KU LEUVEN

FACULTEIT WETENSCHAPPEN

Programmeren met Onzekerheid: Een Case Study

Ondertitel $S = \pi r^2$ *(facultatief)*

Sus VERWIMP

Promotor: Prof. 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

Voorwoord **i**

Korte Samenvatting **ii**

Lijst van afkortingen en lijst van symbolen **iii**

1 Inleiding **1**

2 Achtergrond **3**
2.1 Probabilistische programmeertalen 4
2.1.1 ProbLog2 5

3 Uitwerking **7**
3.1 Probleem verzinnen 7
3.1.1 Spel 7
3.1.2 Strategiën 8

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 hoofddooelen:

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

Voorheen was het zeer moeilijk als programmeur om werelden met onzekerheid te modelleren. Probabilistische problemen werden geprogrammeerd aan de hand van grafische modellen zoals bayesiaanse netwerken. Het redeneren over deze modellen gebeurde met inferentie methodes 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 PPLs 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 in deze PPL. 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.

Sinds de laatste jaren zijn er veel verschillende PPL's beschikbaar zoals ProbLog, Anglican,... De url `http://probabilistic-programming.org/wiki/Home` geeft een overzicht van de verschillende PPL's die momenteel beschikbaar zijn. Omdat er zo veel PPLs beschikbaar zijn, is het niet altijd duidelijk wat de voordelen of nadelen ten opzichte van elkaar zijn. Verschillende van deze PPLs 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 PPLs met hetzelfde programmeerparadigma (bvb. logische of functionele programmeren) [2]. Het probleem bij deze vergelijkingen en evaluaties is dat er nooit evaluaties gedaan worden voor PPL's met een verschillend programmeerparadigma.

In deze thesis ben ik van plan PPLs met verschillende programmeerparadigma zoals ProbLog en Anglican te vergelijken en evalueren. het is de bedoeling deze PPLs te evalueren ten opzichte van elkaar aan de hand van qualitatieve en quantitatieve criteria zoals: performantie, expressiviteit, geheugengebruik, uitbreidbaarheid, beschikbare tools, moeilijkheidsgraad,... Omdat het niet triviaal is om programmeertalen te vergelijken die totaal anders gemplementeerd zijn gebeurd de evaluatie aan de hand van een case study. De implementatie van deze case-study in ProbLog en Anglican vormt het vertrekpunt voor een vergelijking van deze twee programmeertalen op basis van de hogervernoemde criteria. Uiteindelijk zal deze thesis kunnen aantonen welke PPL het best presteert in welke criteria aan de hand van het opgegeven probleem.

Hoofdstuk 2

Achtergrond

In deze sectie geef ik de nodige achtergrondinformatie om de rest van mijn thesis te begrijpen.

Onzekerheid in Artificiële Intelligentie 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 van 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 onzekerheidsprobleem. Een model is een weergave van een onzekerheidsprobleem waarbij iedere mogelijke wereld kan gesimuleerd worden (zie voorbeeld 2.0.1). Onzekerheidsproblemen zijn problemen waar er weinig of geen zekerheid is over de uitkomst van een actie.

Voorbeeld 2.0.1. Een voorbeeld van een onzekerheidsprobleem met weinig zekerheid is het tossen van een eerlijk muntstuk. Het resultaat van deze tos kan kop of munt zijn, maar er is geen zekerheid wat het resultaat is. In dit onzekerheidsprobleem zijn er twee resultaten mogelijk, of twee mogelijke werelden, die bestaan als de tos actie wordt uitgevoerd: namelijk de wereld waar de munt land op kop en de wereld waar het muntstuk land op munt. Bij een eerlijke munt is de kans dat de wereld bestaat waar het muntstuk op kop land 50% en de kans dat de wereld bestaat waar het muntstuk op munt land 50%.

Voorbeeld 2.0.2. Een voorbeeld van een onzekerheidsprobleem met geen zekerheid is het tossen van een random muntstuk. Hier weten we niet of het een eerlijk muntstuk is of een muntstuk waar mee geknoeid is. In dit onzekerheidsprobleem zijn er nog steeds

twee mogelijke werelden die bestaan als de tos actie wordt uitgevoerd. Het verschil met het vorige voorbeeld is dat de munt nu een random munt is, dus de werelden die kunnen bestaan een andere kansverdeling kunnen hebben. Een onzekerheidsprobleem met geen zekerheid wordt zo gemodelleerd dat het de kansverdeling leert van interpretaties, in dit geval de resultaten van alle tossen. Dit komt niet voor in mijn thesis.

Het redeneren met dit onzekerheidsprobleem maakt gebruik van het model. Het model stelt de kansdistributie voor elke mogelijke wereld die kan bestaan in het model voor. Het redeneren over dit model stelt vragen over specifieke werelden.

Voorbeeld 2.0.3. Wat is de kans dat we een eerlijk muntstuk hebben als we deze 20 keer tossen en 15 keer kop en 5 keer munt verkrijgen?. In dit voorbeeld is de vraag: "Wat is de kans dat we een eerlijk muntstuk hebben?". Het bewijsmateriaal dat we hebben is: "20 keer tossen en 15 keer hoofd en 5 keer munt". Wat we willen is de kans dat de hypothese klopt, m.a.w. de kans dat het muntstuk eerlijk is.

Om te berekenen wat de kans is of een muntstuk eerlijk is, kunnen we gebruik maken van de regel van Bayes:

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

Voor het voorbeeld waar we de kans willen weten dat een muntstuk eerlijk is na 20 keer tossen en 15 keer kop en 5 keer munt te verkrijgen:

$$P(Eerlijk|15kop\&5munt) = \frac{P(15kop\&5munt|Eerlijk)P(Eerlijk)}{P(15kop\&5munt)} \tag{2.2}$$

De regel van Bayes geeft de kans dat een hypothese waar is in een wereld waar er al dan niet bewijsmateriaal is dat directe of indirecte influentie heeft op de kans van de hypothese. Voor meer informatie verwijs ik naar het boek [1]. PPL's berekenen hetzelfde aan de hand van het inferentieproces dat de taal implementeert. Hoe ze dit doen verschilt voor elke PPL. (!!!!!voorbeeld van inferentieproces)

Het berekenen van de inferentie is een zeer krachtig, maar een zeer rekenintensief proces. Veel PPL's zoeken een balans tussen hoe efficient ze inferentie kunnen berekenen en welke problemen ze kunnen modelleren (hoe expressief de taal is).

2.1 Probabilistische programmeertalen

Probabilistische programmeertalen (of PPL's van Probabilistic Programming Languages) zijn programmeertalen met 2 hoofdfuncties:

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

Er zijn PPL implementaties die een volledig nieuwe taal hebben gemplementeerd speciaal voor het modelleren en redeneren, maar de meesten zijn gemplementeerd als extensie op een bestaande general-purpose programmeertaal. In deze thesis maak ik gebruik van 2 PPLs, namelijk: ProbLog2 en Anglican.

2.1.1 ProbLog2

ProbLog2 is een PPL die gebaseerd is op de Satos distribution semantics (Sato 1995). Het kan beschouwd worden als een Prolog programma met probabilistische aspecten gemplementeerd. Een ProbLog programma bestaat uit feiten geannotteerd met kansen. 0.5 :: heads(C) :- coin(C). coin(c1).

In het bovenstaande voorbeeld hebben we 1 feit: coin(c1). en 1 probabilistisch predicaat: 0.5 :: heads(C) :- coin(C). wat dit programma zegt is dat het predicaat heads(C) waar is in de wereld met 50

We kunnen ook gebruik maken van annotated disjunction wat eigenlijk syntactische suiker is om hetzelfde te bereiken. Als we het vorige voorbeeld schrijven met annotated disjunction komen we tot: 0.5 :: heads(C, true); 0.5:: heads(C, false) :- coin(C). coin(c1).

In dit voorbeeld geeft het heads predicaat voor een bepaalde munt true voor 50

De combinatie van deze regels en het logic programmeer systeem prolog geeft ons de mogelijkheid om zeer uitgebreide probabilistische werelden te modelleren. We kunnen vragen stellen aan deze modellen en aan de hand van de inferentie machine van ProbLog worden deze vragen opgelost.

Inferentie

Om vragen te stellen over het model maken we gebruik van queries en bewijzen. Als we willen berekenen hoeveel kans we hebben dat we 2 munten tossen en ze beiden op hoofd landen kunnen we dit doen aan de hand van het volgende programma: 0.5 :: heads1. 0.5 :: heads2. two_heads :- heads1, heads2. query(two_heads).

Dit geeft het resultaat 0.25 wat uiteindelijk de kans is van ( ) ( ). We kunnen het resultaat van de query manipuleren door bewijs te geven aan dit model. Stel dat we weten dat de eerste munt zeker hoofd als resultaat heeft, dan kunnen we dit als bewijs meegeven aan het model op de volgende manier: 0.5 :: heads1. 0.5 :: heads2. two_heads :- heads1, heads2. Evidence(heads1, true). query(two_heads).

Dit geeft het resultaat 0.5. Omdat we weten dat de eerste munt zeker in hoofd resulteert is de uitkomst gewoon de kans dat de tweede munt op hoofd resulteert. Dit is voor de munt in het model 0.5. Voor meer voorbeelden verwijs ik naar de tutorials van de ProbLog website: https://dtai.cs.kuleuven.be/problog/tutorial.html

De inferentiemachine in ProbLog is gebasseerd op Knowledge Compilation. Dit houdt verschillende stappen in:

1. Het gronden van het programma. Dit wil zeggen alle variabelen in het programma vervangen door de termen die deze variabelen kunnen bevatten. Dit houdt enkel rekening met de queries dus predicaten in het systeem die niet aangesproken worden, worden ook niet gegrond.

2. Het gegronde programma converteren naar een equivallente booleaanse formula.

3. Het bewijs en gewogen functies gebruiken om de booleaanse formula te converteren naar een gewogen booleaanse formula.

Om de succes probabiliteit (SUCC), de conditionele probabiliteit (MARG) en de meest waarschijnlijke probabiliteit (MPE) te verkrijgen gebruiken we de gewogen booleaanse formula in combinatie met verschillende algoritmes zoals bijvoorbeeld Weighted Model Counting (WMC) om deze te berekenen. Meer informatie over het ProbLog Systeem en de inferentiemachine kunt u terugvinden in het artikel (Inference and Learning in Probabilistic Logic Programs using Weighted Boolean Formulas).

De API van het ProbLog systeem kan ook gebruikt worden via Python. Hierdoor kunnen we gebruik maken van Python om het inferentie process te manipuleren. Dit zorgt voor extra uitbreidbaarheid van verschillende problemen en dus extra expressiviteit van het systeem in totaal. De API is beschikbaar via de volgende URL: https://problog.readthedocs.io/en/latest/ap

Hoofdstuk 3

Uitwerking

3.1 Probleem verzinnen

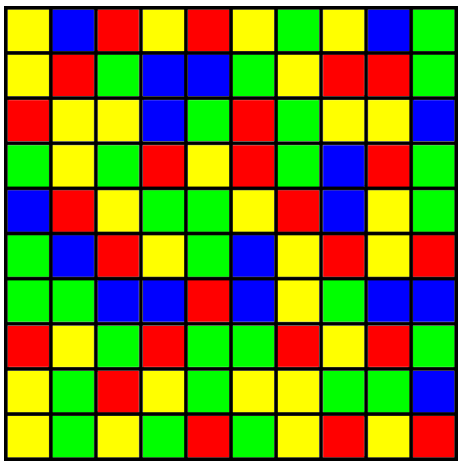
Als onzekerheidsprobleem heb ik gekozen om een spel te modelleren dat onderheven is aan probabilistische aspecten. Om het spel te spelen heb ik 4 spelstrategin ontwikkeld die elks ook onderheven zijn aan probabilistische aspecten.

3.1.1 Spel

Het spel bestaat uit een bord van 10 op 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 (enkel verticaal en horizontaal). Er zijn 4 kleuren in totaal: rood, groen, geel, blauw. In figuur 3.5 ziet u een voorbeeld van een initiëel bord.

De speler kan op elk van de blokken op het bord drukken. Als de speler op een blok drukt verandert deze van kleur. De kleur waar de blok in veranderd hangt af van de probabilistische distributie. Om het simpel te houden gebruik ik hier een uniforme distributie:

In woorden betekent dit dat als er op een rode blok wordt gedrukt er 1/3 kans is dat deze blok in een groene verandert, 1/3 kans in een blauwe verandert en 1/3 kans in een



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.

gele verandert. Voor een groene, blauwe en gele blok is dit analoog.

Als er drie of meer blokken van dezelfde kleur ofwel horizontaal naast elkaar liggen ofwel 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 ziet u het verloop van een beurt in een 10x10 bord.

3.1.2 Strategiën

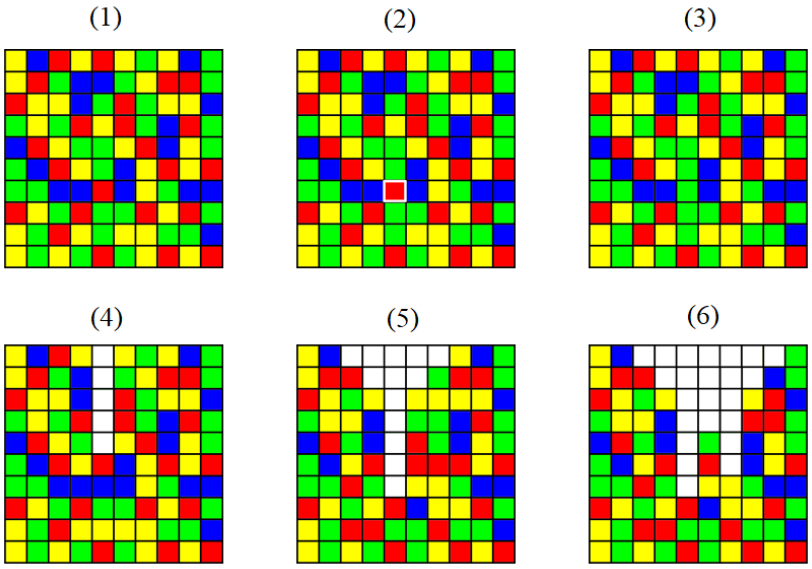
Ik heb 4 strategin ontwikkeld om het spel te spelen.

- Uniforme strategie
- Kleuren ratio strategie
- Mogelijke score strategie
- Gewogen score strategie

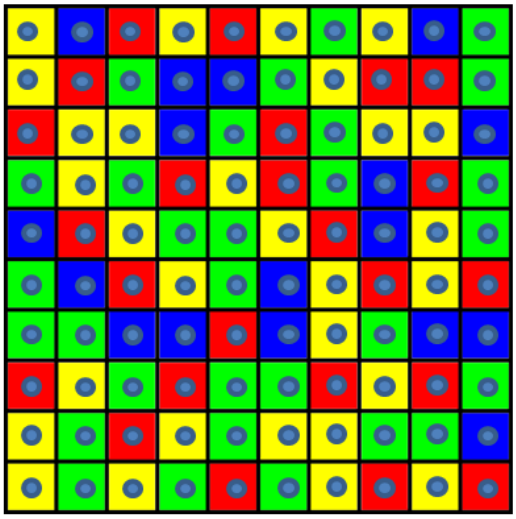
Deze strategin kunnen gebruikt worden om het spel te spelen op een bepaalde wijze. Elke strategie kiest altijd 1 blok uit de mogelijke blokken die beschikbaar zijn. Welk blok dit is hangt van de strategie af.

Uniforme strategie

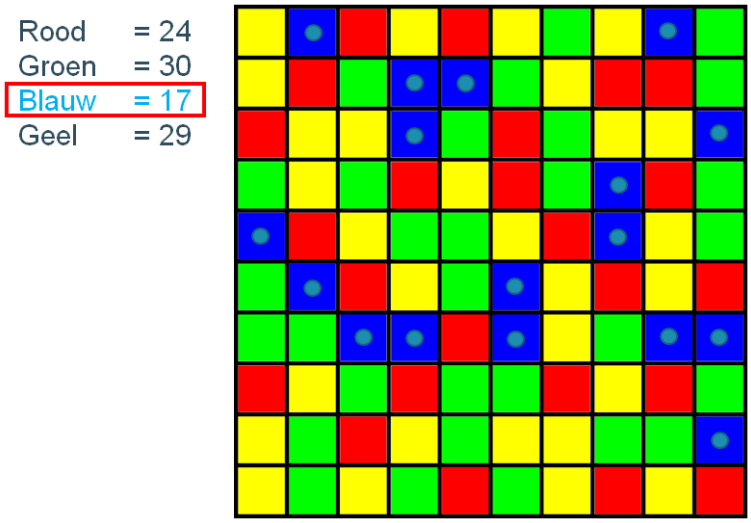
Als de uniforme strategie wordt toegepast is de kans dat een blok gekozen wordt uniform voor elk blok in het spelbord. Voor een 10x10 bord is de kans dat een blok wordt gekozen . In figuur 4 zien we alle mogelijke keuzes die de uniforme strategie kan kiezen in het gegeven bord.



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



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



Figuur 3.4: In bovenstaande figuur is de kleuren ratio voor de blauwe blokken het minst. Hier wordt uniform een blauwe blok gekozen met een 1/17 kans voor elke blok

Kleuren ratio strategie

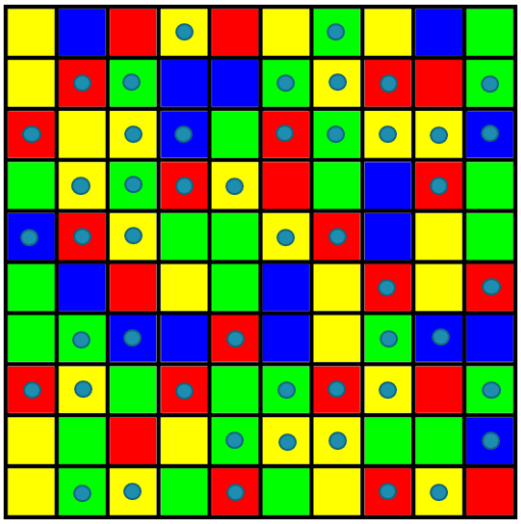
Voor de kleuren ratio strategie worden eerst alle blokken met dezelfde kleur opgeteld. Uit de kleur met het minst aantal blokken wordt uniform een blok gekozen. In figuur 5 zien we dat de blauwe blokken in de minderheid zijn. De strategie zorgt ervoor dat er uniform een blauwe blok wordt gekozen.

Mogelijke score strategie

In de mogelijke score strategie wordt voor elke blok apart nagegaan of deze een mogelijke score kan hebben. Een blok kan een mogelijke score hebben als deze blok kan veranderen in een kleur die een score oplevert. In figuur 6 ziet u alle blokken aangeduid die een mogelijke score kunnen opleveren. Er wordt 1 blok uit deze blokken gekozen met een uniforme kansverdeling.

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 naar de gewogen score. Elke blok kan veranderen van kleur aan de hand van een kansverdeling. 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 in Rood/Groen/Blauw/Geel.



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 kansverdeling is 1/48 voor elke blok

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