KU LEUVEN

FACULTEIT WETENSCHAPPEN

Programmeren met Onzekerheid

Een Case Study

Sus VERWIMP

Promotor: Prof. dr. ir. T. Schrijvers
Affiliatie *(facultatief)*

Begeleider: *A. Vandenbroucke (facultatief)*
Affiliatie *(facultatief)*

Proefschrift ingediend tot het behalen van de graad van Master of Science in Toegepaste Informatica

Academiejaar 2017-2018

Voorwoord

Korte Samenvatting

Lijst van afkortingen en lijst van symbolen

Inhoudsopgave

Hoofdstuk 1

Inleiding

De grote interesse in het redeneren met onzekerheid in Artificiële Intelligentie resulteert in de ontwikkeling van verschillende programmeertalen. Deze talen worden probabilistische programmeertalen of PPL (Probabilistic Programming Language) genoemd. Een PPL heeft in grote lijnen 2 hoofddoelen:

- Het modelleren van een wereld met onzekerheid.
- Het redeneren/infereren van vragen over deze wereld met behulp van dit model.

Zonder PPL is het moeilijk om systemen met probabilistisch gedrag te modelleren. Probabilistische problemen werden geprogrammeerd aan de hand van grafische modellen zoals onder andere Bayesiaanse netwerken. Het redeneren over deze modellen gebeurde met inferentiemethodes die speciaal werden opgesteld voor het gemodelleerde probleem. Dit zorgde ervoor dat code voor een model niet herbruikbaar was en elke inferentie methode opnieuw geïmplementeerd moest worden voor elk probleem. Met de komst van PPL's zijn deze problemen opgelost. Elke PPL heeft zijn eigen manier van implementatie en wordt vaak geïmplementeerd als extensie op een general-purpose programmeertaal. Dit zorgt ervoor dat een PPL expressiever wordt. Gebruikers van een PPL kunnen een probabilistisch probleem volledig specifieren in het model en de bestaande inferentie methodes gebruiken om te redeneren over dit probleem. De general-purpose programmeertaal zorgt voor het hergebruiken van bestaande libraries en modellen. Algemene inferentiealgoritmes worden eenmalig gemaakt voor deze PPL zodat deze werkt voor alle modellen. Veel van deze PPLs streven naar een balans tussen performantie en expressiviteit. Het is belangrijk voor een PPL om problemen te kunnen modelleren en tegelijkertijd op een aanvaardbare tijd te kunnen redeneren over vragen over dit model.

Op het moment van schrijven zijn er veel verschillende PPL's beschikbaar zoals ProbLog2, Anglican,...[1]. Omdat er zo veel PPL's beschikbaar zijn, is het niet altijd duidelijk wat de voordelen of nadelen ten opzichte van elkaar zijn. Verschillende van deze PPL's zijn in recente artikels vergeleken met elkaar op het vlak van eigenschappen en concepten van de taal [3]. Andere artikels vergelijken vorige iteraties van dezelfde taal of PPL's met hetzelfde programmeerparadigma (bvb. logische of functionele programmeren) [2]. Het probleem bij deze vergelijkingen en evaluaties is dat er nooit evaluaties gedaan worden

[1] `http://probabilistic-programming.org/wiki/Home` geeft een overzicht van de verschillende PPL's die momenteel beschikbaar zijn.

voor PPL's met een verschillend programmeerparadigma.

Deze thesis behandelt het volgende:

- De evaluatie van twee PPL's, namelijk ProbLog2 en Anglican, aan de hand van kwalitatieve en kwantitatieve criteria en
- hoe deze talen verhouden ten opzichte van elkaar aan de hand van de evaluatiecriteria.

Deze thesis evalueert en vergelijkt ProbLog2 en Anglican omdat deze verschillende programmeerparadigma hebben en een verschillende semantiek voor het modelleren en infereren van probabilistische problemen. De evaluatiecriteria voor het evalueren en vergelijken van de PPL's zijn de volgende:

- Performantie (Hoe snel verloopt de inferentie.)
- Expressiviteit (Welke probabilistische systemen kunnen er gesimuleerd worden.)
- Geheugengebruik (Hoeveel geheugen is er nodig voor de inferentie.)
- Uitbreidbaarheid (Hoe makkelijk is het om regels en kansdistributies te wijzigen.)
- Beschikbare tools (Welke tools en features stelt de PPL ter beschikking.)
- Moeilijkheidsgraad (De moeilijkheidsgraad van het leren van en het programmeren in een PPL.)

Omdat het niet triviaal is om programmeertalen te vergelijken die totaal anders geïmplementeerd zijn gebeurd de evaluatie aan de hand van een case study. De implementatie van deze case study in ProbLog2 en Anglican vormt het vertrekpunt voor een vergelijking van deze twee programmeertalen op basis van de hogervermelde criteria. Uiteindelijk zal deze thesis aantonen welke PPL het best presteert in welke criteria aan de hand van het opgegeven probleem.

Hoofdstuk 2

Achtergrond

Deze sectie geeft meer informatie over de nodige achtergrondinformatie om de rest van deze thesis te begrijpen.

2.1 Redeneren met Onzekerheid

Redeneren met onzekerheid is één van de invloedrijkste domeinen van Artificiële Intelligentie. De reden hiervoor is omdat het universum van nature veel onzekerheid bevat. Denk aan de volgende punten:

- Kennis (We kunnen niet alles over het toepassingsdomein weten.)
- Onvolledige modellen (verschijnselen die niet onder het model vallen)
- Sensoren (We kunnen de wereld enkel observeren met de tools die geen exacte resultaten geven.)

Er zijn 3 belangrijke kwesties in verband met het werken met onzekerheid:

- Voorstellen van onzekerheid
- Redeneren met onzekerheid
- Leren aan de hand van onzekerheid

Het voorstellen van onzekerheid gebeurt in een model van een probabilistisch systeem. Een probabilistisch systeem is een systeem met probabilistisch gedrag. Het gedrag van een systeem is een verzameling van acties dat het systeem kan uitvoeren. Een model simuleert het gedrag van een systeem. Een model kan dus elke mogelijke uitkomst van een actie simuleren (zie voorbeeld 2.1.1). Onzekerheidsproblemen zijn problemen waar er weinig of geen zekerheid is over de uitkomst van een actie.

Voorbeeld 2.1.1. Een voorbeeld van een probabilistisch systeem is het opgooien van een eerlijk muntstuk[1]. Hier zijn we zeker dat als we het muntstuk opgooien dat het 50% kans

[1]In het voorbeeld van een eerlijk muntstuk kennen we de kansdistributie van het opgooien van het muntstuk. Als deze kansdistributie niet gekend is moet de kansdistributie geleerd worden aan de hand van bewijsmateriaal van de actie. Dit noemt het leren van interpretaties en wordt niet verder gebruikt in deze thesis.

heeft om te landen op kop of munt, maar er is geen zekerheid wat het resultaat is tenzij we de actie uitvoeren en het resultaat zien. De actie is hier het opgooien van een munt. Deze actie heeft twee mogelijke resultaten, of twee mogelijke werelden: namelijk de wereld waar het muntstuk land op kop en de wereld waar het muntstuk land op munt. Bij een eerlijke munt is de kans dat de eerste wereld bestaat 50% en de kans dat de tweede wereld 50%. We weten pas wat het resultaat is als we de actie uitvoeren. Het resultaat van de actie verandert dan in een gegeven of bewijsmateriaal van het model.

Het redeneren over een probabilistisch systeem maakt gebruik van het model. Een model simuleert alle mogelijke werelden die kunnen bestaan in het probabilistisch systeem. Het redeneren over een probabilistisch systeem geeft de kansdistributie voor elk element in het domein van de vraag die gesteld wordt over het systeem. Voorbeeld 2.1.2 is een simpel voorbeeld voor het opgooien van een eerlijk muntstuk.

Voorbeeld 2.1.2. We hebben een eerlijk muntstuk. Wat is de kans dat we na drie keer opgooien drie keer kop verkrijgen? Hier weten we dat het muntstuk eerlijk is dus 50% kans heeft om op kop of munt te landen. De kans dat we 3 keer tossen en 3 keer munt verkrijgen is dus:

$$P(c1 = kop \wedge c2 = kop \wedge c3 = kop) = \frac{1}{2} * \frac{1}{2} * \frac{1}{2} = \frac{1}{8}$$

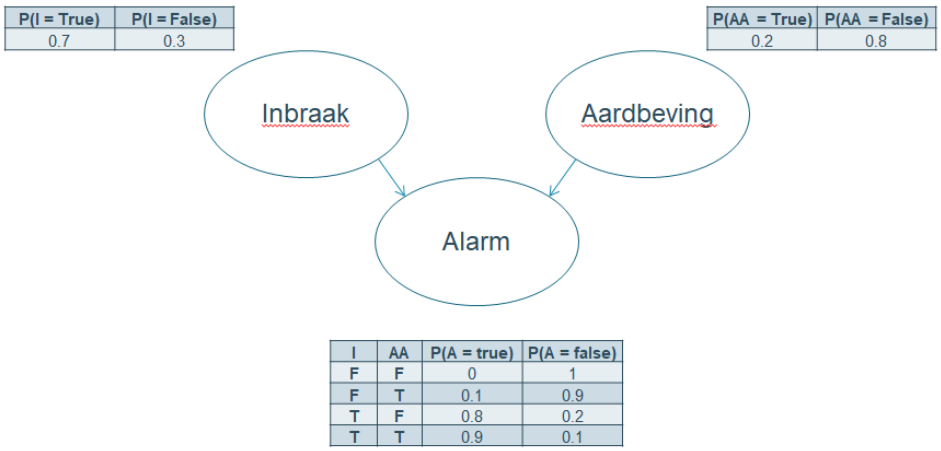
Het domein van de vraag is ofwel true of false. Het is ofwel drie keer kop of niet drie keer kop. Stel dat we het resultaat van de eerste opgooi weten en deze op kop landt. Omdat we dit weten kunnen we dit aan het model meegeven als bewijsmateriaal van het model. Dit geeft:

$$c1 = kop$$
$$P(c1 = kop \wedge c2 = kop \wedge c3 = kop \mid c1 = kop) = 1 * \frac{1}{2} * \frac{1}{2} = \frac{1}{4}$$

Voor kleine onzekerheidsproblemen als deze is dit nog makkelijk op te lossen met eenvoudige kans regels. Als het onzekerheidsprobleem groter is wordt het al moeilijker om dit te berekenen. In voorbeeld 2.1.3 stellen we een voorbeeld voor dat al moeilijker op te lossen is met simpele kans regels.

Voorbeeld 2.1.3. Stel dat we in een huis wonen in een regio waar er 20% kans is op een aardbeving en 70% kans is op een inbraak in het huis. We installeren een alarm in het huis. Op de verpakking van het alarm staan de kansen dat het alarm afgaat:

- 90% van de tijd gaat het alarm af als er een inbraak en een aardbeving is.
- 80% van de tijd gaat het alarm af als er een inbraak is.
- 10% van de tijd gaat het alarm af als er een aardbeving is.



Figuur 2.1: Bayesiaans netwerk van een onzekerheidsprobleem. Alarm is conditioneel afhankelijk van Inbraak en Aardbeving.

- Als er geen inbraak of aardbeving is gaat het alarm niet af.

Stel dat we het alarm horen afgaan. Wat is de kans dat er een aardbeving is als het alarm afgaat?

Elementen in het onzekerheidsprobleem die beïnvloed worden door kansen noemen we variabelen. In voorbeeld 2.1.3 zijn de variabelen:

- Aardbeving
- Inbraak
- Alarm

Variabelen Aardbeving en Inbraak hebben niets met elkaar te maken. Als we bewijsmateriaal hebben over de variabel aardbeving verandert de kansdistributie van inbraak niet, dit is analoog in de omgekeerde richting. Dit wilt zeggen dat deze variabelen conditioneel onafhankelijk zijn van elkaar. Bovendien, alarm is conditioneel afhankelijk van Inbraak en Aardbeving. Als we weten dat er een inbraak is kunnen we zeggen dat de kans dat het alarm afgaat vergroot. Hetzelfde gebeurd als er een aardbeving is.

Dit probleem kunnen we visueel illustreren met een Bayesiaans netwerk. Een Bayesiaans netwerk illustreert alle variabelen van een onzekerheidsprobleem in cirkels en de pijlen tussen de cirkels stellen de afhankelijkheid van een variabele ten opzichte van een andere variabele voor. Figuur 2.1 is een voorbeeld van een Bayesiaans netwerk voor voorbeeld 2.1.3.

Om het onzekerheidsprobleem uit voorbeeld 2.1.3 op te lossen maken we gebruik van de regel van Bayes.

$$P(Hypothese|Bewijs) = \frac{P(Bewijs|Hypothese)P(Hypothese)}{P(Bewijs)} \tag{2.1}$$

Als we de benamingen van figuur 2.1 gebruiken wordt dit:

$$P(AA = true|A = true) = \frac{P(A = true|AA = true)P(AA = true)}{P(A = true)} \tag{2.2}$$

De regel van Bayes geeft de kans dat een hypothese waar is in het model waar er al dan niet bewijsmateriaal is dat influentie heeft op de kans dat de hypothese waar is. Meer informatie over Bayesiaanse netwerken en het redeneren over deze netwerken in het boek "Bayesian Reasoning and Machine Learning" [1].

PPL's berekenen deze kansen aan de hand van het inferentieproces dat door de taal geïmplementeert wordt. Hoe ze dit doen verschilt voor elke PPL en wordt duidelijk uitgelegd in Secties 2.2.1 en 2.2.2. Het berekenen van de inferentie is een zeer krachtig, maar een zeer rekenintensief proces. Veel PPL's zoeken een balans tussen hoe efficiënt ze inferentie kunnen berekenen en welke problemen ze kunnen modelleren (hoe expressief de taal is).

2.2 Probabilistische Programmeertalen

Probabilistische programmeertalen zijn programmeertalen met twee hoofddoelen:

- Het modelleren van een wereld met onzekerheid.
- Het redeneren/infereren van vragen over deze wereld met behulp van dit model.

PPL's worden meestal geïmplementeerd als extensie op bestaande general-purpose programmeertalen. PPL's zoals Anglican en Church zijn gebaseerd op een dialect van LISP (een functionele programmeertaal). ProbLog2 en PRISM zijn gebasseerd op Prolog (een logische programmeertaal). Voor sommige talen zoals Stan is een speciale taal ontwikkeld die niet gebaseerd is op een bestaande programmeertaal. Deze thesis behandelt 2 PPL's, namelijk: ProbLog2 en Anglican. Secties 2.2.1 en 2.2.2 geven meer informatie over deze twee PPL's en hoe het modelleer- en inferentieproces werkt.

2.2.1 De Programmeertaal ProbLog2

ProbLog2 is een PPL gebaseerd op de programmeertaal Prolog. De volgende Secties 2.2.1 en 2.2.1 geven meer informatie over het modelleer- en inferentieproces.

Modelleren in ProbLog2

Een ProbLog2 programma bestaat uit 3 delen:

- een verzameling van gegronde probabilistische feiten,
- een logisch programma (verzameling van niet probabilistische regels),
- vragen en bewijsmateriaal over het model.

een gegrond probabilistisch feit ziet er uit als `p::f`. p is de kans dat het feit `f` waar is in het model. Het is ook mogelijk om gegronde probabilistische feiten volgens te schrijven als `p::f(X1,X2,...,Xn) :- body`. Het is belangrijk dat body het domein definieert van de variabelen X1, X2,..., Xn, zodat het deze probabilistische feiten geground kunnen worden. Meer informatie over het grounden van probabilistische feiten vindt u in Sectie 2.2.1.

Een logisch programma bestaan uit niet probabilistische regels. Deze regels kunnen gebruik maken van probabilistische feiten.

Vragen over het model komen in de vorm van `query(f)` en bewijsmateriaal van het model komt in de vorm van `evidence(f,Boolean)`. De expressie `query(f)` vraagt aan ProbLog wat de kans is dat de regel f waar is in het model. f is een query atoom van het model. `evidence(f,Boolean)` geeft bewijs dat regel f waar is of niet waar is in het model naar gelang de booleaanse waarde die meegegeven wordt.

Als we het onzekerheidsprobleem met het alarm en de aardbeving en de inbraak van voorbeeld 2.1.3 willen schrijven in ProbLog ziet het er als volgt uit:

```
0.7::burglary.
0.2::earthquake.
0.9::p_alarm1.
0.8::p_alarm2.
0.1::p_alarm3.

alarm :- burglary, earthquake, p_alarm1.
alarm :- burglary, \+earthquake, p_alarm2.
alarm :- \+burglary, earthquake, p_alarm3.

evidence(alarm,true).
query(earthquake).
```

In bovenstaande code is het feit `burglary`, `earthquake`, `p_alarm1`, `p_alarm2` en `p_alarm3` geannotteerd met een kans. Dit zijn gegronde probabilistische feiten. `0.7::burglary.` moet gelezen worden als: "er is 70% kans dat inbraak waar is in het model", Dit is analoog voor `earthquake`, `p_alarm1`,`p_alarm2` en `p_alarm3`. Het feit `alarm` is ook een gegrond probabilistisch feit maar deze is afgeleid van andere gegronde probabilistische feiten. Als de body van een afgeleid gegrond probabilistisch feit niet waar is, is het feit ook niet waar.

We geven aan het model mee dat het alarm is afgegaan aan de hand van bewijsmateriaal `evidence(alarm,true)`. Daarna vragen we aan het model wat de kans is dat er een aardbeving is `query(earthquake)`.

In de bovenstaande code maken we gebruik van gegronde probabilistische feiten zoals `p_alarm1`, `p_alarm2` en `p_alarm3` om de kansen te bepalen dat het alarm afgaat bij bepaalde situaties zoals: "Er is 90% kans dat het alarm afgaat als er een inbraak en een aardbeving is.". Aan de hand van geannoteerde disjunctie kunnen we dit op een overzichtelijke manier schrijven. De code word dan:

```
0.7::burglary.
```

```
0.2::earthquake.

0.9::alarm :- burglary, earthquake.
0.8::alarm :- burglary, \+earthquake.
0.1::alarm :- \+burglary, earthquake.

evidence(alarm,true).
query(earthquake).
```

Een geannoteerde disjunctie wordt ook gebruikt voor de kansverdeling te bepalen voor variabelen met een groter domein. In bovenstaande code is `earthquake` ofwel waar of niet waar in het model. Als we dit willen uitbreiden naar 1% kans op een zware aardbeving, 19% kans op een lichte aardbeving en 80% kans op geen aardbeving maken we gebruik van geannoteerde disjunctie. De code wordt dan:

```
0.7::burglary.
0.01::earthquake(heavy); 0.19::earthquake(mild); 0.8::earthquake(none).

0.90::alarm :-   burglary, earthquake(heavy).
0.85::alarm :-   burglary, earthquake(mild).
0.80::alarm :-   burglary, earthquake(none).
0.10::alarm :- \+burglary, earthquake(mild).
0.30::alarm :- \+burglary, earthquake(heavy).

evidence(alarm,true).
query(earthquake(_)).
```

De puntkomma tussen de feiten `earthquake` wilt zeggen dat er precies één bepaalde aardbeving (`earthquake(heavy)`, `earthquake(mild)` of `earthquake(none)`) waar is in het model.

Stel dat de staat van het alarm verslechtert met de jaren en de kans dat het alarm afgaat vermindert met het aantal jaar dat het oud is. Dit kunnen we oplossen aan de hand van flexibele kansen. Flexibele kansen worden gebruikt als de kans dat een gegrond probabilistisch feit afhangt van andere waarden. De volgende code geeft een voorbeeld van een alarm systeem dat verslechterd met de jaren:

```
0.7::burglary.
0.01::earthquake(heavy); 0.19::earthquake(mild); 0.8::earthquake(none).

P::alarm(Old) :- P is 0.90 - (Old / 100),   burglary, earthquake(heavy).
P::alarm(Old) :- P is 0.85 - (Old / 100),   burglary, earthquake(mild).
P::alarm(Old) :- P is 0.80 - (Old / 100),   burglary, earthquake(none).
P::alarm(Old) :- P is 0.10 - (Old / 100), \+burglary, earthquake(mild).
P::alarm(Old) :- P is 0.30 - (Old / 100), \+burglary, earthquake(heavy).

evidence(alarm(10),true).
query(earthquake(_)).
```

De combinatie van deze regels en het logisch programmeer systeem Prolog geeft ons de mogelijkheid om zeer uitgebreide onzekerheidsproblemen te modelleren. We kunnen vragen stellen aan deze modellen en deze vragen aan de hand van de inferentie machine van ProbLog2 oplossen. De volgende sectie 2.2.1 legt uit hoe deze vragen worden opgelost in ProbLog2.

Inferentie

De ProbLog2 inferentie machine kan verschillende inferentie taken oplossen, namelijk:

- Marginale kansverdeling (MARG)
- Meest waarschijnlijke verklaring (MPE, Most Probable Explenation)
- Leren van interpretaties

Deze thesis maakt enkel gebruik van het berekenen van de MARG.

ProbLog2 kan de exacte en benaderende MARG berekenen van alle query atomen van het model. De exacte inferentie berekent de exacte kans dat een query atoom waar is in het model rekening houdend met alle mogelijke werelden die kunnen bestaan in het model. Benaderende inferentie maakt gebruik van X aantal samples om de MARG van een query atoom te berekenen. X is hier een vrije keuze. Elke sample berekent de kans dat een query atoom waar is voor één mogelijke wereld van het model. Als we gebruik maken van een groot aantal samples convergeert de benaderende MARG naar de exacte MARG. Dit komt een groot aantal samples ervoor zorgt dat de kans groot is dat alle mogelijke werelden in het model gesampled worden.

Het inferentieproces voor het berekenen van de MARG in ProbLog2 maakt gebruik van een proces genaamd kennis compilatie (knowledge compilation). Het kennis compilatie proces volgt een aantal chronologische stappen. De volgende stappen zijn hetzelfde voor exacte en benaderende inferentie.

1. ProbLog2 genereert een lijst van de query atomen en bewijsmateriaal van het programma.
2. Aan de hand van de query atomen wordt het programma geconverteerd naar een ground programma.
3. Het ground programma wordt geconverteerd naar booleaans gewogen formules.

Door gebruik te maken van de booleaans gewogen formules kunnen we gebruik maken van wel gekende algoritmes. het berekenen van de exacte MARG gebeurd door de booleaanse gewogen formules te converteren naar een datastructuur zoals een “deterministic, decomposable negation normal form (d-DNNF)” of een “Binary decision diagram (BDD)”. De exacte inferentie wordt berekent door “Weighted Model Counting (WMC)” uit te voeren op de rekenkring. De rekenkring zorgt ervoor dat WMC met een aanvaardbare tijdcomplexiteit uitgevoerd kan worden.

Benaderende inferentie wordt berekent aan de hand van een sampling techniek genaamd

MC-SAT. Voor de geïnteresseerde is er het artikel [2]. Dit artikel gaat dieper in op de verschillende stappen voor het berekenen van exacte en benaderende inferentie in ProbLog2. Voor deze thesis is het genoeg om te weten welke stappen er worden uitgevoerd tijdens het inferentieproces. Sectie 4.1 vertelt meer over de performantie van de verschillende stappen in het inferentieproces.

Het ProbLog2 systeem kan gebruikt worden als een stand-alone tool of via Python. Via Python kan elke stap van het inferentieproces gemanipuleert worden. In sectie 4.1 staan verschillende voorbeelden waarom het gebruik maken van Python het inferentieproces kan versnellen voor de case study voorgesteld in sectie 3.1.1.

2.2.2 Anglican

Modelleren in Anglican

Inferentie

Hoofdstuk 3

Uitwerking van de case study

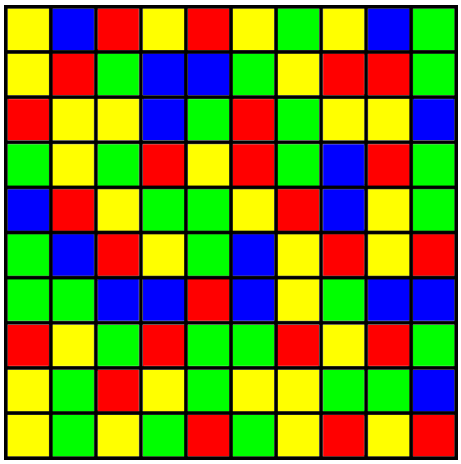
Omdat het niet triviaal is om PPL's te vergelijken en evalueren wordt er een case study gebruikt. Aan de hand van deze kwalitatieve en kwantitatieve criteria worden de PPL's geëvalueerd gebruik makend van de case study. Volgende Secties 3.1 en 3.2 geven meer informatie over het onzekerheidsprobleem van de case study en de evaluatiecriteria.

3.1 Onzekerheidsprobleem

Het onzekerheidsprobleem bestaat uit een speld dat dat beïnvloed word door probabilistische aspecten. Het spel wordt gespeeld aan de hand van vier verschillende strategieën die ieders ook beïnvloed worden door probabilistische aspecten.

3.1.1 Spel

Het spel bestaat uit een bord van 10 bij 10 blokken. Wanneer het spel gestart wordt krijgen de blokken een random kleur toegewezen maar er kunnen geen 3 van dezelfde blokken op een rij staan (verticaal en horizontaal). Er zijn 4 mogelijke kleuren: rood, groen, geel, blauw. In figuur 3.1 ziet u een voorbeeld van een initiëel bord.



Figuur 3.1: voorbeeld van een initiëel bord

Verandert in / Kleur blok	**Rood**	**Groen**	**Blauw**	**Geel**
Rood	0	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
Groen	$\frac{1}{3}$	0	$\frac{1}{3}$	$\frac{1}{3}$
Blauw	$\frac{1}{3}$	$\frac{1}{3}$	0	$\frac{1}{3}$
Geel	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	0

Tabel 3.1: Probabilistische distributie voor het veranderen van kleuren. Een rood blok heeft 1/3 kans dat het verandert in een groen blok, 1/3 kans dat het verandert in een blauw blok en 1/3 kans dat het verandert in een geel blok. Analoog voor een groen, blauw en geel blok.

De speler kan op elk van de blokken op het bord drukken. Als de speler op een blok drukt verandert dit van kleur. De kleur waar het blok in verandert hangt af van de probabilistische distributie. Om het simpel te houden wordt er voor een uniforme distributie gekozen. Tabel 3.1 geeft de kansdistributie voor elk blok weer.

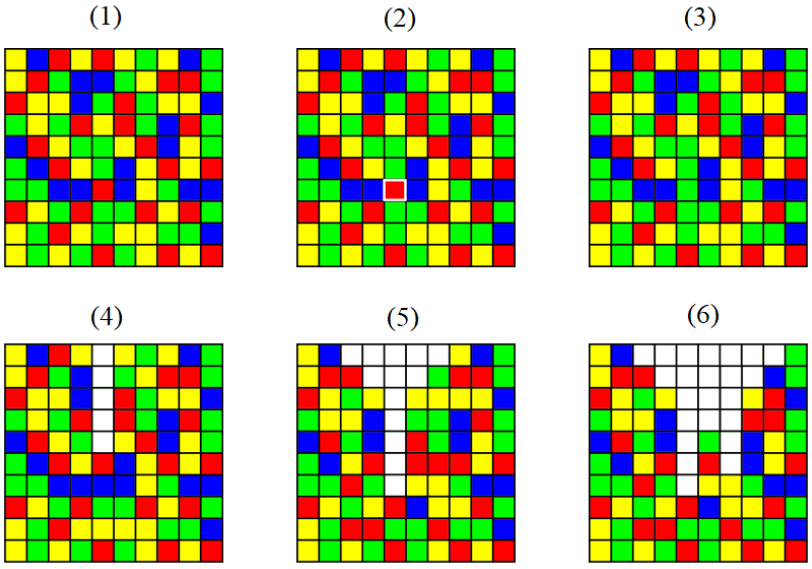
Als er drie of meer blokken van dezelfde kleur horizontaal of verticaal naast elkaar liggen verdwijnen ze en dit levert punten op. De blokken die zich boven de verdwenen blokken bevinden vallen naar beneden tot ze op een andere blok belanden ofwel op de bodem van het spelbord belanden. Voor elke blok die verwijderd wordt krijgt de speler 1 punt. De bedoeling van het spel is om in 5 beurten zoveel mogelijk punten te behalen waarin de speler in elke beurt 1 blok van kleur mag veranderen. De beurt eindigt wanneer er geen 3 blokken van dezelfde kleur meer op een rij staan. In figuur 3.2 ziet u het verloop van een beurt in een 10 bij 10 bord.

3.1.2 Strategieën

Ik heb 4 strategieën ontwikkeld om het spel te spelen.

- uniforme strategie,
- kleuren ratio strategie,
- mogelijke score strategie en
- gewogen score strategie.

Deze strategieën worden gebruikt om het spel te spelen op een bepaalde wijze. Elke strategie kiest altijd één blok uit de mogelijke blokken die beschikbaar zijn. Welk blok dit is bepaalt de strategie.



Figuur 3.2: Er wordt een blok gekozen om op te drukken, in dit geval een rood blok. Het blok verandert met een 1/3 kans in een groene blok. Omdat er meer dan 2 groene blokken op een rij staan worden deze verwijderd en de bovenstaande blokken vallen naar beneden. Dit wordt herhaald tot er niet meer dan 2 blokken van dezelfde kleur op een rij staan

Uniforme strategie

Als de uniforme strategie wordt toegepast is de kans dat een blok gekozen wordt uniform voor elk blok in het spelbord. Voor een 10 bij 10 bord is de kans dat een blok wordt gekozen dus $\frac{1}{10*10} = \frac{1}{100}$. Figuur 3.3 toont alle mogelijke keuzes die de uniforme strategie kan kiezen in het gegeven bord.

Kleuren ratio strategie

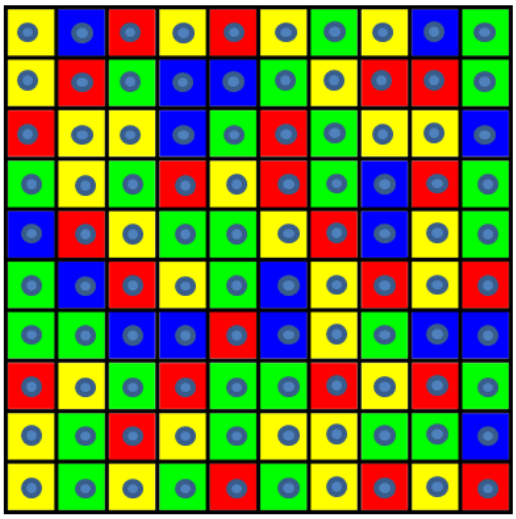
Voor de kleuren ratio strategie worden eerst alle blokken met dezelfde kleur opgeteld. Uit de kleur met het minst aantal blokken wordt uniform een blok gekozen. In figuur 3.4 zien we dat de blauwe blokken in de minderheid zijn. De strategie zorgt ervoor dat er uniform een blauw blok wordt gekozen.

Mogelijke score strategie

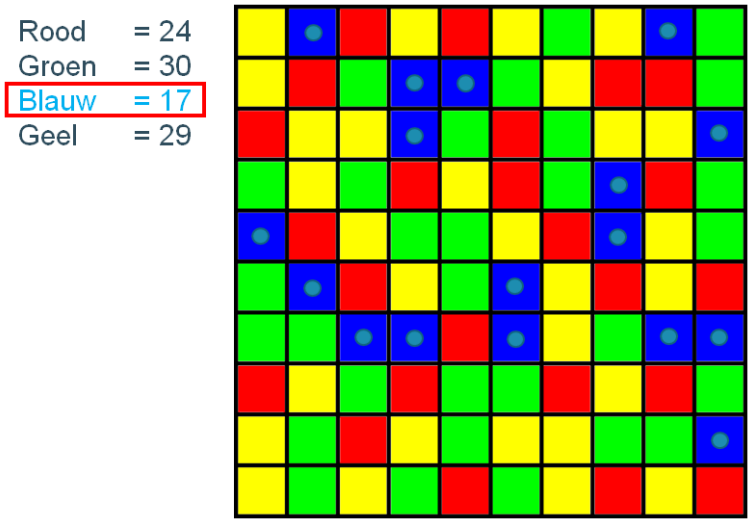
In de mogelijke score strategie wordt voor elk blok apart nagegaan of deze een mogelijke score kan hebben. Een blok heeft een mogelijke score als deze blok kan veranderen in een kleur die een score oplevert. Figuur 3.5 toont alle blokken die een mogelijke score kunnen opleveren. De mogelijke score strategie zorgt ervoor dat er uniform één blok gekozen wordt uit de lijst van blokken met een mogelijke score.

Gewogen score strategie

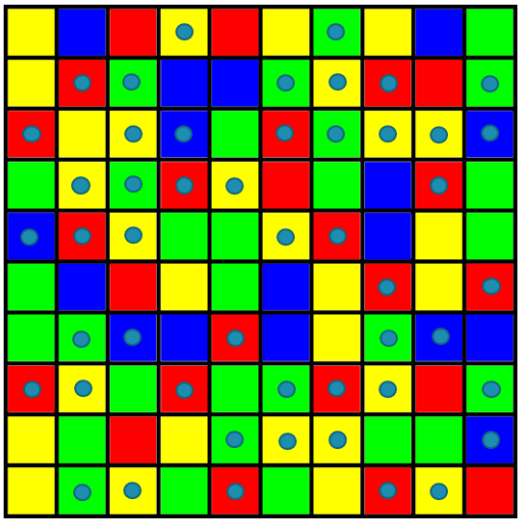
De gewogen score strategie is een uitbreiding op de mogelijke score strategie waar we niet enkel naar de mogelijke score zien, maar ook naar de gewogen score. Elk blok kan



Figuur 3.3: In de uniforme strategie kunnen alle blokken gekozen worden met een uniforme kans verdeling. De kans is $\frac{1}{100}$ voor elke blok in dit geval.



Figuur 3.4: De kleuren ratio voor de blauwe blokken is het minst. Hier wordt uniform een blauw blok gekozen met kans $\frac{1}{17}$ voor ieder blok.



Figuur 3.5: In een mogelijke score strategie wordt er uniform een blok gekozen uit alle blokken met een mogelijke score. In dit geval zijn er 48 blokken met een mogelijke score dus de kans is $\frac{1}{48}$ voor elke blok.

veranderen van kleur aan de hand van een kansverdeling zoals in tabel 3.1. De gewogen score strategie houdt rekening met deze kansverdeling. De gewogen score wordt berekend aan de hand van de score als een blok in Rood/Groen/Blauw/Geel verandert gewogen met de kans dat de blok verandert deze kleur.

3.2 Evaluatiecriteria

3.2.1 Performantie

3.2.2 Expressiviteit

3.2.3 Uitbreidbaarheid

3.2.4 Geheugengebruik

3.2.5 Beschikbare Tools

3.2.6 Moeilijkheidsgraad

Hoofdstuk 4

Evaluatie en Vergelijking van ProbLog2 en Anglican

4.1 ProbLog2

4.1.1 Performantie

4.1.2 Expressiviteit

4.1.3 Uitbreidbaarheid

4.1.4 Geheugengebruik

4.1.5 Beschikbare Tools

4.1.6 Moeilijkheidsgraad

4.2 Anglican

4.2.1 Performantie

4.2.2 Expressiviteit

4.2.3 Uitbreidbaarheid

4.2.4 Geheugengebruik

4.2.5 Beschikbare Tools

4.2.6 Moeilijkheidsgraad

4.3 ProbLog2 - Anglican

4.3.1 Performantie

4.3.2 Expressiviteit

4.3.3 Uitbreidbaarheid

4.3.4 Geheugengebruik

4.3.5 Beschikbare Tools

4.3.6 Moeilijkheidsgraad

Hoofdstuk 5

Conclusie

5.1 Toekomstig werk

Bibliografie

[1] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[2] D. Fierens, G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, and L. De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, 15(3):pp. 358 – 401, 2015.

[3] L. De Raedt and A. Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100(1):pp. 5 – 47, 2015.

Faculteit Computerwetenschappen
Geel Huis, Kasteelpark Arenberg 11 bus 2100
3001 LEUVEN, BELGIË
tel. + 32 16 32 14 01
www.kuleuven.be