Flash fill en Equation Discovery*

Jeroen Craps & Tom De Groote
KU Leuven
Leuven, Belgie
jeroen.craps@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 gespecifieerd. 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

→ Hier moet situering van het werk komen + de bijdragen die het zal leveren aan de mensheid Doel van de paper Key results Waarom - Nog niemand heeft dit exact onderzoek gedaan - Aanhalen Countdown

2 Probleemstelling

Hier komen definities en evaluatiecriteria

2.1 Context-vrije grammatica

2.2 Boomstructuur

2.3 Evaluatiecriteria

3 Aanpak

3.1 Brute Force

Dit is de vertrek basis, we willen dit enorm verbeteren door het toepassen van optimalisaties

3.2 Toevoegen van gewichten

Verhoogt de zoekruimte → vertraagt maar hogere oplossingsgraad

3.3 Prunen op basis van vergelijkingen

Verkleint zoekruimte zonder de oplossingsgraad te verlagen

3.4 Prunen op basis van vergelijkingen en gewichten

Verkleint de zoekruimte enorm maar sommige oplossingen kunnen niet gevonden worden, toch de beste keuze zal blijen uit experimenten

4 Dataset

we beginnen met het vermelden dat er geen gebruikersdata is

4.1 Verschillende randomgeneratoren

We vermelden welke mogelijke randomgeneratoren we hebben

4.2 Verantwoording keuze

We vergelijken de randomgeneratoren met de brute force en kieze de generator die het dichtst bij gebruikersdata zou liggen. We zullen alle experimenten met deze generator uitvoeren

5 Experimenten

Alle experimenten worden enerzijds getest op oplossingsgraad als de volledige boom wordt doorlopen en anderzijds binnen een tijdmarge van 2s Aanhalen dat we naarmate de expemimenten 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 gewicten enzo)

Vermelden welke processor enzo

5.1 Boom op voorhand opstellen

Enkel tijdswinst 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 geprunde boom (experiment hierboven)

5.4 Prunen op vergelijkingen en gewichten

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

*Voor verandering vatbaar.

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 ultiem geprunede 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 $\hat{}$.

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 gebasseerd zijn op constantes eerder dan op kolomwaarden

8 Toekomstig werk

8.1 Toevoegen haakjes

8.2 Berkenen van 1 losstaand gewicht

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