonlijke omstandigheden. Het woord *programmering* is om historische redenen in de naam van dit soort problemen terug te vinden, net zoals in lineaire programmering. Een betere benaming zou zijn stochastisch dynamisch *optimaliserings*probleem.

Het modelleren

De theorie van de stochastische dynamische programmering levert ons methoden om optimale beslissingen te nemen in situaties met onzekerheid en herhaalde beslissingen. Maar daartoe moeten we eerst beslissen wat optimaal is. Om verschillende beslissingspolitieken te kunnen vergelijken willen we aan elke politiek een getal toekennen. De eerste stap is het toekennen van kosten en/of opbrengsten aan de verschillende beslissingen en situaties. (We gaan uit van de situatie met enkel kosten, opbrengsten kunnen worden gezien als negatieve kosten.) Daarna dienen we te beslissen hoe we de kosten op elke tijdstip kunnen vertalen naar één grootheid voor de hele beslissingsperiode. Een veel genomen keuze is het gemiddelde. Deze grootheid zal in het algemeen een stochastisch grootheid zijn, vanwege het onvoorspelbare karakter van het te modelleren probleem. Om tot een enkel getal te komen neemt men daarna in het algemeen de verwachting. Nu zijn we in staat politieken te vergelijken. In de volgende paragraaf formuleren we een en ander op wiskundig verantwoorde wijze. Maar laten we eerst nog eens naar onze hypotheek kijken.

Een moeilijke stap bij veel alledaagse problemen is het opstellen van een geschikt wiskundig model. Zo ook voor onze hypotheek. Hoe vertalen we het begrip "de meest geschikte hypotheek" in wiskundige termen? Laten we een poging wagen, zonder de ambitie alle aspecten te kunnen modelleren. We gaan uit van een maandelijks budget (dat variabel is), en een looptijd van n jaar. Onze definitie van "de meest geschikte hypotheek" wordt nu het zo aanwenden van het maandelijks budget voor het kopen en onderhouden van een hypotheek dat het vermogen na n jaar wordt gemaximaliseerd.

Het centrale begrip binnen de stochastische dynamische programmering is de *toestand*. De toestand is zodanig dat kennis van de huidige toestand alle mogelijke informatie geeft over het toekomstig gedrag van het systeem: alle informatie over het verleden is samengebracht in de toestand. Deze eigenschap wordt de *Markov eigenschap* genoemd, naar een beroemde Russische wiskundige (1856–1922). De Markov eigenschap zorgt ervoor dat beslissingen alleen van de huidige toestand af hoeven te hangen, en niet van de voorgaande, de geschiedenis. Dan moet de toestand natuurlijk wel waarneembaar zijn door de beslisser. Dit is de tweede belangrijke eigenschap waaraan de toestand moet voldoen. Een geschikte toestand voor het hypotheekprobleem bevat alle relevante informatie: het beschikbare budget, de huidige rente, het huidige hypotheekproduct, en het reeds opgebouwde vermogen.

De laatste stap in de specificatie van het probleem is het vaststellen van de wijze waarop het systeem van toestand verandert. Gezien het stochastische karakter van de toestandsovergangen, definiëren we kansen die de waarschijnlijkheden van de overgangen aangeven. In het hypotheekpobleem zijn de overgangskansen van toestand naar toestand samengesteld uit de overgangskansen voor de rente en het beschikbare budget. Samen met de te nemen beslissing vormen ze voor elke toestand een kansverdeling op de toestanden voor volgend jaar. Voor een eenvoudiger maar minder allerdaags probleem zullen we later/in het kader deze kansen specificeren.

Een optimalisatiealgoritme

Het is nu tijd om notatie in te voeren. Laat de ruimte van alle toestanden gegeven worden door X, en de ruimte van mogelijke akties of beslissingen door A. Als we ons bevinden in toestand $x \in X$, en we nemen aktie $a \in A$, dan is de kans dat er een overgang naar $y \in X$ plaatsvindt gelijk aan $p(x, a, y)$. Meestal kennen we aan een toestand-aktiecombinatie (x, a) directe kosten $c(x, a)$ toe. Met deze notatie kunnen we nu een algoritme gaan bestuderen dat in staat is de optimale politiek te vinden.

Een centraal concept in dit algoritme is de grootheid $V_t(x)$, met de volgende interpretatie: $V_t(x)$ zijn de verwachte minimale totale kosten als we nog t tijdseenheden te gaan hebben. Wat V_t zo interessant maakt is het feit dat V_{t+1} zich eenvouding uit laat drukken in V_t. Stel namelijk dat je in x, met nog $t+1$ beslissingen te gaan, aktie a kiest. Dat krijg je meteen kosten $c(x, a)$, en daarna ga je naar toestand y met kans $p(x, a, y)$. Vanuit y heb je dan nog t tijdseenheden te gaan, met kosten $V_t(y)$. De *verwachte* kosten vanuit x, bij aktie a, zijn dus

$$c(x, a) + \sum_{y \in X} p(x, a, y) V_t(y).$$

De verwachte *minimale* kosten zijn

$$V_{t+1}(x) = \min_{a \in A} \{c(x, a) + \sum_{y \in X} p(x, a, y) V_t(y)\}.$$

De kosten $V_0(x)$ worden voor elke x bekend verondersteld: er zijn per slot van rekening geen beslissingen meer te nemen. Voor de hypotheek zal dit -1 maal het eindbedrag zijn. Vanuit V_0 kan nu V_1 berekend worden, en V_2, enzovoorts. Voor de hypotheek geldt dat met x de huidige toestand, het getal $V_n(x)$ ons vertelt wat we na n jaar als vermogen kunnen verwachten. Het woord "verwachten" moet hier in de strikt wiskundige zin worden geïnterpreteerd: of het precies dit wordt of dat er een enorme variantie mogelijk is vertelt het algoritme ons niet! Dit principe van achterwaartse recursie staat onder vele namen bekend: dynamisch programmeren en waardeiteratie zijn de bekendste. Het stelt ons in staat voor veel problemen van bescheiden omvang de optimale strategie te berekenen. Wat wel en mogelijk is in de praktijk bestuderen we in de volgende paragraaf.

Toepassingen en de Curse of Dimensionality

Op het eerste gezicht lijkt dynamisch programmeren een wondermiddel voor veel praktische problemen. Het aantal in principe aan te pakken problemen is nog uit te breiden door niet-dynamische problemen met een dynamische component te herformuleren. Zo kan het vinden van het kortste pad van A naar B worden gezien als de *snelste* weg van A naar B: in alle toestanden onderweg betalen we kosten 1 omdat we nog niet bij de bestemming zijn, zodra we B bereiken betalen we niet meer.

De onvermijdelijke adder onder het gras is het volgende: de rekentijden die nodig zijn voor het doorrekenen van elke toestand op elke tijdseenheid kunnen enorm worden.

Dit is zeker het geval als de toestandsruimte uit meerdere componenten bestaat, d.w.z. meer-dimensionaal is. We zagen dit eigenlijk al bij het hypotheekprobleem: daar was de toestand vier-dimensionaal. Meer realistische problemen kunnen nog veel meer dimensies hebben. Proberen we een productieproces te modelleren, dan zal de toestand het aantal op behandeling wachtende tussenfabricaten van elke soort bij elke productiestap bevatten. Tientallen tot zelfs honderden dimensies kunnen het gevolg zijn! Eenzelfde hoge dimensionaliteit duikt op bij call centers, wanneer er sprake is van vele verschillende soorten calls en medewerkers met verschillende capaciteiten.

Deze hoge dimensionaliteit blijft niet zonder gevolgen. Voor elke t moeten we voor elke toestand x de waarde $V_t(x)$ berekenen. Stel nu dat er D dimensies zijn met elk N mogelijke toestanden. Het totaal aantal toestanden is dan N^D, en dit aantal wordt al bij relatief lage D zo groot dat het praktisch onmogelijk wordt de berekeningen uit te voeren. Dit verschijnsel staat bekend als de *Curse of Dimensionality*, een naam gegeven door een van de pioniers van de stochastische dynamische programmering, R. Bellman [1]. Sindsdien is er veel onderzoek gedaan op het gebied van dynamisch programmeren, maar het is pas de laatste jaren dat we een hernieuwde belangstelling zien in benaderingsalgoritmes die niet gehinderd worden door de Curse of Dimensionality. Op termijn zou dit ook tot nieuwe toepassingen van de theorie kunnen leiden.

Een optimale strategie voor Risk

We sluiten dit artikel af met een toepassing van dynamisch programmeren. Een eenvoudig te modelleren probleem met een elegante oplossing is het volgende.

Risk is een bordspel waarbij de wereld opgesplitst is in 42 *territoria*. De territoria zijn verdeeld over de spelers, die deze elk met een of meer legers bezet houden. Naburige territoria kunnen worden veroverd in zgn. *dobbelsteengevechten*. Hierbij gooit de aanvallende speler 3 dobbelstenen (3 legers op het aanvallende land voorstellend), waarop de verdediger kan kiezen met 1 of met 2 dobbelstenen te verdedigen. Afhankelijk van dit aantal verdwijnen 1 of 2 legers van het bord. Dit herhaalt zich totdat er op een van beide territoria (vrijwel) geen legers meer staan, of totdat de aanvaller besluit te stoppen met aanvallen.

Laten we het gooien eens wat nader bekijken. Nadat de aanvaller heeft gegooid, wordt de worp op aflopende volgorde gelegd, bijv. 5-3-2. Besluit de verdediger met 1 dobbelsteen te gooien, dan wordt deze met de hoogste van de aanvallende worp vergeleken. De aanvaller verliest een leger indien de verdediger hoger of even hoog gooit, in het voorbeeld bij een worp van 5 of 6. In de overige gevallen verliest de verdediger een leger.

Als de verdediger met 2 dobbelstenen verdedigt, wordt de hoogste worp met de hoogste van de aanvaller vergeleken, en de laagste worp van de verdediger met de middelste worp van de aanvaller. Ook hier geldt dat als de worpen even hoog zijn, de aanvaller het betreffende leger verliest. Gooit de verdediger 4-3 tegen 5-3-2, dan worden de 5 en de 4 en de beide 3-en met elkaar vergeleken, en beide spelers verliezen een leger. De vraag die we met dynamisch programmeren gaan beantwoorden is de volgende: Voor elke aanvalsworp, met hoeveel dobbelstenen moet de verdediger gooien?

Om te bepalen wat de beste speelwijze is moeten we eerst de te minimaliseren doelfunctie opstellen. We zijn geïnteresseerd in de situatie waarin beide partijen een grote voorraad legers hebben. Dit geeft ons tevens een eenvoudige vuistregel die ook bij kleinere aantallen legers te gebruiken is. Het doel bij grote aantallen legers is niet het voorkomen dat de aanvaller het aangevallen land verovert, maar het behouden van zoveel mogelijk legers terwijl de aanvaller er zoveel mogelijk verliest.

Belangrijk hierbij is dat een worp met 2 dobbelstenen 2 maal zo zwaar weegt als een worp met 1, want er gaan altijd evenveel legers van het bord als de verdediger dobbelstenen gooit. Een goede kandidaat voor doelfunctie is het gemiddeld verlies per door verdediger ingezet leger. Een speelwijze die dit minimaliseert, maximaliseert tegelijkertijd het gemiddeld verlies van de aanvaller. Dus we nemen de volgende doelfunctie: Minimaliseer het gemiddelde verlies per ingezet leger. Een tijdscomponent wordt geïntroduceerd door aan te nemen dat we per tijdseenheid één leger inzetten. Indien we dus met 2 legers verdedigen gaan we dus van zeg t naar $t-2$. Met 1 leger verdedigen reduceert de horizon als gebruikelijk met 1.

Een voorbeeld van een verkeerde doelfunctie is het gemiddelde verlies per worp. Inderdaad, dit verlies wordt geminimaliseerd door steeds met 1 dobbelsteen te gooien. Dit is overduidelijk niet de beste speelwijze; vooral tegen slechte aanvalsworpen zullen we met 2 dobbelstenen willen verdedigen om het verlies van de aanvaller zo groot mogelijk te doen zijn.

Voor de verder analyse hebben we wat notatie nodig. Laat $u(x,y)$ de kans op een aanvalsworp van x-y zijn, met $x \geq y$. Dus in $u(5,3)$ tellen we de kansen op worpen als 5-3-3, 2-5-3 en 3-1-5 bij elkaar. Evenzo, schrijf $q^2(r,s)$ voor de kans op een verdedigingsworp van r-s, met $r \geq s$. Voor het gemak noteren we ook $q^1(r)$ voor de worp met 1 dobbelsteen. Verder noteren we met $k((x,y),(r,s))$ $(k((x,y),r))$ het gemiddelde verlies van de verdediger bij een aanvalsworp van x-y en een verdediging met r-s. Zo is bijvoorbeeld $k((5,3),(4,3)) = 1$ en $k((3,1),(6,2)) = 0$. Al deze getallen kunnen eenvoudig worden uitgerekend en in tabellen worden opgeslagen.

Als toestand kiezen we de mogelijke worpen van de aanvaller, als aktie het aantal door de verdediger geworpen dobbelstenen, en als kosten het verwachte aantal verloren legers. De enige afwijking van het besproken algoritme is dat we bij aktie 2 in een keer 2 tijdseenheden verspringen. De door het algoritme gebruikte grootheden zijn als volgt:

$$X = \{(x,y) | 1 \leq y \leq x \leq 6\}, \quad A = \{1,2\}, \quad p((x,y),a,(r,s)) = u(r,s),$$

$$c((x,y),1) = \sum_{1 \leq r < x} q^1(r), \quad c((x,y),2) = \sum_{1 \leq r \leq s \leq 6} q^2(r,s)\Big[\mathrm{I}\{r < x\} + \mathrm{I}\{s < y\}\Big].$$

De getallen $c((x,y),a)$ zijn het verwachte verlies aan legers bij aanvalsworp x-y en een verdedigingsworp met a dobbelstenen. De notatie $\mathrm{I}\{\cdot\}$ is hierbij gebruikt voor de indicatorfunctie: $\mathrm{I}\{B\} = 1$ als B waar is, 0 anders.

De recursie voor het verwachte verlies aan legers wordt nu:

$$V_{t+1}(x,y) = \min\Big\{c((x,y),1) + \sum_{(r,s) \in X} p((x,y),1,(r,s)) V_t(r,s),$$

$$c((x,y),2) + \sum_{(r,s)\in X} p((x,y),2,(r,s))V_{t-1}(r,s)\Big\},$$

voor alle $t > 0$. Gegeven V_0 en V_1 kunnen we nu een computerprogramma schrijven dat V_n voor $n = 0, 1, 2, \ldots$ berekend. We kiezen $V_0(x,y) = 0$ en $V_1(x,y) = c((x,y),1)$.

De resultaten van de berekeningen zijn als volgt samen te vatten: het is optimaal met 2 dobbelstenen te verdedigen alleen wanneer de middelste worp 1, 2 of 3 is. Dus tegen 6-4 moet met 1 steen gegooid worden, tegen 6-3 met 2. De theorie vertelt ons ook dat V_n/n voor grote n tot het gemiddeld verlies nadert. Dit blijkt na afronding gelijk aan 0.499743 te zijn. Dus bij optimaal verdedigen is de verdediger iets in het voordeel, want het gemiddeld verlies van de aanvaller is 0.500257, iets groter dan 0.499743.

Achteraf vragen we ons natuurlijk af of het anders en misschien eenvoudiger had gekund. Kunnen we bijvoorbeeld elke worp niet apart bekijken, zoals Wierda & Helmich [3] doen? Een eenvoudige analyse laat zien dat dit niet verstandig is.

Laten we het verwachte verlies berekenen, bijvoorbeeld voor de aanvalsworp 5-3. Verdedig je nu met 1 dobbelsteen, dan verlies je in 4 van de mogelijke 6 worpen een leger, dus is het gemiddeld verlies per ingezet leger $\frac{2}{3}$.

Gooi je met 2 dobbelstenen, dan zijn er 36 mogelijkheden, die elk met kans $\frac{1}{36}$ voorkomen. Merk op dat de meeste worpen dubbel voorkomen. Het is eenvoudig na te rekenen dat de kans om 0, 1 of 2 legers te verliezen elk precies $\frac{1}{3}$ is. Dus het gemiddeld verlies over de worp is $\frac{1}{3}(0+1+2) = 1$, per ingezet leger precies $\frac{1}{2}$! Het spreekt voor zich dat we tegen 5-3 met 2 dobbelstenen verdedigen. Echter, laten we dezelfde som eens herhalen voor 6-5.

Bij een worp met 1 dobbelsteen is het gemiddeld verlies $\frac{5}{6}$. Met 2 dobbelstenen tegen 6-5 is het verlies echter $\frac{1}{12}0 + \frac{1}{4}1 + \frac{2}{3}2 = \frac{19}{12}$. Per ingezet leger dus $\frac{19}{24}$, minder dan $\frac{20}{24} = \frac{5}{6}$. Moeten we nu concluderen dat het tegen 6-5 beter is om met 2 dobbelstenen te verdedigen?

De denkfout die we hier maken is dat een aanvaller niet steeds 5-3 of 6-5 gooit. Het is dus verstandig tegen 6-5 met 1 dobbelsteen te verdedigen, om het andere leger te gebruiken voor een latere, gunstigere worp. We kunnen de aanvalsworpen niet apart bestuderen; de winstkansen bij de ene worp hebben invloed op de beslissingen bij andere worpen. Vandaar de noodzaak om met behulp van een methode als stochastisch dynamisch programmeren ook toekomstige worpen in de huidige beslissing te betrekken.

Dankwoord

Mijn dank gaat uit naar Richard Boucherie, die door consequent doorvragen mij motiveerde een stelling bij mijn proefschrift uit te werken tot [2]. De sectie over de dobbelsteengevechten bij Risk is daarop gebaseerd.

Referenties

[1] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[2] G.M. Koole. An optimal dice rolling policy for Risk. *Nieuw Archief voor Wiskunde*, 12:49–52, 1994.

[3] S.J. Wierda and J.P. Helmich. De aanvaller en de verdediger in het spel RISK. *Nieuwe Wiskrant*, 12(1):18–24, 1992.