

PE Taak: Performance onderzoek

Daan Bergmans

Sander Elsen

Arno Bruynseels

3AONC/D

2016-2017

# Sudokusolver-project op Github

Het project waar we voor gekozen hebben is een grafische sudokusolver die een sudoku kan oplossen en verbeteren.

Bron: <https://github.com/miga9/sudoku-solver-java>

# Wat is peformance analysis?

## Inleiding

Indien we opzoek gaan naar wat "performance analysis" betekent, waarvoor het gebruikt wordt en het nut ervan, stellen we vast dat dit concept gekend is en gebruikt wordt in talrijke domeinen:

- Software & Algoritmes - Human resource management

- Sport - Verkoop

- Economie - Marketing

- Verkeer - ...

In deze analyse gaan we ons echter focussen op het domein van software en algoritmes waaronder de programmeertaal java.

## Beschrijving

Performance analysis is eigenlijk een discipline waarbij systematische observaties worden gedaan om prestaties en besluitvorming te verbeteren. Hierbij wordt alle informatie hoofdzakelijk verstrekt door middel van objectieve statistische (Data Analysis) en visuele feedback (Video Analysis).

Het doel van performance analysis is dus registreren wat de huidige situatie is en hoe deze zo optimaal mogelijk kan verbetert worden.

Hiervoor onderschiedt het performance analysis proces 3 verschillende stappen:

### 1. Verzamelen van gegevens

Het verzamelen van gegevens is het proces waarbij data, over de prestaties van programma's, wordt verkregen uit een uitvoerend programma.

Om dit te kunnen realiseren worden er observatietools gebruikt die gegevens kunnen meten, registreren en visueel voorstellen.

Dit proces heeft een zeer uiteenlopende tijdsduur en is vrijwel afhankelijk van de software samen met welke doeleinden er geraliseerd moeten worden.

Doorgaans is dit een langdurig en intensief gebeuren waarbij de hoeveelheid data die verzamelt wordt zal verschillen per analsye.

### 2. Gegevens transformeren

Om data goed te kunnen bestuderen is het nodig dat de gegevens zodanig in een vorm worden omgezet die het mogelijk maken antwoorden te bieden. Nieuwe verzamelde gevegens voldoen zelden aan deze eisen en is het dus noodzakelijk om een transformatie te verwezenlijken.

Vaak wordt het volume van gegevens verminderd en gestructureerd tot een bestudeerbaar resultaat.

### 3. Gegevens visualiseren

Nadat de data een transformatie ondergaan heeft, zou het een enorme bijdrage zijn voor de analyse dat alle data visueel voorgesteld kan worden. Gegevens visualiseren kan vele vormen aannemen. Een van de meest voorkomende vormen zijn: grafieken, diagrammen en statistieken. Het is dus belangrijk dat er goed nagedacht wordt over welke visualisatietechnieken er gebruikt zullen worden in het analyse proces.

Visualisatie bevorderd dus het inzicht in de gegevens waardoor er sneller observaties kunnen worden gedaan en het nemen van beslissingen bekrachtigt.

## Waarom performance anylyse?

Het uitvoeren van een performence analyse brengt vele voordelen met zich mee. Zoals eerder vermeldt is het een grote aanwinst om een goed inzicht te krijgen in de performantie van de software. Hierdoor kunnen toekomstige prestaties geoptimaliseerd worden en wordt het nemen van beslissingen vergemakkelijkt.

# Tools voor performance analysis

In het software & algoritme domein wordt er voornamenlijk gebruikt gemaakt van observatietools die de prestaties van lopende software kunnen gaan registeren, meten en visualiseren.

Bij het selecteren van welke tool je gaat gebruiken is het belandrijk om volgende aspecten te beschouwen:

* **Accuraatheid**: De tijdspanne waarin data wordt verzamelt. Als dit realtime moet gebeuren is het belangrijk dat de tool dit ondersteunt. Indien er enkel periodiek data moet geregistreerd worden kan men naar een tool opzoek gaan die hier beter voor geschikt is.
* **Flexibiliteit**: Hoe belangrijk is het voor de analyse dat de tool uitgebreid kan worden om bijkomende informatie te verzamelen? Anderzijds kan het ook van beland zijn dat de toepassing verschillende visualisatie technieken bevatten.
* **Abstractie:** Welke data ga je analyseren met welke graad van accuraatheid.

Voor java zijn vlogende observatie tools beschikbaar:

* NetBeans Profiler
* JProfiler
* GC Viewer
* VisualVM
* Java Performance Analysis Tool
* Jrockit
* Eclipse Memory Analyzer
* Java Interactive Profiler
* Profiler4J
* JConsole

Voor de uitwerking van onze analysa hebben we gekozen voor (TOOL) omdat deze tool aan al onze belangen voldoet met bovenstaande aspecten in beraad genomen.

(UITLEG OVER TOOL)

WEBSITE VAN ALLE TOOLS: <https://blog.idrsolutions.com/2014/06/java-performance-tuning-tools/>

# Oplissongsmethode sudoku

Solve Button

1. **Check of de sudoku vol is**

* *Uitleg***:** Controleer of elke cel een waarde > 0 bevat.
* *Ja* : Toon bericht dat de sudoku vol is, stop algoritme
* *Neen*: Ga naar punt 2.

1. **Check of de sudoku geldig is om op te lossen**

* *Uitleg***:** Ga elke cel af en check eerst in de 3x3 box of er geen 2 zelfde getallen in voorkomen, daarna hetzelfde voor de rij en kolom.
* *Ja* : Toon bericht dat de sudoku niet geldig is, stop algoritme
* *Neen*: Maak de ingegeven getallen onbewerkbaar en maak ze grijs (Preset). Ga naar punt 3.

1. **Los de sudoku op**

*Uitleg***:** Het algoritme maakt begruikt van recursie (methode die zichzelf aanroept) en backtracking.

1. Check of elke cel is opgelost, het einde van de sudoku is bereikt

*Ja* : einde algoritme, sudoku is opgelsot

*Neen*: Stap 2

1. Check of de cell al een preset was of een waarde bevat

*Ja* : Ga naar voglende cell, door zichtzelf aan te roepen met de voglende celwaarde

*Neen*: Stap 3

1. Maak een lijst aan met cijfers aan van 1 t.e.m. 9 en shud deze door elkaar. (schudden om een meer willekeurige uitkomst te bekomen indien men gewoon op de solve button klikt zonder een preset van nummers. Niet noodzakelijk).
2. Ga elk nummer in deze lijst af en test of het gekozen nummer niet voorkomt en de 3x3box, rij en kollom.

*Lukt* : Vul dit nummer in de cel in en ga naar voglende cell door zichzelf aan te roepen met de volgende celwaarde.

*Faalt*: Stap 5

1. Als er geen waarde in de lijst met nummers van 1-9 gevonden wordt dat klopt is de sudoku wordt de huidige celwaarde op 0 gezet. Vanwege recursie wordt er dan aan backtracking gedaan, wat wil zegge dat er terug naar de vorige cel wordt gegaan en een andere waarde probeert te vinden die ook klopt. (Dit kan terug gaan tot de begincel en helemaal opnieuw beginnen)

**Algoritme verwijzing**

Wikipedia: <https://en.wikipedia.org/wiki/Sudoku_solving_algorithms>

Code My Road: <https://codemyroad.wordpress.com/2014/05/01/solving-sudoku-by-backtracking/>

Ugent: <http://lib.ugent.be/fulltxt/RUG01/002/224/798/RUG01-002224798_2015_0001_AC.pdf>

# Onderzoek

# Performantieverbeteringen (4)

## 1 - Shuffle methode

### Beschrijving

In de broncode wordt er een lijst van unieke getallen van 1 t.e.m. 9 gebruikt om de sudoku op te lossen. Hierbij gaat het algoritme voor elke cel telkens de lijst af totdat er een getal gevonden is dat klopt in de sudoku.

Hoewel deze lijst, in de broncode, willekeurig door elkaar wordt geschud voordat het algoritme deze gaat gebruiken, kan het juiste getal telkens zeer snel gevonden worden (steeds één van de eerste getallen in de lijst) of kan het net zeer lang duren (steeds één van de laatste getallen in de lijst).

Als dezelfde sudoku dus meermaal opgelost wordt met een lijst van getallen die steeds een willekeurige volgorde heeft, zal het algoritme altijd een verschillend aantal keer de lijst doorlopen.

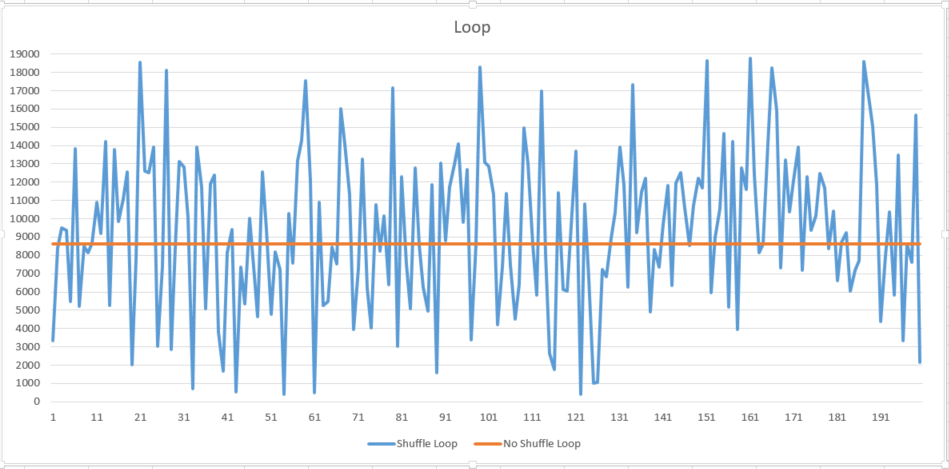
Omdat het algoritme functioneerd met de backtricking methode, zal eveneens het aantal keer dat er aan backtracking wordt gedaan verschillen.

Welk effect zou het hebben op het algoritme indien de lijst niet willekeurig door elkaar wordt geschud?

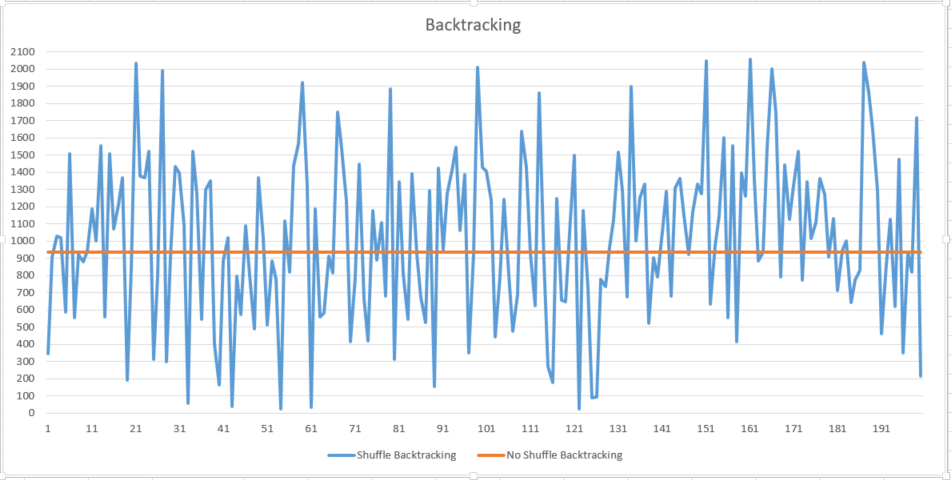
## Meting

Na een sudoko 200 maal op te lossen, met en zonder de shuffle methode. Hebben we het aantal keer dat de lijst doorlopen wordt gerigistreerd samen met het aantal maal dat er aan backtracking werd gedaan.

Loop



Backtracking

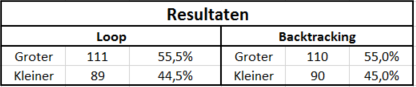


Deze resultaten hebben we in een lijngrafiek gegoten om beter inzicht te krijgen in de verzamelde resutlaten.

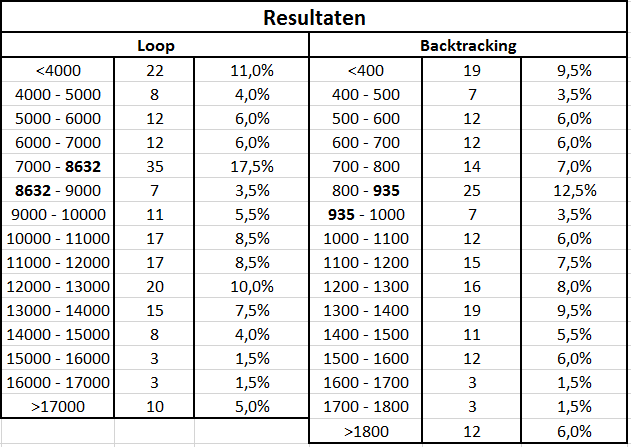
Zoals we merken is er een constante lijn bij beide grafieken indien we geen shuffle methode toepassen. Dit is te wijten aan de statische volgorde van de getallen in de lijst. Het algoritme zal steeds opnieuw op het zelfde ogenblik dezelfde getallen gaan gebruiken waardoor we altijd het zelfde resultaat bekomen.

Wanneer we kijken naar de data in de grafieken die afkomstig zijn met de shuffle methode nemen we meteen waar dat er zeer uiteenlopende resultaten zijn telekens de sudoku werd opgelost.

Als we gegevens nader gaan bekijken en gaan vergelijken met de waarde van de non-shuffle methode, bekomen we onderstaand resultaat.



We constateren dat in 5% van de 200 oplossingen de shuffle methode meer iteraties maakt zowel bij de loop als bij de backtracking.



Indien we dan naar absolute aantallen van iteraties gaan kijken, stellen we vast dat er bij de loop en backtracking het grootste aantal van iteraties, die een winst marge hebben, maar net onder de constante liggen. Hoewel dit een winst aan performantie blijkt te zijn is dit in tegen stelling met de iteraties die hoger liggen dan de constante. Het grooste aandeel hiervan ligt dan veel hoger boven de constante waarde.

### Vaststelling

We kunnen dus vaststellen dat er een deel van de tijd met de shuffle methode een winst aan performantie gegenereerd wordt. Echter wordt er bij 5% van alle oplossingen een groter aantal iteraties uitgevoerd. En wanneer er meer iteraties worden overlopen, ligt dit aantal gemiddeld hoger dan het aantal iteraties die er gewonnen worden bij een winst marge.

# Verslag bevindingen (analyse, testen, cijfergegevens,…)

# Voorstellen verbeteringen aan auteur

# Reflectie