Projektna naloga Matematika z računalnikom - Sudoku

Patrik Mikuž

22. maj 2024

Predstavitev problema

Sudoku je miselna igra s številkami. Cilj igre je pravilno izpolniti mrežo velikosti $n^2 \times n^2$. Standardna izbira velikosti je $n = 3$, kar nam da mrežo velikosti 9×9. Mreža je tako razdeljena v 9 stolpcev in vrstic, ter v 93×3 območij. Pravilno izpoljnjeno pomeni, da v vsakem stolpcu, vrstici in območju nastopa vsaka od številk 1 do n^2 natanko enkrat. Sudoku problem je veljaven, natanko tedaj ko obstaja enolična rešitev, to pomeni da ne moremo najti dveh različnih rešitev za isti začetni problem. Primer podanega problema in njegovo rešitev predstavlja slika 1.

Začetni namigi so lahko podani povsem naključno ali pa strukturirano. V japonski različici igre velja, da morajo biti začetni namigi podani simetrično. Kot zanimivost velja omeniti, da obstaja takšna sudoku igra, ki ima podanih 17 začetnih namigov (od vseh možnih 81). Enolično rešljiva igra s podanimi 16 ali manj namigi, še ni bila najdena.

Težavnost sudoku problema ocenjujemo na različne načine. Najbolj očiten je število začetno podanih namigov, mogoče je pa tudi ocenjevanje težavnosti s pomočjo globine odločitvenega drevesa, ki ga dobimo pri reševanju z računalnikom.

Projektna naloga je razdeljena v dva dela. V prvem delu sem se osredotočil na reševanje in izboljševanje algoritma, ter primerjavo rezultatov. V drugem delu pa sem preizkusil generiranje novih sudoku problemov ter ocenjevanje njihove težavnosti. Preizkusil sem tudi generiranje na ne standardnih velikostih mreže.

1 Reševanje igre

Z računalnikom rešujemo igro rekurzivno s tako imenovanim *back trackanjem*. To poteka tako, da izberemo prazno polje in pogledamo katere številke lahko notri vpišemo. Izberemo eno izmed teh številk ter postopek ponovimo na tako izpolnjeni mreži. Reševanje je uspešno zaključeno, ko pridemo do končne rešitve. Ko pridemo do primera, da ne moremo več pravilno izpolniti mreže (recimo polje nima več veljavne opcije) s postopkom zaključimo.

Back trackanje samo po sebi je zelo zahteven in zamuden proces. Sploh ker z začetno izbiro odpremo celotno pod drevo. V primeru, da je bila začetna številka napačno izpolnjena, bomo tako morali najprej preiskati začetno drevo.

		3						
8		9	4	6		7		2
2				1	8	6		
					6		7	
		8				4		
	7		8					
		2	9	4				5
5		6		3	2	8		7
						2		

6	1	3	2	7	9	5	8	4
8	5	9	4	6	3	7	1	2
2	4	7	5	1	8	6	3	9
4	2	5	3	9	6	1	7	8
9	6	8	7	5	1	4	2	3
3	7	1	8	2	4	9	5	6
1	8	2	9	4	7	3	6	5
5	9	6	1	3	2	8	4	7
7	3	4	6	8	5	2	9	1

Slika 1: Leva slika prikazuje primer mreže, ki jo želimo pravilno izpolniti. Desna slika prikazuje dobljeno rešitev.

Klasičen back tracking sem tako najprej dopolnil s funkcijo, ki najprej izpolni vsa polja, ki imajo natanko eno možno vrednost, saj lahko s tem bistveno zmanjšamo število vseh vej v drevesu.

V nadaljevanju, sem poskušal reševanje z back trackanjem še bolj pohitriti z vpeljavo osnovne logike v reševanje. Ta sloni na odkrivanju tako imenovanih *skritih parov*. S takšno logiko sem tako eliminiral naslednje veje oziroma možnosti iz polj, kot so navedeni v [3]:

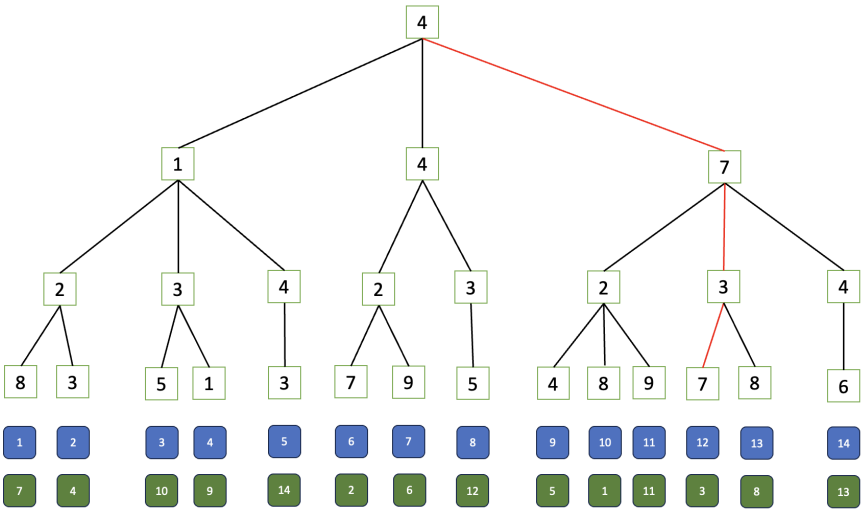
- Če je neka števila v nekem stolpcu, vrstici ali območju mogoča le na enem polju, potem jo postavim na to polje,
- Če se neka številka v nekem stolpcu ali vrstici pojavi le v nekem območju, potem lahko odstranim to številko iz preostalih polj tega območja
- Če se neka številka v nekem območju le v določenem stolpcu ali vrstici, potem jo lahko odstranim iz preostalih polj tega stolpca ali vrstice.

S tako zasnovano logiko, je solver uspešno rešil že veliko večino sudokujev, so pa še vedno obstajali primeri večje težavnosti, ki jih ni uspelo zajeti. Naslednja ideja za pohitritev je bila ta, da vedno izbrem namesto polja, ki odpre čim manj pod dreves. izberem takšno polje, da najbolj zmanjša število možnosti za izpolnjevanje.

Tudi s to pohitritvijo, še vedno nisem uspešno rešil vseh sudokujev, saj še vedno obstaja možnost, da je bila prva izbira napačna in se potem posledično počasi premikamo skozi drevo. Tako sem se odločil zamenjati takšen deterministični solver z naključnim.

Ideja naključnega izbora vej je, da na vsakem koraku naključno izberemo polje in naključno izberemo številko, ki jo vanj vpišemo. V primeru, da pridemo do neveljavne rešitve se tokrat ne vrnemo le en korak nazaj, ampak zavržemo celotno drevo in začnemo zopet z začetno mrežo. Na tak način vsakič izberemo novo vejo in ne raziskujemo celotnega pod drevesa predem ga zavržemo.

Na podlagi tega, sem pripravil 4 različne solverje, s katerimi sem testiral delovanje igre.



Slika 2: Primer drevesne strukture reševanja sudokuja. Z deterministčnim back trackanjem rešujemo drevo od leve proti desni (modri kvadratki), medtem ko z random back trackanjem izbiramo veje naključno (zeleni kvadratki). Pravilna veja je označena z rdečo.

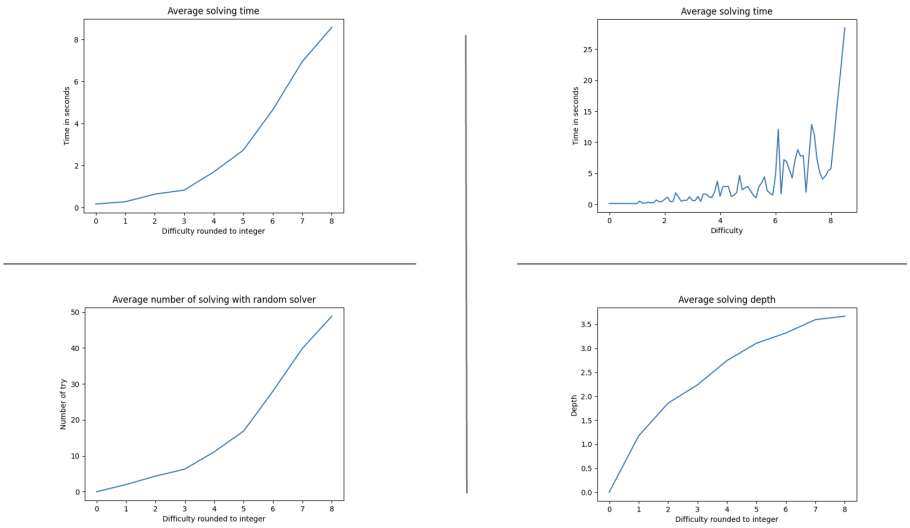
1. Solver z logiko
2. Solver z logiko in iskanjem najboljse pozicije
3. Random solver
4. Random solver z logiko

Deterministična solverja sem hitro opustil, saj sta se izkazala, da dobro delujeta le na neki določeni podmnožici sudokujev, kjer drevo ni preveč razvejano in zlahka zadanemo prvo pozicijo. V vseh ostalih primerih se random solver bistveno bolje odnese.

Opomba 1 *Testiral sem tudi možnost solverja z logiko in raziskovanjem v širino, kot je omenjena v [5]. To je pomenilo, da polje, ki ga naslednjega izberemo pri back trackanju, ne sme biti v enakem stolpcu, vrstici ali območju kot prejšnje. To metodo sem nato opustil, saj to ne vpliva na pohitritev naključnega algoritma.*

Testiranje solverjev sem izvedel na testni množici sudokujev, ki sem jo pridobil v [2]. Sudokuji v testni množici imajo ocenjeno težavnost na podlagi globine drevesa, ki jo doseže program pri reševanju povprečene čez 10 uspešnih reševanj. Za posplošitev težavnosti, sem jo še dodatno zaokrožil na celo število. Povprečna težavnost celotnega dataseta je 1.23, najtežje ocenjen sudoku pa ima težavnost 8.5.

V prvem delu reševanja, sem najprej naredil testno preizkus med random solverjem z logiko in random solverjem brez logike. Preizkus sem naredil na 10 sudokujih, pri čemer sem za težavnost vzel največ 4. Vsak sudoku sem



Slika 3: Grafični prikaz povprečnih vrednosti časa reševanja, število neuspešnih iskanj in povprečne globine drevesa z uporabo random solverja z logiko na testni podmnožici.

rešil 5−krat, vsakič pri drugem random.seed(i) $(0, 1, 2, 3, 4)$. Izkazalo se je, da random solver z logiko v povprečju opravi s sudokujem v 0.59 sekunde, medtem ko solver brez logike potrebuje kar 18.73 sekund. Zato sem se v nadaljevanju fokusiral le še na solver z logiko.

Najprej sem fiksiral verjetnostni algoritem na random.seed(0). Iz vsakega težavnostnega sklopa sem izbral 30 sudokujev, razen iz sklopa s težavnostjo 8, sem izbral le 10 iger, kar je skupaj sestavilo testno množico velikosti 250 sudokujev. Povprečna težavnost moje testne množice je tako postala 3.59.

Vsako od tako dobljenih iger sem rešil z random solverjem z logiko ter primerjal rezultate. Ker metoda deluje na podlagi verjetnosti, sem samo reševanje ponovil z različnimi začetnimi pogoji random.seed(i), pri čemer sem poskus ponovil 10−krat. Tako sem za vsako igro dobil povprečni čas reševanja, globino drevesa rešitve in pa v katerem poskusu sem rešitev našel, katere sem potem grupiral glede na oceno težavnosti. V povprečju smo potrebovali 2.5 sekunde za reševanje enega sudokuja. Spreminjanje povprečnega časa reševanja, globine in neuspelih poskusov glede na ocenjeno zahtevnost in zaokroženo ocenjeno zahtevnost prikazuje slika 3

Dodatno sem testiral še solverja, v primeru, domnevno najtežjega sudokuja na svetu, kot je naveden v viru [4]. Sudoku sem reševal z random solverjem z logiko in sicer 10−krat, pri različnih random.seed(i). V povprečju je solver reševal sudoku 198.71 sekund, rešitev pa je v povprečju bila najdena v 803 poskusih. Če bi ocenjeval težavnost z mojim solverjem in na enak način kot v testni množici, bi ocena bila 6.3.

2 Generiranje novih iger

V drugem delu projekta sem se osredotočil na generiranje novih problemov, pri čemer se nisem omejil le na standardne velikosti, ampak tudi na splošne $m \times n$ igre.

Problema sem se lotil z generiranjem polne, pravilno izpolnjene mreže. Ko smo zgenerirali polno mrežo, smo pričeli z odstranjevanjem polj. Na vsakem koraku, sem iz mreže odstranil indeks ter preveril, ali je tako dobljena igra še vedno enolično rešljiva. Če je odgovor pritrdilen, sem ta indeks odstranil in postopek ponovil, v nasprotnem primeru, sem indeks ohranil v mreži. Ko sem se sprehodil čez celotno mrežo, sem dobil novo generirano igro.

Ta del, je časovno bistveno bolj zahteven kot samo reševanje, saj moramo preverjati enoličnost, kar ni enostavno. Preveriti moramo namreč vse možnosti, da lahko z gotovostjo trdimo, da je igra enolično rešljiva. Zaradi tega sem sprejel odličitev, da če v 10 uspešnih korakih naključnega algoritma, ne najdem dveh različnih rešitev, vseeno sklepam, da je sudoku enolično rešljiv. To je bistvena predpostavka, ki sem jo bil primoran sprejeti, da sem lahko generiral težje sudokuje.

Težavnost generiranih sudokujev sem nato preizkusil z naključnim algoritmom za reševanje in brez logike. V primeru, da sem v kakšen reševanju našel dvoličnost rešitve, sem ga odstranil iz nabora generiranih ugank. Težavnost je ocenjena na enak način kot v podanem datasetu. Vsak sudoku rešim 10−krat in vsakič preštejem globino rekurzije pri reševanju. Za testiranje težavnosti uporabljam random back tracker brez logike. Težavnost je nato definirana kot povprečna vrednost teh reševanj, zaokrožena na 1 decimalko.

Vsaka igra, ima poleg značilko is unique. To pomeni, ali lahko z gotovostjo trdimo, da je takšen problem enolično rešljiv, ali pa obstaja možnost dodatne rešitve.

Generiral sem po 20 sudokujev različnih velikosti in sicer standardnih 3×3 sudokuje ter pravokotne 2×3 in 2×4 sudokuje. Povprečna težavnost klasičnih sudokujev je 3.75. Glede samega časa generiranja, smo v primeru 3×3 sudokuja, v povprečju porabili 37.4 sekunde za generiranje ter kar 193.6 sekund za ocenjevanje težavnosti. Najtežju sudoku, ki sem ga uspel generirati ima ocenjeno težavnost 5.5. Podanih ima 24 začetnih namigov, ocenjevanje njegove težavnosti pa je trajalo 1430 sekund kar je skoraj 24 minut.

Primer najtežjega generiranega sudokuja prikazuje slika 4, slika 5 pa prikazuje še primera pravokotnih sudokujev.

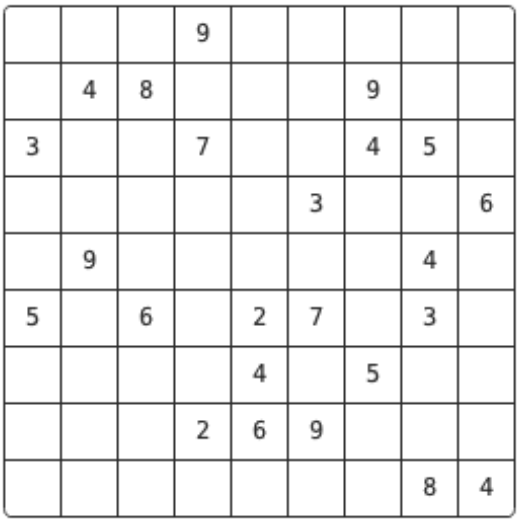
Opomba 2 *$m \times n$ sudoku pomeni, da so bloki pravokotniki m stolpcev in n vrstic. Celotna mreža ima bločno strukturo, kjer je v vsaki od m vrstic n takšnih blokov*

Literatura

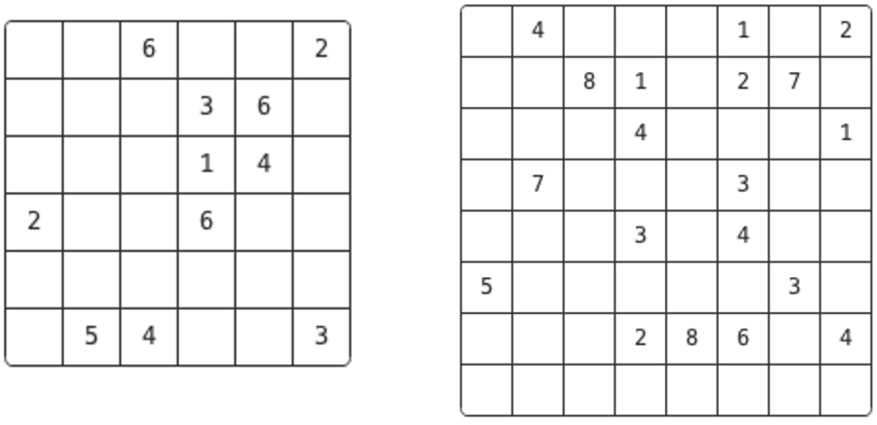
[1] `https://en.wikipedia.org/wiki/Mathematics_of_Sudoku`

[2] `https://www.kaggle.com/datasets/radcliffe/3-million-sudoku-puzzles-with-ratings`

[3] `https://www.sudokuwiki.org/Main_Page`



Slika 4: Najtežje generiran sudoku s predstavljenim generatorjem.



Slika 5: Primera pravokotnih sudokujev. Levi sudoku je 2×3, desni pa 2×4.

[4] https://sudoku2.com/play-the-hardest-sudoku-in-the-world/

[5] https://dlbeer.co.nz/articles/sudoku.html