Řešení sudoku pomocí optimalizace

František Hájek[1], Šimon Let[2], Ráchel Sgallová[3], Václva Trličík[4]

Gymnázium Uničov[1], Gymnázium Most[2], Gymnázium Christiana Dopplera[3], Masarykovo Gymnázium Vsetín[4]

frahah@seznam.cz[1], lets@seznam.cz[2], rachel.sgallova@centrum.cz[3], flying.viper@seznam.cz[4]

Abstrakt

Sudoku je oblíbená hra pro krácení volné chvíle. Cílem tohoto miniprojektu je seznámení s optimalizačními heuristikami a jejich využití při optimalizaci řešení sudoku. Součástí práce je také porovnání různých optimalizačních metod a klasického řešení pomocí tužky a papíru.

1 Úvod

Heuristika je postup, kterým nenalezneme přesné řešení problému, avšak dostaneme přibližné řešení, které můžeme dostat v poměrně krátkém čase. Velkou úlohu zde hraje pravděpodobnost. Heuristické metody se obvykle používají, pokud neznáme algoritmus, který nám zaručí řešení.

Sudoku N je hlavolam, který obsahuje N^4 číslic 1 až N^2, které se nesmí opakovat v sloupci, řádku a čtverci o rozměrech $N \times N$.

Naše skupina pro řešení sudoku pomocí heuristiky použila programovací jazyky C++, Python a Matlab. Zároveň jsme využili různé metody, které se liší náročností a efektivitou.

2 Random Shooting a Shoot and Go

Random Shooting náhodně generuje vektory a jakmile najde lepší než předchozí, zamění jej. Ukončen je při nalezení vektoru s nulovou chybou. Nevýhoda je velká časová náročnost, jelikož počet stavů je určen vztahem $N^{2\ \text{počet volných míst}}$.

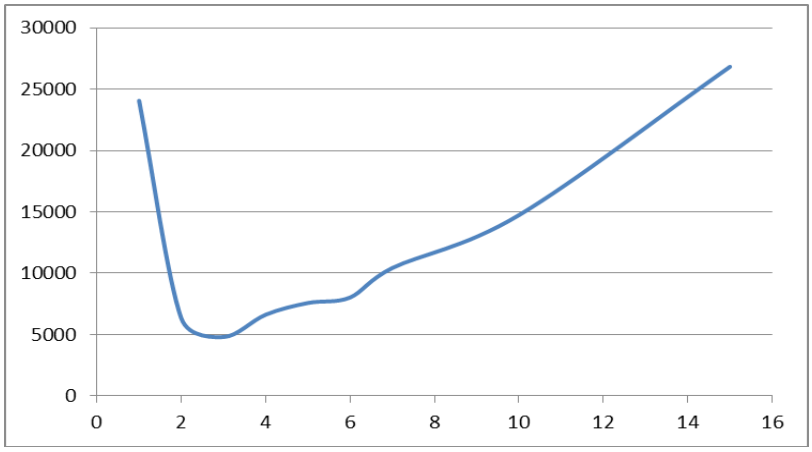
Shoot and Go funguje na podobném principu jako Random Shooting. Rozdíl je v tom, že tento postup po náhodném výstřelu do bodu prohledává jeho okolí až do povolené hodnoty `hmax`. V případě nenalezení uspokojivého výsledku pokračuje dalším náhodným výstřelem. U Shoot and Go jsme vyzkoušeli metodu nalepení na okraj a periodického rozšiřování okolí. Tabulka zachycuje průměrný počet potřebných pokusů k nalezení uspokojivého výsledku u obou metod v závislosti na `hmax`. Obě tabulky se vztahují pro $N = 2$ s následujícím zadáním:

	3		
			3
	4	1	2

Metoda nalepení na okraj (`cmeth = 1`)

hmax	1	2	3	4	5	6	7	10	15
Počet pokusů	24061	6320	4797	6619	7576	8019	10428	14734	26832

Počet pokusů

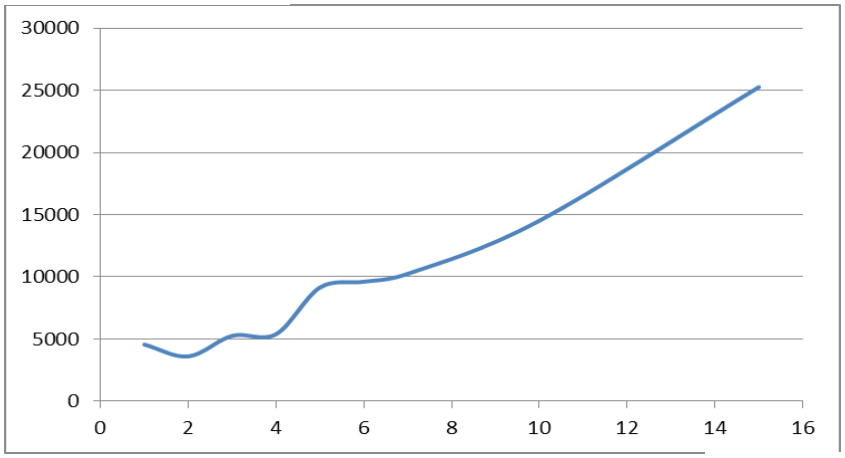


hmax

Metoda periodického rozšíření oblasti (`cmeth = 2`)

hmax	1	2	3	4	5	6	7	10	15
Počet pokusů	4550	3602	5258	5393	9145	9608	10244	14517	25257

Počet pokusů



hmax

```
% method:
% 1 ... nalepeni na okraj oblasti
% 2 ... periodicke rozsireni oblasti
if nargin==3
    method=1;
end
xcor=x;
d=b-a;
n=length(x);

if method==1
    xcor=max(a,min(b,x));
elseif method==2
    xcor=a+mod(x-a,d+1);
```

Ukázka kódu

Z grafů vyplývá, že metoda periodického rozšíření oblasti je při určitých parametrech až dvakrát efektivnější. Stále je však nevhodná pro řešení sudoku vyšších stupňů. U obou metod Shoot and Go se proměnná `hmax` ukázala jako nejvhodnější v rozmezí od 2 do 6. Při vyšších hodnotách se užívání Shoot and Go ukázalo jako neefektivní.

3 Permutační přístup

Doplnění buněk

Buňky se doplní náhodnými čísly, tak aby se v rámci čtverce neopakovala.

Počítání pokuty f(x)

Spočítají se chyby tzn. počet čísel chybějících v každém řádku nebo sloupci.

Cesta k sousedovi

Z celého sudoku se náhodně vybere buňka, která nebyla zadána od začátku, a prohodí se s buňkou, ve stejném čtverci, která také nebyla zadána od začátku. Znovu se spočítá pokuta f(*xnew*) a porovná se s pokutou předcházející, pokud je nová pokuta menší, nový stav se použije. Pokud je pokuta nového stavu větší, použije se s pravděpodobností *p*.

Simulované žíhání(annealing)

Pravděpodobnost p spočítáme vztahem $p = e^{((f(x)-f(xnew))/T)}$, kde T je teplota, která se postupně zmenšuje, když po určitý počet kroků není nalezena dvojice buněk jejichž prohození by způsobilo zmenšení pokuty.

4 Numerické experimenty

Při periodickém přístupu je možné modifikovat hodnoty některých proměnných, a tím měnit účinnost metody při řešení sudoku. Je možné měnit počáteční teplotu - T_0 změnu teploty - Q a *Mez*. *Mez* je určitý počet kroků, když uběhne *Mez* a není nalezena dvojice buněk jejichž prohození by způsobilo zmenšení pokuty, tak se T vynásobí Q ($Q \approx 0.99$).

Tabulka závislosti počtu kroků programu na T_0 Q a *Mezi*.

T_0	Q	Mez	Ø počet kroků (Python)	Ø počet kroků (C++)
80	0.9	30	6000	1525941
40	0.9	30	4140	2316701
20	0.9	30	3752	400269
120	0.9	30	7177	1476392
200	0.9	30	10321	876392
80	0.9	20	5630	904333
80	0.9	10	5382	4671017
80	0.9	60	8234	4547235
80	0.9	180	58467	1104567
80	0.899	30	5752	1419230
80	0.999	30	357266	1123363

Z tabulky lze vypozorovat, že větší efektivity lze dosáhnout snížením T_0 Q nebo *Meze,* ale pokud bychom chtěli řešit sudoku 4 a více nebo velmi složité verze sudoku 3

snížením T_0 Q nebo *Meze* bychom mohli způsobit, že program zůstane viset v lokálním minimu a nikdy nezjistí správný výsledek. Různorodost výsledků a mnohonásobně vyšší počet potřebných kroků při použití C++ připisujeme způsobu, kterým C++ náhodná čísla generuje.

5 Závěr

Zjistili jsme, že nejprimitivnější metoda, Random Shooting, není z časového hlediska výhodná a je nereálné pomocí ní dojít řešení. O něco lépe na tom je metoda Shoot and Go, ale pořád je rychlejší vyplnit sudoku ručně, než ji použít. Při použití Permutačního přístupu se Simulovaným žíháním lze dosáhnout mnohem lepšího výsledku, pokud použijeme tento přístup na C++ a Pythonu, dosáhneme mnohem lepšího výsledku v případě Pythonu a to v řádech sekund, C++ sice za Pythonem pokulhává, ale je stále lepší než když použijete Random Shooting nebo Shoot and Go.

Poděkování

Děkujeme doc. Ing. Jaromíru Kukalovi, Ph.D., děkujeme Ing. Vojtěchu Svobodovi, Csc. za organizaci Týdne vědy a v neposlední řadě děkujeme ČVUT FJFI za poskytnutí prostor k našemu miniprojektu.

Reference

[1] J. Gunther a T. Moon, „Entropy Minimization for Solving Sudoku“, *IEEE TRANSACTIONS ON SIGNAL PROCESSING*, VOL. 60, NO. 1, pp 508-513, JANUARY 2012

[2] P. Babu, K. Pelckmans, P. Stoica, Fellow, IEEE, and J. Li, Fellow, IEEE, *IEEE SIGNAL PROCESSING LETTERS*, VOL. 17, NO. 1, JANUARY 2010