# Big Data Analytics – Abgabe 3

# Sudoku

Ein 9x9 Sudoku besteht aus 9 3x3 Blöcken, die jeweils die Zahlen von 1 bis 9 beinhalten. Zusätzlich darf in jeder Zeile sowie Spalte jede der neun Ziffern nur einmal vorkommen. Der Schwierigkeitsgrad des Sudokus ist dabei abhängig von der Anzahl der freien Felder sowie der Anzahl an Möglichkeiten die Felder zu befüllen.

## Lösungsvorgang

Sudokus können mit verschiedenen Strategien gelöst werden. Am einfachsten ist es damit zu beginnen Felder zu suchen und zu befüllen welche nur wenige Möglichkeiten bieten. Dabei sind dies jene Felder, bei denen der umschließende 3x3 Block sowie die zugehörige Zeile und Spalte viele Ziffern enthält. Ein anderer Ansatz ist es sich verschiedene Zahlen „rauszupicken“, welche schon oft im Sudoku enthalten sind und diese dann zu vervollständigen.



Abbildung : Sudoku (entnommen aus: http://www.247sudoku.com/sudokuEasy.php)

Zum Beispiel könnte man dem obigen Sudoku die Ziffer 4 in dem Block beginnend mit 2,9 nur an einer Stelle einfügen.

## Lösungsalgorithmen

Der erste Ansatz lässt sich vergleichsweise leicht als Algorithmus darstellen. Dafür werden für jedes Feld die Möglichkeiten berechnet und schließlich das Feld mit den wenigsten Möglichkeiten ausgewählt und dort eine Ziffer eingetragen. Diese Vorgehensweise wird wiederholt bis alle Felder ausgefüllt sind. Ist es während der Abarbeitung dieses Algorithmus nicht möglich das Feld weiter auszufüllen, weil die Grundregeln des Sudoku Spiels verletzt werden so wird an dieser Stelle abgebrochen und die vorherige gesetzte Zahl durch eine andere mögliche Zahl ersetzt. Daraufhin wird die Ausfüllung fortgesetzt. Ist es nicht möglich die vorherige Zahl zu korrigieren wird der Algorithmus komplett abgebrochen. In diesem Fall existiert auch keine Lösung für das Sudoku. Es ist möglich mit diesem Algorithmus mehrere Sudoku Lösungen für dieselbe Ausgangssituation zu finden, in dem nicht nach dem ersten Sudoku, sondern erst nach weiteren Backtrackingschritten gestoppt wird.

Ein anderer Ansatz ist der stochastische, dabei werden die Felder zunächst zufällig befüllt. Anschließend wird auf jedes Feld für jede Zeile, Spalte und 3x3 Quadrat eine Fehlerrate über eine Fitnessfunktion berechnet. Das Feld mit der höchsten Fehlerrate wird danach verändert, bis die Fehlerrate null erreicht oder eine zeitliche Abbruchbed

Ein anderer Ansatz ist es das Lösen des Sudokus als Optimierungsproblem darzustellen und mithilfe eines genetischen Algorithmus zu lösen. Dabei werden anders als beim Backtracking die Leerfelder zufällig gefüllt und anschließend bewertet. Dazu werden die beim ersten Durchgang gemachten Fehler – dargestellt durch eine Fitnessfunktion – versucht zu minimieren. Dabei werden z. B. Ziffern in Zeilen vertauscht. Dieser Vorgang wird solange wiederholt, bis es gelungen ist den Fehler auf 0 zu reduzieren bzw. eine Abbruchbedingung wie z. B. eine gewisse Anzahl an Iterationen erreicht ist.  
In Anlehnung an die Evolution werden Individuen gebildet, welche hier als einzelne Sudoku dargestellt werden. Gene bezeichnen jene Werte, welche ein Feld eines Sudokus annehmen kann (Ziffern 1 – 9). Mit der Fitnessfunktion wird die Fehlerrate des jeweiligen Individuums gemessen. Im nächsten Schritt werden bei der Reproduktion zwei oder mehrerer Individuen (Sudokus) genommen und ihre Gene so kombiniert, dass ein völlig neues Sudoku entsteht; dabei werden die Gene ausgewählt (Selektion), die die größte Wahrscheinlichkeit (Likelihood) aufweisen um die Fehlerrate des neuen Individuums zu reduzieren. Dieser Ansatz benötigt allerdings eine relativ hohe Anzahl an Individuen, um gewährleisten zu können, dass sich die Fitness guter Individuen nicht zu sehr von nicht zu sehr von nicht so guten Individuen unterscheiden und eine hohe Mutationsrate aufweisen.

# Implementation

Für die Implementation wurde der zuerst vorgestellte Ansatz verwendet. Dabei kann das Sudoku direkt in eine Klasse überführt werden, welche die Werte zu den jeweiligen Feldern beinhaltet. Als Methoden wird dieser noch die Berechnung der freien Felder sowie des Status, ob das Sudoku gelöst ist hinzugefügt. Als nächstes wird der Solver implementiert, dieser erhält ein Board, welches Rekursiv gelöst werden kann. Dazu wird im ersten Schritt überprüft, ob das Sudoku schon gelöst ist. Ist dies der Fall wird das Board (Sudoku) zurückgegeben. Falls nicht werden für jedes der freien Felder die Möglichkeiten berechnet. Gibt es ein freies Feld, welches keine Möglichkeiten besitzt, so wird abgebrochen und eine leere Liste an Boards zurückgegeben. Andernfalls wird das „beste“ Feld gewählt und befüllt. „Bestes“ Feld ist dabei das Feld mit den wenigsten Möglichkeiten, dieses wird bevorzugt, da die Wahrscheinlichkeit geringer ist bei einem Feld mit wenigen Lösungsmöglichkeiten falsch zu liegen als mit vielen. Aus den Möglichkeiten des „besten“ Feldes werden n Boards generiert. n ist dabei die Anzahl an möglichen Ziffern für das betroffene Feld, eines für jede mögliche Ziffer. Anschließend wird für jedes dieser Boards neue Solver erstellt, welche jeweils eines dieser Boards erhalten und lösen sollen. Sequenziell werden diese Solver dann aufgerufen, liefert einer ein vollständig gelöstes Sudoku zurück wird dieses Sudoku der Ergebnismenge hinzugefügt und zurückgegeben.

# Evaluation

Evaluationsbedingungen: Die Messungen wurden auf einem Rechner mit 8 GB Arbeitsspeicher einer Intel i5-5200U CPU und unter dem Betriebssystem Windows 7 mit Oracle Java 8 JRE durchgeführt. Dabei wurden die Tests in der gleichen Java-Instanz ausgeführt (nachdem bereits ein Sudoku gelöst wurde) damit die Startup-Zeit der JVM und die JIT-Kompilierung die Lösungszeit nicht beeinflussen. Die gemessenen Werte sind jeweils gemittelte Werte von 100 Einzelmessungen.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Besetzte Felder | 35 | 34 | 33 | 34 | 35 | 30 | 30 | 28 | 32 | 29 | 27 | 27 | 26 | 28 | 28 | 24 | 26 | 26 | 26 | 25 | 21 | 17 |
| Lösungszeit | 11 | 4 | 4 | 4 | 3 | 6 | 6 | 4 | 6 | 5 | 3 | 14 | 5 | 10 | 5 | 30 | 4 | 19 | 23 | 10 | 896 | 3307 |

Die Tabelle zeigt 22 Sudokus, die ersten 20 sind von der Webiste www.websudoku.com. Dort gibt es vier Schwierigkeitsgrade, daraus wurden jeweils fünf entnommen. Die Reihenfolge ist dabei easy, medium, hard und evil. Das 21. Sudoku ist das „härteste“ Sudoku, das 22. wurde aus Wikipedia (https://en.wikipedia.org/wiki/Sudoku\_solving\_algorithms) entnommen und ist ein Sudoku, welches speziell gegen Brute Force Lösungen designt ist. Aus den Messdaten lassen sich keine genauen Rückschlüsse zur Abhängigkeit der Lösungszeit bezüglich der Anzahl an besetzten Feldern ziehen. Lediglich Sudoku 21 und 22 zeigen einen starken Anstieg.  
Zusätzlich wurden noch Sudokus mit der Mindestanzahl an notwendigen besetzten Feldern für eine eindeutige Lösung angeschaut. Diese Sudokus haben exakt 17 vorgegebene Felder.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Besetzte Felder | 17 | 17 | 17 | 17 | 17 |
| Lösungszeit | 763 | 334 | 124 | 577 | 123 |

Bei Betrachtung einiger Sudokus der Kategorie mit 17 festen Feldern lässt sich dort auch keine klare Abhängigkeit zur Zeit feststellen, jedoch liegen diese Zeiten deutlich über denen von Sudokus mit 24 – 35 Feldern, was sich durch die Möglichkeiten der „17er“ Sudokus ergibt, welche weit mehr Möglichkeiten der Felder Ausfüllung bieten als andere.