BI-ZUM Report k semestrální práci

Matěj Latka
FIT ČVUT v Praze

13. května 2020

Téma

Zadáním bylo vytvořit program na **řešení sudoku pomocí technik umělé inteligence** tak, aby bylo řešení provedeno co nejrychleji a garantovaně, tj. řešící algoritmus vždy skončil a dal správné řešení.

Doporučenými technikami bylo CSP (constraint satisfaction problem), tedy modelovat sudoku jako CSP a nad ním pak provádět prohledávání, nebo převod na výrokovu splnitelnost.

Řešení

Vytvořil jsem program, který načte tabulku, zkontroluje, zda její vyplněné hodnoty splňují pravidla a řeší sudoku jedním z následujících způsobů, které využívají CSP:

1. **Manuální**: sudoku řeší uživatel, zadává souřadnice políčka a hodnotu, kterou chce vyplnit. Program ho po každém vyplnění informuje, zda vyplněná buňka splňuje omezení (constraints), či nikoliv. Program končí ve chvíli, kdy je sudoku vyřešeno.

2. **Poloautomatický**: sudoku řeší počítač, ale dokáže vyplnit pouze takové sudoku, které má v každém kroku při vyplňování nevyplněné políčko, jehož přípustná hodnota je právě jedna. V praxi tento požadavek splňují pouze velmi jednoduchá sudoku. Program uživatele informuje, zda se sudoku podařilo vyřešit, či nikoliv, pokud ano, vypíše, která čísla do kterých buněk doplňoval.

3. **Automatický**: sudoku řeší počítač. Dokáže vyplnit libovolnou tabulku tak, aby výsledkem bylo korektně vyřešené sudoku. Rekurzivně generuje n-ární strom možností doplňování přípustných hodnot do políček s tím, že přednostně vyplňuje taková políčka, která mají těchto hodnot nejméně. Po vyplnění políčka pokračuje vyplňováním políček, která jsou na dříve doplněném políčku závislá (ostatní políčka v dotčeném sloupci, řádku a "čtverci") - **constraint propagation**. Pokud jsou všechna závislá políčka vyplněná, vybere algoritmus nejlepší políčko nezávisle na jeho umístění. Během tohoto postupu program kontroluje, zda doplněná hodnota neporušuje některé z pravidel. Pokud by se tak stalo, v generování příslušné větve již nepokračuje a strom tímto prořeže. Když program zjistí, že vytvořená tabulka je již korektním řešením, zastaví rekurzi a projde z uzlu s oním řešením do kořene a ukládá si postup řešení. Ten pak v opačném pořadí vizualizuje společně s vyřešeným sudoku.

   Tento způsob řešení je ošetřen i pro případ, že by se algoritmus zastavil a nenalezl správné řešení. V takovém případě program oznámí, že řešení nenašel. Toto by ale nikdy nemělo nastat, protože vstupní tabulka je při načítání kontrolována, zda splňuje pravidla a doplňování hodnot se též řídí těmito pravidly.

Experimenty

Testoval jsem zejména třetí způsob řešení. Pozorované jevy shrnuje následující tabulka.

soubor	obtížnost	# prázdných	# vyplnění	# vyplnění/ pr.buňka	t bez opt.	t s opt.
sudoku0a.in	zač.	44	74	$\frac{74}{44} \approx 1.68$	0.103 s	0.025 s
sudoku0b.in	zač.	44	44	$\frac{44}{44} = 1$	0.094 s	0.037 s
sudoku1.in	snad.	54	1202	$\frac{1277}{54} \approx 22.26$	1.132 s	0.206 s
sudoku2.in	s. t.	54	3209	$\frac{3209}{54} \approx 59.43$	3.221 s	0.493 s
sudoku3.in	obt.	53	47236	$\frac{47236}{53} \approx 891.25$	58.073 s	3.682 s
sudoku4.in	v. o.	56	3122	$\frac{3122}{56} = 55.75$	4.145 s	0.597 s
sudoku5.in	zap.	53	1773	$\frac{1773}{53} \approx 33.45$	1.340 s	0.234 s
incorrect.in	-	52	0	$\frac{0}{52} = 0$	0.013 s	0.007 s
???	???	81	81	$\frac{81}{81} = 1$	0.181 s	0.065 s

Legenda:

- t: čas (real time),
- opt. optimalizace,
- zač.: začátečnické,
- snad.: snadné,

- s. t.: středně těžké,
- obt.: obtížné,
- v. o.: velmi obtížné,
- zap.: zapeklité,
- pr. buňka: prázdná buňka.

Z tabulky je patrné, že kompilace s optimalizací (-O3) přinesla zásadní zrychlení programu. Vzhledem k tomu, že při běhu programu dochází mnohokrát ke kopírování velkých struktur, předpokládám, že toto zrychlení přineslo například vynucení move sémantiky tam, kde je to možné a podobná vylepšení.

Pokud je zadán neexistující soubor, vytvoří se prázdná tabulka velikosti 9×9. Prázdná tabulka nepochybně splňuje pravidla sudoku, proto je z ní možno vytvořit korektně vyřešené sudoku, což program dokáže při minimálním možném počtu iterací.

Všechna sudoku v tabulce výše byla standardní velikosti 9×9. Program však umí řešit i sudoku jiných velikostí např. 12×12 s vnitřními "obdélníky" 4×3 a 4×4 s vnitřními "čtverci" 2×2, což je možno ověřit na vstupních souborech sudoku-big.in a sudoku-small.in.