Sudoku Solver Benchmarking

Forschungsmethoden VU 185.A34 WS14/15

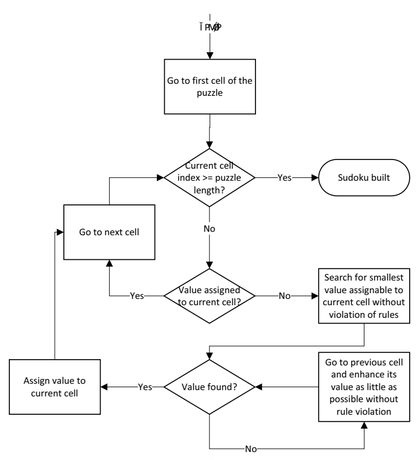
Autoren: Friedrich Decker, Robin Müller, Xiaolin Zhang, Stefan Zischka

Problemstellung:

- Untersuchung von oft benutzten Sudoku Solver Algorithmen und deren Implementierungen in Java
- Effizienz und Einsetzbarkeit dieser Sudoku Solver in der Praxis

4 Sudoku Solver:

Brute Force (BF):
- Backtracking (BT) Strategie
- Implementierung mit Schleife
- Implementierung benötigt keinen Stack

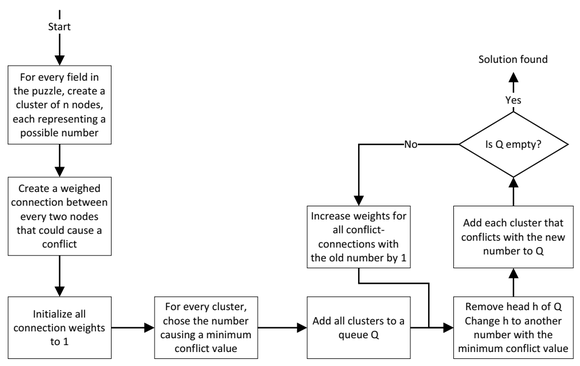


Quelle: eigene Implementierung

Recursive Backtracking (RBT):
- rekursive Backtracking Strategie
- basiert auf gleichem Algorithmus wie Brute Force
- Implementierung mit rekursiven Aufrufen des Lösungsalgorithmus

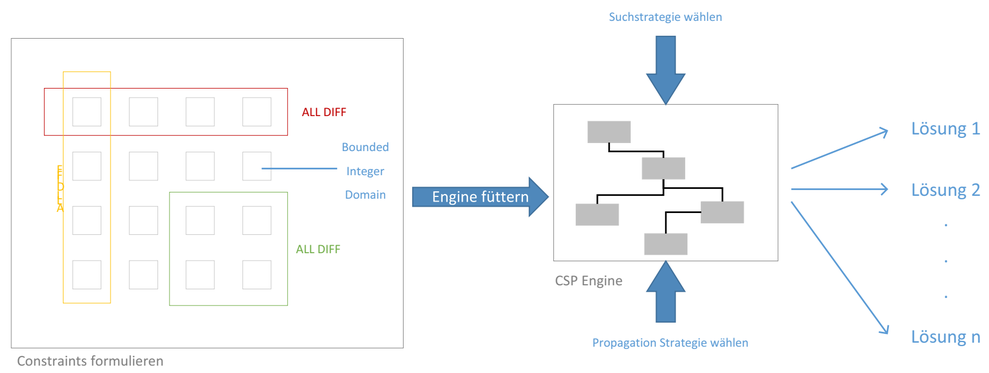
Quelle: eigene Implementierung

Progressive Stochastic Search (PSS):
- sukzessive Reduktion von Konflikten zwischen einzelnen Feldern



Quelle: Jeetesh Mangwani & Pankaj Prateek: Using Progressive Stochastic Search to solve Sudoku CSP; Indian Institute of Technology, Kanpur, India

Constraint Propagation (CP):
- systematische Suche nach Lösung, bis alle Bedingungen erfüllt
- aus bestehenden Bedingungen werden neue abgeleitet bis Wertebereiche aller Zellen so eingeschränkt, dass Lösung vorliegt



Quelle:jacopguide.osolpro.com

Vergleichskriterien & Messmethoden:

- Durchlaufzeit der Kernalgorithmen der Solver
- Maximal durch Solver belegter Speicher während der Ausführung der zentralen Algorithmen
- Difficulty Scaling: Effizienzänderung bei variierender Schwierigkeit von 9x9 Sudoku
- Size Scaling: Effizienzänderung bei variierender Feldgröße

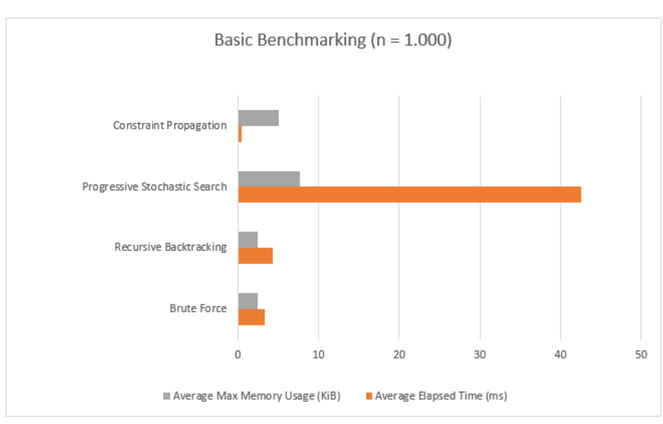
Teststrategie:

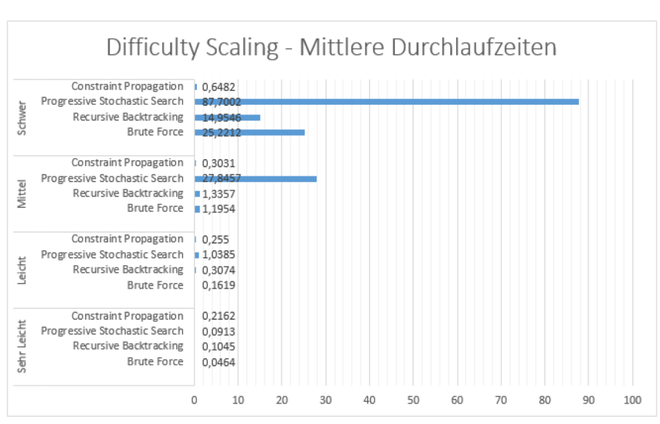
- Messung von Durchlaufzeit und Speicherbelegung
  - I) Basic Benchmarking:
    - > 1.000 Wiederholungen je Solver
    - > Feldgröße: 9x9
    - > Schwierigkeitsgrad: zufällig bei jeder Wiederholung
  - II) Difficulty Scaling:
    - > 100 Wiederholungen je Solver und Schwierigkeitsgrad
    - > Feldgröße: 9x9
    - > Schwierigkeitsgrade: sehr leicht, leicht, mittel, schwer
  - III) Size Scaling:
    - > 0 bis 100*) Wiederholungen je Solver und Feldgröße
    - > Feldgrößen: 4x4, 9x9, 16x16
    - > Schwierigkeitsgrad: mittel

*) In Abhängigkeit der Leistungsfähigkeit einzelner Solver kann bei Feldgröße 16x16 nur Constraint Propagation exakt gemessen werden.

Ergebnisse:

- Basic Benchmarking:
  - > PSS im Mittel sehr ineffizient bei Durchlaufzeit und Speicher
  - > CP sehr niedrige Laufzeiten, mittlere Speicherauslastung
  - > Beide BT Solver mit geringer Speicherbelastung, aber 5 bis 10 mal höhere Laufzeit als CP
- Difficulty Scaling:
  - > CP auf erhöhte Schwierigkeit relativ unempfindlich
  - > konstante Speicherbelastung bei Backtracking & CP
  - > Starke Ausreißer nach oben machen PSS in Durchschnittsbetrachtung langsam, Median ähnlich wie BT
- Size Scaling:
  - > PSS reagiert am stärksten auf Größenskalierung
  - > PSS und BT nur bis 9x9 in Praxis brauchbar
  - > CP löst 16x16 Felder in durchschn. 11 Sekunden





Fazit:

- große Unterschiede zwischen der Leistungsfähigkeit verschiedener Algorithmen
- unterschiedliche Implementierungen des Backtracking Algorithmus annähernd gleich (leicht überlegene Speichereffizienz von Brute Force)
- Progressive Stochastic Search meist ähnliche Performance wie Backtracking, jedoch vereinzelt starke Ausreißer nach oben
- Constraint Propagation Algorithmus am wenigsten empfindlich auf Größenänderung des Feldes (idR. bis 16x16 einsetzbar)
- andere Algorithmen nur bis Feldgröße 9x9 in Praxis einsetzbar und bei einfachen Sudokus mindestens so effizient wie Constraint Propagation
- Speichermessungen beeinflussen allgemeine Performance (Laufzeit, Speicher) negativ
- Java durch weitgehend automatische Speicherverwaltung für Memory-Benchmarking nur bedingt geeignet
- Backtracking kann durch logic Constraints weiterentwickelt werden, um bei größeren Feldern leistungsfähiger zu arbeiten