

Flash fill en Equation Discovery*

Jeroen Craps & Tom De Groote

KU Leuven

Leuven, België

jeroen.crap@student.kuleuven.be

tom.degroote@student.kuleuven.be

Abstract

Deze paper beschrijft een applicatie voor het vinden van een vergelijking in een set van relatief willekeurige getallen die door een gebruiker zijn gespecificeerd. Het doel van deze applicatie is om gebruikers te helpen in het vinden van een passende vergelijking voor de overige gegevens aan de hand van enkele voorbeelden. De applicatie voorziet dus de gebruiker om meerder voorbeelden te geven met welke de applicatie rekening moet houden. Op basis van enkele experimenten bleek deze applicatie in een beperkte tijd toch satisfieerbare resultaten te voorzien aan de gebruiker.

1 Introductie

Steeds vaker wordt er van een computer verwacht dat deze complexere taken kan uitvoeren. Terwijl het voor de gebruikers meestal duidelijk is wat er gewenst wordt, is dit voor de computer niet altijd het geval. Een recent voorbeeld hiervan is de 'FlashFill' functionaliteit dit momenteel aanwezig is in Excell 2013. Hierbij is het de bedoeling dat de computer het door de gebruiker gewenste patroon op een set van data herkent en kan toepassen op gelijkaardige sets van data om de gebruiker tijd te besparen. Een ontbrekende functionaliteit hierin is het herkennen van wiskundige bewerkingen of functies. Met andere woorden, het 'bepalen' van een passende vergelijking voor de gebruikers data. Verbeteren van het gebruikersgemak bij het bepalen van geschikte wiskundige vergelijkingen is het doel van dit werk.

Het doel van dit werk is om gelijkaardige functionaliteit aan die van Flash Fill te onderzoeken op het vlak van wiskundige vergelijkingen. Het onderzoek heeft zijn plaats in het domein van 'Artificiële Intelligentie', meer bepaald 'Machine Learning By Example'. Slechts een zeer beperkte groep van gebruikers hebben dit probleem. Vandaar dat er nog weinig tot geen onderzoek gedaan is met betrekking tot dit onderwerp. Bijgevolg is er geen dataset die kan gebruikt worden als invoer voor de experimenten van het onderzoek.

Toch is het bepalen van dit soort van vergelijking geen volledig onbekend onderzoeksdomein. In 2002 verscheen

er een paper die een oplossing voor het 'Countdown Problem' gaf. Bij het 'Countdown' is een Brits televisie programma waarbij de deelnemers een wiskunde vergelijking zoeken met de gegeven nummers om zo dicht mogelijk tot bij een bepaald streefdoel te komen. Een belangrijke beperking op dit probleem zijn wel dat elk gegeven cijfer maximaal éénmalig mag voorkomen. Indien een gebruiker bepaalde waarde meermaals wilt gebruiken is dit niet meer volgens de regels van 'Countdown'. Aangezien de gebruiker sommige waarden meermaals wil kunnen gebruiken moet er een alternatieve manier van aanpak gevolgd worden.

2 Probleemstelling

Gebruikers hebben reeds de mogelijkheid om patroonherkenning te gebruiken op hun dataset. Toch kan de gebruiker op zoek zijn naar een wiskundige relatie tussen in zijn dataset in plaats van een patroon.

2.1 Context-vrije grammatica

Om deze wiskundige relaties op te bouwen wordt er gebruikt gemaakt van een contextvrije grammatica. Volgens de 'formele taal theorie' is een contextvrije grammatica een formele grammatica waarbij elke productieregel er uit ziet als volgt: $V \rightarrow w$. Hier staat V voor één niet-terminaal symbool is en w een tekenreeks is van niet-terminale en terminale symbolen. 'Contextvrij' betekend dat de productieregels kunnen toegepast worden los van de context waarin het niet-terminale symbool zich bevindt.

$$\begin{aligned} E &\rightarrow E + E \\ E &\rightarrow E - E \\ E &\rightarrow E \times E \\ E &\rightarrow E \div E \\ E &\rightarrow 1, 2, \dots, 9, a, b, \dots \end{aligned}$$

Als een andere wiskundige operatie gewenst is moet deze toegevoegd worden als een productieregel aan bovenstaande grammatica. De gemakkelijke aanpasbaarheid is een voordeel van het werken met een context-vrije grammatica. Omdat de productieregels volledige onafhankelijk zijn van elkaar is het eenvoudig om de gewenste operaties te definiëren.

*Voor verandering vatbaar.

2.2 Boomstructuur

Door herhaaldelijke toepassing van de productieregels ontstaat er een boomstructuur. In deze boom bevinden zich alle mogelijke samenstellingen van productieregels die door de grammatica bepaald worden. Deze operatie zorgt voor een exponentiële groei r^d waarbij d staat voor de diepte van de boom en r voor het aantal mogelijke productieregels van de vorm $E \rightarrow E \text{ operand } E$. De groei van deze boom staat beschreven in volgende figuur en tabel.

[width=]treeSize.png

Aangezien de groei van de boom exponentieel is bekommt men veel grotere berekeningstijden volgens toename van de diepte in de boom. De grote overhead die het opstellen van de boom is, heeft tot gevolg dat het ten zeerste aangeraden om deze boom slechts éénmalig op voorhand uit te werken.

2.3 Evaluatiecriteria

'Snelheid, correctheid, gebruikersgemak en uitbreidbaarheid' zijn de vier evaluatie criteria waarop het onderzoek gaat beoordeeld worden. Het doel van het onderzoek is dus een efficiënt, correct en uitbreidbaarheid algoritme te vinden dat uit een gegeven set van voorbeelden een geschikte vergelijking kan vinden. Ook het gebruikersgemak is van belang bij het definiëren van wat de gebruiker juist van het algoritme verwacht.

3 Aanpak

Omdat de beperkingen van het 'Countdown' probleem niet van toepassing zijn, kunnen een groot deel van de gekende optimalisaties niet gebruikt worden. Er is met verschillende algoritmes gexperimenteerd. Elk algoritme zal besproken worden aan de hand van enkele stappen.

3.1 Brute Force

Het opstellen van de bewerkingsboom

Dit wordt bereikt door gebruik te maken van de context-vrije grammatica. Er wordt gestart met E vervolgens wordt de boom uitgebreid aan de hand van de regels van de CFG. De regel waarbij E echter waardes krijgt ($E \rightarrow 1, 2, \dots, 9, a, b, \dots$) wordt pas in de volgende stap gebruikt.

Het uitwerken van de vergelijkingen

In deze stap wordt de regel $E \rightarrow a, b, \dots$ gebruikt om elke vergelijking in de bewerkingsboom een waarde te geven. Enkel deze regel wordt gebruikt en elke mogelijkheid bij elke vergelijking wordt afgegaan.

Het resultaat

Zolang er vergelijkingen worden gezocht waarin geen constanten voorkomen vind dit algoritme in theorie altijd een oplossing. Het is echter wel zeer inefficiënt, zeker naarmate het aantal kolomwaarden toeneemt.

3.2 Toevoegen van gewichten

Ons doel is het vinden van een vergelijking die de gebruiker verlangt. Aangezien een vergelijking zoals $2 \times x + 1$ vaak voorkomt moeten er ook constanten toegelaten worden. Om dit te bereiken manier moet er slechts n wijziging plaatsvinden in het vorige algoritme. Namelijk bij het

uitwerken
 $1, 2, \dots, 9$

Het resultaat

Het gevoel is sterk toegenomen. De oplossing wordt door de gebruiker later uit de dataset als bekend.

3.3 Probleem

In de bevindingen is er redundantie die altijd $E \times E +$ worden v. redundantie

4 Dataset

Zoals reeds op het moment van geschikt is noodzakelijk aanpakken zelfgegeven datasets t. voor het a

4.1 Ver

Volledig

Elke waarde 0 en 100. op geen e bepaald a het feit da reeks van algoritme

Gestruct

De doelwillekeurige gegeven waren op er altijd e gelijkheid altijd een bepaald v menteel b

Complex

Hierbij gegenere binatie v dataset.

[width=]

4.2 Verantwoording keuze

Om de meeste geschikte datagenerator te bepalen is er een vergelijking nodig tussen de datasets die gecreëerd worden door de verschillende generatoren. Het criteria waaraan voldaan moet worden is dat de gegenereerde dataset zo dicht mogelijk bij reële gebruikersdata moet liggen. Er wordt dus vergeleken in hoeveel van de gevallen er op een brute-force manier een oplossing zou gevonden worden in de gegenereerde dataset. Hierbij bleek dat de 3^e *datagenerator* de meest geschikte was om gebruikersdata na te bootsen. Alle experimenten die van hieraf aan beschreven staan zijn gebeurd met datasets gegenereerd door deze datagenerator.

5 Experimenten

Alle experimenten worden enerzijds getest op oplossingsgraad als de volledige boom wordt doorlopen en anderzijds binnen een tijdsmarge van 2s. Aanhalen dat we naarmate de experimenten vorderen we ons baseren op voorgaande experimenten om keuzes te maken, anders zou het runnen van de experimenten enerzijds nutteloos zijn en anderzijds gigantisch veel tijd vragen (denk bv maar aan 10 gewichten enzo).

Vermelden welke processor enzo

5.1 Boom op voorhand opstellen

Enkel tijds winst aanhalen hier

5.2 Prunen op vergelijkingen

Boom wordt niet op voorhand opgesteld aangezien we dan meer kunnen prunen, met name K1-K1 enzo

5.3 Gewichten

Uitgevoerd op basis van een gepruned boom (experiment hierboven)

5.4 Prunen op vergelijkingen en gewichten

Gebaseerd op prime gewichten, boom wordt niet meer op voorhand berekend

6 Resultaten

Enkel bespreken van experimenten op basis van tijd en prime gewichten. Vergelijken van de brute force aanpak, de op voorhand berekende boom aanpak (met beperkte pruning mogelijkheden) en de uitiem gepruned boom

7 Andere aanpakken

Hier halen we paden aan die ons doorheen het jaar zijn tegengekomen. In onze ogen hebben ze geen meerwaarde en hier verantwoorden we waarom

7.1 Heuristiek

Resultaten kunnen enorm verschillen door \times , \div en \wedge .

7.2 Constraint programming

Onze kennis is niet goed genoeg om hier degelijke resultaten in te genereren

7.3 Op basis van onbekende

1 vb, je kan met 1 onbekende werken, enz. Niet nuttig aangezien dan altijd een oplossing gevonden wordt. Enerzijds veroorzaakt dit een enorme overhead waardoor de zoektijd enorm toeneemt. Anderzijds zullen de gevonden vergelijkingen gebaseerd zijn op constantes eerder dan op kolomwaarden.

8 Toekomstig werk

Door bepaalde omstandigheden was het onmogelijk om al het onderzoek/werk te doen dat er gedaan kan worden in verband met dit onderwerp. Een aantal zaken waarover reeds nagedacht was, maar wegens tijdgebrek uiteindelijk niet verder onderzocht zijn staan hieronder vermeld.

8.1 Toevoegen haakjes

Door het toevoegen van haakjes kunnen bepaalde vergelijkingen op een hoger niveau gevonden worden, zoals bv. $(A + B)^2$ word voorlopig pas gevonden als $A^2 + 2AB + B^2$. Het verschil van diepte in de boom tussen deze twee termen is 3, want $(A + B)^2$ wordt al reeds na 4 stappen gevonden terwijl $A^2 + 2AB + B^2$ pas op diepte 7 van de boom gevonden wordt. Nadelig aan het toevoegen van haakjes is wel dat er een extra productieregel aan de contextvrije grammatica zou toegevoegd worden (nl. $E \rightarrow (E)$). Een stijging van het aantal knopen op elk niveau is wel een gevolg van deze toevoeging.

Er zal dus een afweging gemaakt moeten worden tussen de kleinere diepte waarop er een geschikte vergelijking gevonden wordt en de breedte van elke diepte die toeneemt afhankelijk van het aantal productieregels die in de contextvrije grammatica aanwezig zijn.

8.2 Berekenen van 1 losstaand gewicht

Stel dat de laatste variabele berekend wordt uit de gegeven voorbeelden, dan kan er ook weer op reeds een veel lager niveau een geschikte vergelijking gevonden worden. Een gevolg hiervan is dan wel dat indien er meerdere voorbeelden zijn waarop het algoritme zich moet baseren er veel passende vergelijkingen voor het eerste voorbeeld bij de overige voorbeelden niet zal kloppen en daar dus een bepaalde overhead zal creëren. Het verwachte resultaat is hier dat deze maatregel voordelig is voor vergelijkingen waar er maar één voorbeeld is waarop er gebaseerd moet worden. Indien er meerdere voorbeelden zijn waar er op gebaseerd moet worden zal deze maatregel meer overhead veroorzaken dan effectief een verbetering zijn voor het algoritme.

9 Conclusie

Hier komt onze conclusie

10 Code

Hier moeten we vermelden dat er een GUI bestaat en dat de code via git beschikbaar is. Alles geschreven met java

Acknowledgments

Referenties

References

[Hutton, 2002] Graham Hutton. Functional pearl: The countdown problem. *Journal of Functional Programming*, 6(12):609–616, November 2002.