

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 men 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 belang zijn dat de toepassing verschillende visualisatie technieken bevat.
* **Abstractie:** Welke data ga je analyseren met welke graad van nauwkeurigheid.

Voor java zijn volgende observatie tools beschikbaar:

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

Voor de uitwerking van onze analyse hebben we gekozen voor Java VisualVM omdat deze tool aan al onze belangen voldoet met bovenstaande aspecten in beraad genomen.

Java VisualVM

De Java VisualVM grafische user interface is een monitoring tool die voldoet aan de Java Management Extensions (JMX) specificatie. Java VisualVM maakt gebruik van het uitgebreide instrumentarium van de Java Virtual Machine (Java VM) om informatie te voorzien, over de prestaties en het gebruik van resources, van applicaties die draaien op het java-platform.

Bron: <https://blog.idrsolutions.com/2014/06/java-performance-tuning-tools/>

# Oplossingsmethode sudoku

De applicatie ondersteunt zowel het controleren van een ingegeven oplossing als het oplossen van een sudoku met een al dan niet ingegeven preset.

Een algoritme beschrijven in een tesktuele vorm kan nogal omslachtig en onduidelijk overkomen. Hierdoor hebben wij ervoor geopteerd het algoritme te beschrijven in een stappenplan waarbij elke stap besproken word dat het algoritme doorloopt.

## 1. Submit button - controleren van een ingegeven oplossing

1. Check de celwaarde

* *Uitleg***:** Controleer of de huidige cel een waarde > 0 bevat.
* *Ja* : Stap 2
* *Neen*: Einde algoritme, toon bericht dat suduko niet compleet is.

1. Check 3x3 box

* *Uitleg***:** Controleer dat de huidige celwaarde geen duplicaties heeft in de 3x3 box.
* *Ja* : Stap 3.
* *Neen*: Einde algoritme, toon bericht dat suduko niet correct is.

1. Check rij en kolom

* *Uitleg***:** Controleer dat de huidige celwaarde geen duplicaties heeft in de rij of kolom. De methode die dit controleerd sorteerd de rij en kolom op volgorde, waarna deze nagaat of de gesorteerde getallen tellen van 1 to 9.
* *Ja* : Waarde van huidige cel is correct.
* *Neen*: Einde algoritme, toon bericht dat suduko niet correct is.

1. Einde controle

* *Uitleg***:** Als elke celwaarde gecontroleerd is en de cel heeft steeds alle testen doorstaan, is de ingegeven oplossing correct.

## 2. Solve button – oplossen van een sudoku

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 opgelost is, 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. Dit is niet noodzakelijk, maar wel veel performanter zonder presets.
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 in 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 genavigeerd en hiervoor een andere waarde probeert te vinden die ook klopt. Waarna het algoritme opnieuw de sudoku zal proberen oplosson met de nieuwe waarde. Dit kan teruggaan tot de begincel.

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>

# Performantieverbeteringen

## 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.

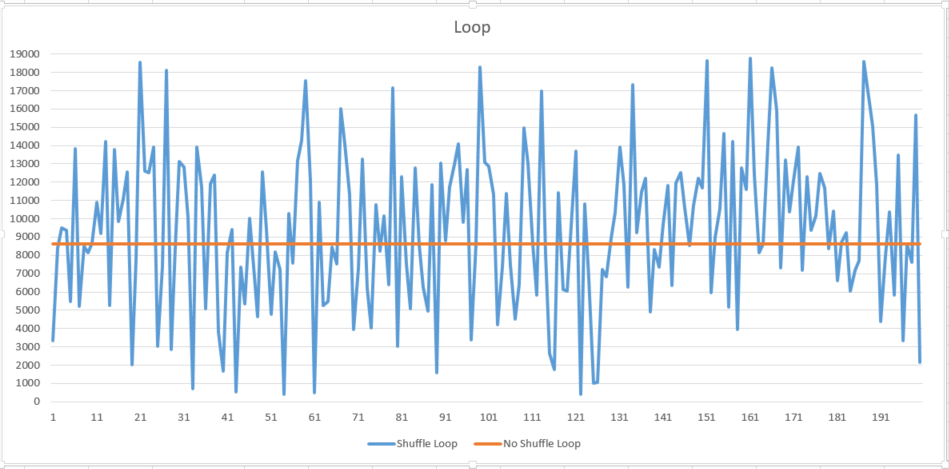
Omdat het algoritme functioneerd met de backtricking methode, zal eveneens het aantal keer dat er aan backtracking wordt gedaan verschillen.

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

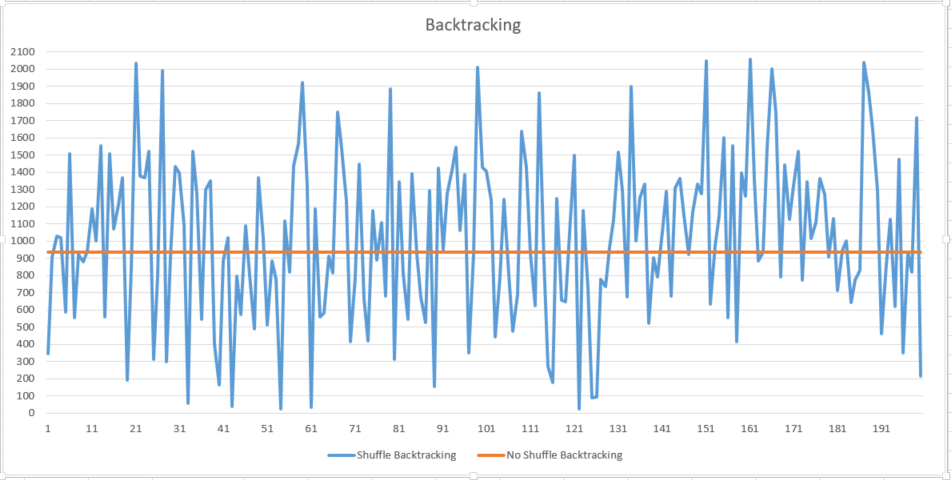
## Meting

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

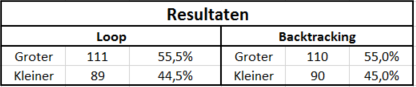


Deze resultaten hebben we in een lijngrafiek gegoten om beter inzicht te krijgen in de verzamelde resutlaten.

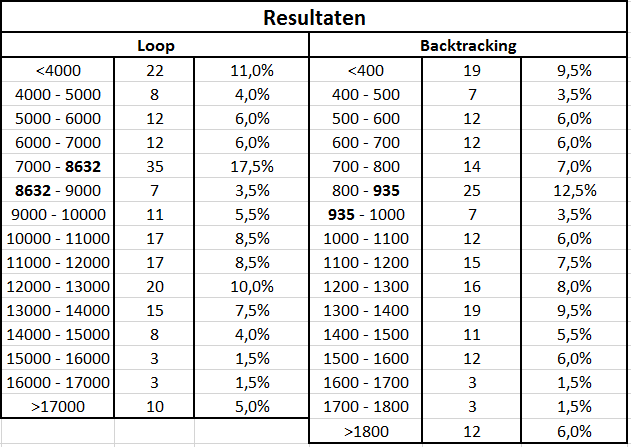
Zoals we merken is er een constante lijn bij beide grafieken indien we geen shuffle methode toepassen. Dit is te wijten aan de statische volgorde van de getallen in de lijst. Het algoritme zal steeds opnieuw op het zelfde ogenblik dezelfde getallen gaan gebruiken waardoor we altijd het zelfde resultaat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

Wanneer we kijken naar de data in de grafieken die gerelateerd zijn met de shuffle methode, nemen we meteen waar dat er zeer uiteenlopende resultaten zijn telkens 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 het grootste aantal iteraties, die een winst marge hebben, zowel bij de loop als bij backtracking 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 besluiten 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.

## 2 – Primitieve datatypes

### Beschrijving

De Java virtuele machine werkt met de heap en de stack. Dit zijn 2 geheugenopslagplaatsen die een verschillende werking hebben wat betreft geheugen allocatie en dataopslag.

De heap

Op een heap kunnen data-elementen (vaak objecten) worden opgeslagen en verwijderd. Dit verwijderen gebeurt door de garbage collector als het geheugen van de heap vol is of indien het data-element niet meer nodig is en afgebroken wordt.

Voor het alloceren van geheugenblokken maakt de heap gebruikt van dynamische allocatie. Dit wil zeggen dat een geheugenblok zowel gealloceerd en vrij gemaakt kan worden op elk gegeven moment.

Een applicatie bevat meestal 1 heap.

De stack

De stack is een gereserveerde geheugenruimte die verdeeld is in frames. Er wordt telkens een stackframe aangemaakt waneer er een methode wordt uitgevoerd. Elk stackframe bevat de referentie van lokale variabelen (primitieve datatypes) en andere relevante data. Een stackframe wordt terug vrij gemaakt als de methode is afgelopen.

De stack wordt altijd gereserveerd in een LIFO (last in first out) volgorde; de laatst gereserveerde blok zal altijd het volgende blok zijn dat vrij gemaakt wordt.

Elke thread in een applicatie krijgt een stack toegewezen.

Besluit

Door de verschillende werking van geheugen allocatie en dataopslag zal de stack dus sneller werken dan de heap.

Indien we bovenstaande aspecten in beraad gaan nemen, kunnen we hiermee rekening gaan houden in de java code. In java bestaan er wrapperklassen die afgeleid zijn van de primitieve datatypes. Deze wrapperklassen zijn echter objecten die op de heap worden opgeslagen. Dit in tegenstelling tot de primitieve datatypes dia worden bijgehouden op de stack wanneer deze geen referentie bevatten (bv. array).

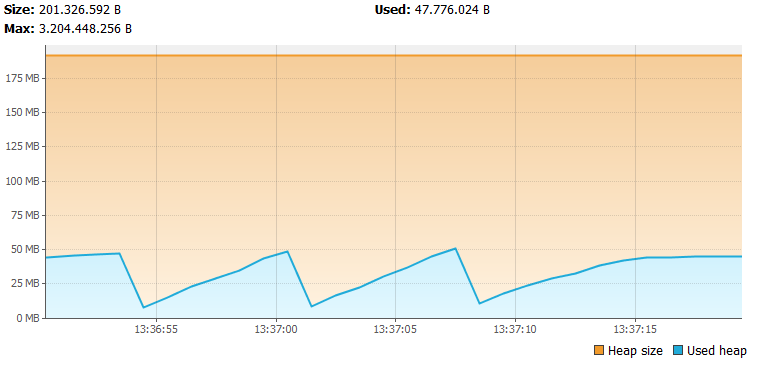
Het zou dus efficiënter zijn in termen van geheugengebruik om primitieve datatypes te gebruiken i.p.v. wrapperklassen.

### Meting

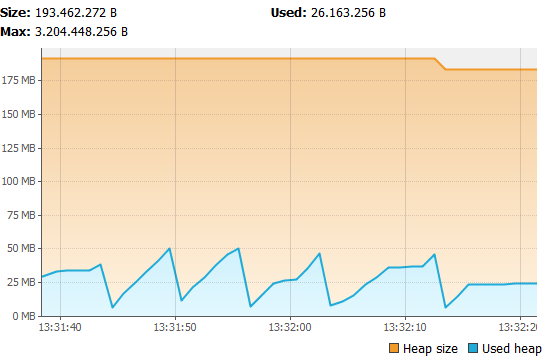
We hebben de sudoku 50 maal opgelost met zowel wrapperklassen als primitieve datatypes.

Bij het uitvoeren van deze metingen hebben we gebruik gemaakt van de Java VisualVM heap monitor om het geheugengebruik van de heap te visualiseren.

Wrapperklassen



Primitieve datatypes



Op het eerste zicht ziet de grafiek er hetzelfde uit bij beide resultaten. Samen hebben ze opwaarste bewegingen, wat het ophopen en creëren van object voorstelt, en dalingen die het opruimen van de niet meer overlevende objecten in kaart brengt.

Hoewel er dus grafisch niet meteen een verschil zichtbaar is, zit er een verschil in het totaal aantal geheugen dat de heap in gebruikt heeft op het einde van de meting. Als we kijken naar de ‘Used’ paramater die Java VisualVM heeft geregistreed, zien we dat de wrapperklassen 47 776 024 B geheugen in gebruikt heeft en de primitieve datatypes 26 163 256 B.

Dit is vooral omslachtig en zegt weinig over het eigenlijke verschil. Dus zetten we deze resultaten even om leesbare vorm waarbij de factor staat t.o.v. de grootte van de primitieve datatypes.



In bovenstaande afbeelding merken we meteen op dat de wrapperklassen effectief meer geheugen gebruikt hebben (47.7MB) dan de primitieve datatypes (26.1MB) hoewel ze beide even vaak zijn uitgevoerd maar toch met een verschillende hoeveelheid eindigen.

Hierdoor kunnen we vaststellen dat het gebruik van wrapperklassen meer geheugen in beslag zal nemen waarbij de heap meer belast zal worden. In tegenstelling tot primitieve datatypes die op de stack worden bijgehouden en dus sneller zullen functioneren.

Merk op dat het bestuderen van geheugen allocatie zoals de heap en de stack zeer moeilijk is. Hierdoor zullen de resultaten van deze meting niet volledig accuraat zijn. In hoofdfunctie wilden we aantonen dat wrapperklassen minder geheugen-vriendelijk zijn de primitieve datatypes.

### Vastelling

Hoewel er in deze applicatie maar een erg klein performantie verschil zal zijn door het zeer weinig gebruik van wrapperklassenn. Merken we dus toch op dat het gebruik van wrapperklassen meer geheugen zal gerbruiken dan primitieve datatypes wat ervoor zal zorgen dat de heap meer belast zal worden en eventueel zal de applicatie trager functioneren.

## 3 – String – StringBuilder

### Beschrijving

### Meting

### Vastelling

## 4 - ?

### Beschrijving

### Meting

### Vastelling

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

# Voorstellen verbeteringen aan auteur

# Reflectie

## Arno Bruynseels

Voor de aanvang van deze opdracht was ik mezelf er niet van bewust dat performantie analyse een activiteit was die in vele verschillende domeinen gekent is. Het verbaasde me dan ook dat er zo frequent aan performantie analyse gedaan wordt.

Nu realiseer ik me hoe belangrijk en impactvol performantie analyse kan zijn voor een applicatie, algortime of om het even welke zaak waarnaar er onderzoek wordt gedaan.

Door het uitwerken van de taak heb ik mezelf kunnen verdiepen in heel het verloop van wat er zich achter de code afspeelt. Dit vond ik een unieke en aangename ervaring omdat dit niet iets is dat elke dag op school wordt aangeleerd.

Tijdens de opdracht hebben we ook kennis gemaakt met de verschillende tools die allemaal beschikbaar zijn om het proces van performantie analyse te ondersteunen. Ik vond het zeer leerrijk om al deze tools eens te vergelijken met elkaar en te kijken wat nu de voordelen/nadelen zijn van een tool t.o.v. van een andere.

Nu ik weet wat performantie analyse inhoud, de impact ervan kan zijn en de voordelen die het aanbeidt. Zal ik in de toekomst, wanneer ik aan het coderen ben, dit altijd in het achteroofd houden waardoor ik kritischer zal kijken naar wat ik heb geschreven heb.