Automatické generovanie logických úloh s využitím SAT solvera

BAKALÁRSKA PRÁCA

Ivan Labáth

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
KATEDRA INFORMATIKY

9.2.1 Informatika

Vedúci: RNDr. Michal Forišek

BRATISLAVA 2010

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

. . . . . .

Chcel by som sa poďakovať svojmu školiteľovi, RNDr. Michalovi Foriškovi, za pomoc, ktorú mi poskytol pri písaní mojej bakalárky.

Abstrakt

Pri tvorbe logických hier vo veľkej miere sa používajú počítače, na čo sa používajú špecializované programy na riešenie daných hier. Žiadúce je, aby zadania mali práve jedno riešenie, pričom pre hry, ako napríklad sudoku, hamilton snake, slither link, nonogram, je dokázané, že problém odpovedania na túto otázku je NP-úplný[Yat03].

V tejto práci skúmame možnosť aplikácie všeobecných SAT-solverov na problematiku tvorenia zadaní. Ukazujeme na troch konkrétnych hrách (sudoku, hamilton snake a chaos), že SAT-solvery nás môžu zbaviť potreby vývoja špecializovaných algoritmov pre mnohé logické hry.

Kľúčové slová: logické hry, SAT, SAT-solver, sudoku, chaos, hamilton snake

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

S rozvojom výpočtovej sily počítačov sa vývoj SAT-solverov stal zaujímavejším, keďže je už možné riešiť dostatočné veľké problémy na to, aby boli zaujímavé aj z praktického hľadiska.

Populárne oblasti aplikácie SAT-solverov v dnešnej dobe sú najmä dizajn a verifikácia hardvéru a plánovanie, ale čoraz viacej sa objavujú aj iné aplikácie.

Naša práca je zameraná na problematiku tvorenia zadaní[1] logických hier. Ako je v [Yat03] ukázané, tento problém často býva NP-úplný, z čoho sa domnievame, že môže byť výhodné riešiť ho SAT-solverom.

Väčšina dostupných zadaní populárnych logických hier je v dnešnej dobe tvorených počítačom, pričom je na to potrebné vyvíjať osobiné programy a občas aj inovatívne algoritmy. My skúmame možnosť tvorenia zadaní deklaratívnejším spôsobom, kde SAT-solveru popíšeme pravidlá hry a generickým algoritmom iteratívnym použitím SAT-solvera vytvoríme zadanie.

Kapitola 1 predstavuje hry ktoré sme v práci skúmali. V nej sú opísané a formálne definované hry chaos, sudoku a hamilton snake, ako aj pojmy, ktoré v tejto práci používame.

Kapitola 2 transformuje formálne definície hier z kapitoly 1 na SAT kódovanie vhodné pre riešenie SAT-solverom. Okrem jednoduchých transformácií, v sekcii 2.3 je riešený zaujímavý problém efektívneho vyjadrenia existencie hamiltonovej cesty ako SAT problém.

Kapitola 3 sa zaoberá samotným tvorením zadaní. Najprv prezentuje generický algoritmus, ktorý sme vytvorili a potom ukazuje na možné spôsoby ovplyvňovania vlastností vytvorených zadaní. Na záver sú skrátka opísané SAT-solvery, ktoré sme používali a ich zdroje.

[1] keď hovoríme zadanie, myslíme napríklad na čiatočne vyplnenú sudoku tabuľku

Kapitola 1

Definície hier

V tejto kapitole slovne opíšeme a formálne definujeme tri hry, ktoré sme skúmali: chaos, sudoku a hamilton snake.

1.1 Všeobecné definície

Najprv definujeme a uvedieme notáciu pre niektoré všeobecné pojmy, ktoré budeme v tejto práci používať.

Značenie. *Množinu prvých n prirodzených čísel budeme označovať* $\mathbb{N}_n$.

$$\mathbb{N}_n = \{1, 2, \ldots, n\}$$

Polia tabuľky budeme číslovať začínajúc v ľavom dolnom uhle (obr. 1.1).

1,m	2,m	$\cdots$	n,m
$\vdots$	$\vdots$	$\ddots$	$\vdots$
1,2	2,2	$\cdots$	n,2
1,1	2,1	$\cdots$	n,1

Obr. 1.1: Číslovanie polí tabuľky

Značenie. ***Polia*** *(prvky)* $\boldsymbol{n \times m}$ ***tabuľky*** $\boldsymbol{T}$ *budeme označovať* $\boldsymbol{T(x, y)}$ *pre* $x \in \mathbb{N}_n, y \in \mathbb{N}_m$.

Definícia 1.1. O **tabuľky T** budeme hovoriť, že je **nad doménou D**, ak platí: $(\forall x)(\forall y)\ T(x, y) \in D$.

Definícia 1.2. SAT kódovanie je dvojica (B, P), kde B je množina alebo vektor boolovských premenných a P je logický výraz v CNF nad premennými z B.

1.2 Chaos

Chaos je hra umiestňovania čísel do tabuľky s cieľom vyplniť tabuľku tak, aby v nej platili určité vzťahy medzi číslami. Pravidlá sú lokálneho rázu, čo by malo umožniť malé $O(nm)$ SAT kódovanie. Prvý výskyt chaosu, ktorý sme našli je v inštrukciách pre World Puzzle Championship 2008[Cha08].

4			2			2
3						
2						
		1			2	
						1
3						1
1	3	4				4

(a) Zadanie

4	3	4	2	2	1	2
3	3	1	4	3	3	1
2	2	4	1	3	3	4
1	2	1	4	2	2	4
4	3	4	1	2	1	1
3	2	3	4	3	3	1
1	3	4	1	2	3	4

(b) Riešenie

Obr. 1.2: Chaos

Cieľ hry je vyplniť obdĺžnikovú tabuľku číslami 1 až 4, pričom nesmú byť rovnaké čísla na ťah šachového koňa, ani v troch za sebou idúcich políčkach, či už horizontálne, vertikálne alebo diagonálne.

Definícia 1.3. Tabuľka T je **$n \times m$ chaos tabuľka**, ak je tabuľka dimenzie $n \times m$ na doméne $\{\emptyset, 1, 2, 3, 4\}$ (obr. 1.2a je 7×7 chaos zadanie).

Definícia 1.4. Tabuľka T je **riešený $n \times m$ chaos**, ak je $n \times m$ chaos tabuľka a platí:

- Každé políčko obsahuje jednu z hodnôt 1 až 4:

$$(\forall x)(\forall y)\, T(x,y) \in \{1,2,3,4\} \tag{1.1}$$

- Nie je trikrát za sebou rovnaké číslo horizontalne, vertikálne, ani diagonálne:

$$D = \{(0,1),(1,0),(1,1),(1,-1)\}$$
$$\neg(\exists x)(\exists y)(\exists(\bar{x},\bar{y}) \in D)\;\; T(x-\bar{x},y-\bar{y}) = T(x,y) = T(x+\bar{x},y+\bar{y}) \tag{1.2}$$

- Nie sú umiestnené dve rovnaké čísla na ťah koňa:

$$K = \{(1,2),(2,1),(1,-2),(2,-1)\}$$
$$\neg(\exists x)(\exists y)(\exists(\bar{x},\bar{y}) \in K)\; T(x,y) = T(x+\bar{x},y+\bar{y}) \tag{1.3}$$

Definícia 1.5. Tabuľka T je **chaos zadanie**, ak jej doplnením je možné získať práve jeden *riešený chaos*.

1.3 Sudoku

Sudoku je populárna hra umiestňovania čísel do tabuľky tak, aby platili doluuvedené pravidlá. Sudoku bola prvýkrát zverejnená v roku 1979 v puzzle časopise v New Yorku, následne v roku 1984 v Japáne, ale v západnom svete sa stala populárna až v 2004. roku, keď sa začala objavovať v The Times of London.[Enc10]

Klasické sudoku

Treba vyplniť 9×9 tabuľku číslami 1 až 9 tak, aby neboli dve rovnaké v žiadnom stĺpci, riadku, ani zvýraznenom 3×3 štvorci, pričom niektoré čísla sú predvyplnené (obr. 1.3a).

Formálne je klasické sudoku definované v definícii 1.8 ako 3×3 sudoku.

Generalizácia

Zmysluplná generalizácia sudoku je parametrizácia veľkosti. Pre zachovanie stĺpcových a riadkových pravidiel, tabuľka musí mať rovnaký počet riadkov ako stĺpcov. Vnútorné oblasti môžu mať rôzny tvar, ale pre naše účely sa ohraničíme na obdĺžniky.

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

(a) Zadanie

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

(b) Riešenie

Obr. 1.3: Klasické sudoku

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

Obr. 1.4: Obdĺžnikové sudoku

			3	2		d			8			7	b	
2		8			7	3				f	d	e		
					4		b	1		c		2		0	5
		9	f	6						0				1	
			d	4			1	6			b	7			
8		4		9	2				a		5				
a							f		1	e			d		9
				8		a							2		c
					e		6		4		3				
	7	0					9		b			f			a
		e					7	0		d			4	c	
		f				8	3					9	5		b
		1	9		0			3		b	4	e	a	f	7
					1				e			b			4
7						6		c			8				
					a								1	5	

Obr. 1.5: Väčšie (4×4) sudoku

Definícia 1.6. Tabuľka T je $\boldsymbol{n \times m}$ **sudoku tabuľka**, ak je tabuľka dimenzie $nm \times nm$ nad doménou $\{\emptyset, 1, 2, \ldots, nm\}$, zložená z $m \times n$ zvýraznených oblastí dimenzie $n \times m$ (obr. 1.4 je 4×2 zadanie sudoku).

Zvýraznené oblasti sudoku tabuľky budeme nazývať **podtabuľky**.

Definujeme si transformáciu, ktorá nám uľahčí indexovať polia poradovým číslom podtabuľky a poľa v nej, aby sme v pravidlách nemuseli písať zložité výrazy pre kvantifikáciu polí podtabuliek.

Definícia 1.7. Transformácia $\boldsymbol{S}$ $n \times m$ sudoku tabuľky T je definovaná bijetívnou reláciou:

$$T^S(s+1, p+1) = T(\lfloor s/m \rfloor + (p \bmod n) + 1, \lfloor p/n \rfloor + (s \bmod m) + 1)$$

Definícia 1.8. Tabuľka T je **riešené** $\boldsymbol{n \times m}$ **sudoku**, ak je $n \times m$ sudoku tabuľka a platí:

- Každé políčko obsahuje jednu z hodnôt 1 až nm:

$$(\forall x)(\forall y)\, T(x, y) \in \{1, 2, \ldots, nm\} \tag{1.4}$$

- V každom riadku sa každé číslo vyskytuje práve raz:

$$(\forall y)(\forall c)(\exists! x)\, T(x, y) = c \tag{1.5}$$

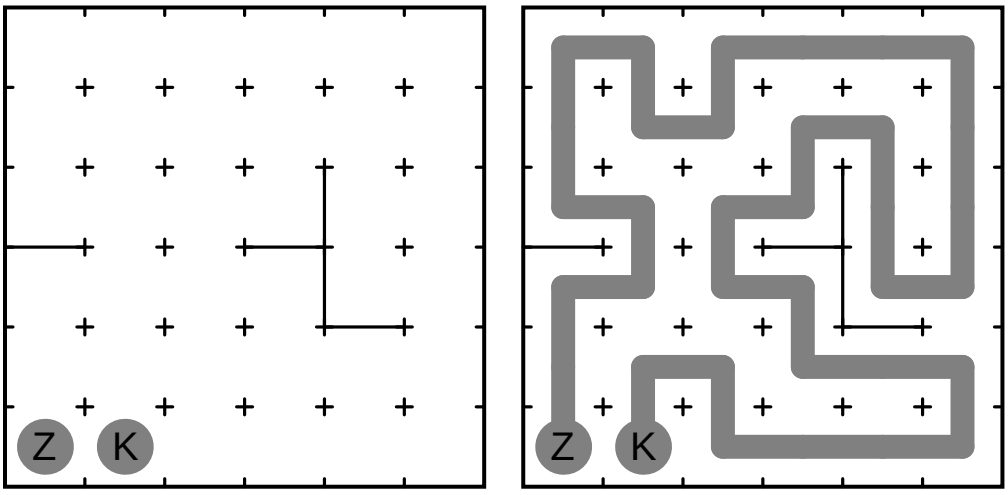
- V každom stĺpci sa každé číslo vyskytuje práve raz:

$$(\forall x)(\forall c)(\exists! y)\, T(x, y) = c \tag{1.6}$$

- V každom zvýraznenom obdĺžniku sa každé číslo vyskytuje práve raz:

$$(\forall s)(\forall c)(\exists! p)\, T^S(s, p) = c \tag{1.7}$$

Definícia 1.9. Tabuľka T je **zadanie sudoku**, ak jej doplnením je možné získať práve jedno *riešené sudoku*.



(a) Zadanie (b) Riešenie

Obr. 1.6: Hamilton snake

1.4 Hamilton Snake

Známych je mnoho hier s názvom snake, kde ide o umiestňovanie hada na hraciu plochu s rôznymi pravidlami. V hre hamilton snake, ako aj jej názov napovedá, ide o umiestnenie hada tak, aby prechádzal všetkými políčkami.

Na danú obdĺžnikovú tabuľku treba nakresliť spojitú čiaru, ktorá prechádza každým políčkom práve raz a medzi políčkami prechádza stenami ktoré nie sú zakázané.

V niektorých verziách cesta má byť uzavretá a v iných je zadaný začiatočný a konečný bod. Kedže je požiadavku uzavretej cesty možné triviálne redukovať na zadanie koncových bodov pri sebe v rohu, ohraničíme sa na prípad so zadanými koncovými bodmi.

Definícia 1.10. Smery v tabuľky definujeme ako vektory:

$$\vec{H} = (0, 1) \qquad \vec{L} = (0, -1)$$
$$\vec{D} = (0, -1) \qquad \vec{P} = (0, 1)$$

Definícia 1.11. Množina D_h smerov platných pre hamilton snake je:

$$D_h = \{\vec{H}, \vec{D}, \vec{L}, \vec{P}\}$$

Definícia 1.12. Hranová $n \times m$ tabuľka T je tabuľka, ktorej vonkajšie hrany sú vyznačené a vnútorné hrany (hrany medzi poľami) sú vyznačené alebo nie, podľa funkcie (1.8), pre ktorú platí (1.9).

Strana poľa $T(x, y)$ v smere $\vec{v} \in D_h$ je vyznačená práve vtedy, keď $T(x, y, \vec{v})$ je $\top$.

$$T : \mathbb{N}_n \times \mathbb{N}_m \times D_h \to \mathcal{B} \tag{1.8}$$

Každá vnútorná hrana patrí dvom poliam a funkcia (1.8) musí byť v tomto ohľade so značením konzistentná.

$$(\forall p)(\forall \vec{v})\ T(p, \vec{v}) \iff T(p + \vec{v}, -\vec{v}) \tag{1.9}$$

Definícia 1.13. Tabuľka T je **$n \times m$ hamilton tabuľka** ak je hranová tabuľka dimenzie $n \times m$ na doméne $\{\emptyset, \mathsf{Z}, \mathsf{K}\}$, kde

$$(\exists! x)(\exists! y)\ T(x, y) = \mathsf{Z}$$
$$(\exists! x)(\exists! y)\ T(x, y) = \mathsf{K}$$

Definícia 1.14. Postupnosť $\{v_i\}$ polí tabuľky je **riešenie** $n \times m$ hamilton tabuľky T, ak platí

- Všetky prvky v postupnosti sú rôzne

$$(\forall i)(\forall j)(i < j)\ v_i \neq v_j \tag{1.10}$$

- Dĺžka postupnosti je rovná počtu polí tabuľky

$$|\{v_i\}| = n \cdot m \tag{1.11}$$

- Prvky tvoria postupnosť susedných polí

$$(\forall i)\ (v_i - v_{i+1}) \in D_h \tag{1.12}$$

- Prvý prvok je v začiatočnom poli tabuľky

$$v_1 = (x, y) \iff T(x, y) = \mathsf{Z} \tag{1.13}$$

- Posledný prvok je v konečnom poli tabuľky

$$v_{nm} = (x, y) \iff T(x, y) = \mathsf{K} \tag{1.14}$$

- Hrana neprechádza zakázanou stenou.

$$(\forall i)\ :\ \neg T(v_i, (v_{i+1} - v_i)) \tag{1.15}$$

Definícia 1.15. Tabuľka T je **hamilton zadanie**, ak je hamilton tabuľka a má práve jedno riešenie.

Kapitola 2

SAT kódovanie

V tejto kapitole ukážeme, ako je možné existenciu riešenia skúmaných logických hier vyjadriť ako SAT problém.

SAT kódovanie sudoku je známy problém, skúmaný aspoň v prácach [ILO06], [Web05] a [KJ06]. Napriek tomu, kódovanie v sekcii 2.2 sme tvorili samostatne.

SAT kódovanie chaosu a hamilton snake sme v dostupnej literatúre nenašli. Chaos kódovanie je veľmi jednoduché a nie je hodné rozoberania v osobitných vedeckých prácach. Je tu najmä ako ukážka možného jednoduchého kódovania niektorých logických hier.

Kódovanie hamilton snake v sebe obsahuje zaujímavý problém kódovania hamiltonovej cesty v grafe. Napriek snahe, v literatúre sme nenašli žiadne pokusy o SAT kódovanie hamiltonovej cesty, až na [HHU07], kde sú ale skúmané výhradne grafy K_n^{*}[1]. V sekcii 2.3 sme vyvinuli kódovanie vhodné pre hľadanie hamiltonovej cesty na grafe ako hamilton snake. Veríme, že naša práca bude prínosom v oblasti SAT kódovania hamiltonovej cesty.

Naším cieľom je získať kódovanie, ktoré umožní SAT-solveru riešiť problém potrebnej veľkosti dostatočne rýchlo na vytváranie zadaní a zároveň je jednoduché na implementáciu.

2.1 Chaos

Chaos je dobrým príkladom na to, ako ľahko je niektoré hry transformovať na SAT problém. Ukážeme priamočiary prevod do SAT kódovania priradením

[1] graf K_n^{*} je kompletný graf na n vrcholov s pridaným jedným samostatným vrcholom

štyroch premenných pre každé pole tabuľky a prepísaním pravidiel do vzťahu premenných.

V nasledovných definíciách formálne popíšeme SAT kódovanie $n \times m$ chaos tabuľky T.

Premenné

Definícia 2.1. Vektor $\boldsymbol{B}$ je **vektor** $4 \cdot n \cdot m$ boolovských **premenných**.

$$B = \{0,1\}^{4nm} \tag{2.1}$$

Definícia 2.2. Funkcia $\boldsymbol{t}$ je bijektívna **funkcia**, ktorá adresuje premenné podľa polohy poľa v tabuľky a čísla na ňom:

$$\begin{gathered} t : \mathbb{N}_n \times \mathbb{N}_m \times \{1,2,3,4\} \to B \\ t(x+1, y+1, c) = B[(x \cdot m + y) \cdot 4 + c] \end{gathered} \tag{2.2}$$

Definícia 2.3. Význam premennej je daný vzťahom:

$$(\forall x)(\forall y)(\forall c) \quad t(x,y,c) \iff T[x,y] = c \tag{2.3}$$

Klauzuly

Pravidlá z definície chaos riešenia (def. 1.4) prepíšeme do sady klauzúl premenných z B, ku ktorým pridáme klauzuly o predvyplnených poliach.

- Každé políčko obsahuje jednu z hodnôt 1 až 4 (relácia 1.1):

$$(\forall x)(\forall y) \;:\; \bigvee_c t(x,y,c) \tag{Pl+}$$

$$(\forall x)(\forall y)(\forall c_1)(\forall c_2)(c_1 < c_2) \;:\; \neg t(x,y,c_1) \vee \neg t(x,y,c_2) \tag{Pl−}$$

- Nie je trikrát za sebou rovnaké číslo horizontálne, vertikálne, ani diagonálne (relácia 1.2):

$$(\forall x)(\forall y)(\forall c) \;:\; \neg t(x-0, y-1, c) \vee \neg t(x,y,c) \vee \neg t(x+0, y+1, c) \tag{Tr1}$$

$$(\forall x)(\forall y)(\forall c) \;:\; \neg t(x-1, y-0, c) \vee \neg t(x,y,c) \vee \neg t(x+1, y+0, c) \tag{Tr2}$$

$$(\forall x)(\forall y)(\forall c) \;:\; \neg t(x-1, y-1, c) \vee \neg t(x,y,c) \vee \neg t(x+1, y+1, c) \tag{Tr3}$$

$$(\forall x)(\forall y)(\forall c) \;:\; \neg t(x+1, y-1, c) \vee \neg t(x,y,c) \vee \neg t(x-1, y+1, c) \tag{Tr4}$$

- Nie sú umiestnené dve rovnaké čísla na ťah koňa (relácia 1.3):

$$(\forall x)(\forall y)(\forall c) \ : \ \neg t(x,y,c) \vee \neg t(x{+}1,y{+}2,c) \qquad \text{(Kn1)}$$
$$(\forall x)(\forall y)(\forall c) \ : \ \neg t(x,y,c) \vee \neg t(x{+}2,y{+}1,c) \qquad \text{(Kn2)}$$
$$(\forall x)(\forall y)(\forall c) \ : \ \neg t(x,y,c) \vee \neg t(x{+}1,y{-}2,c) \qquad \text{(Kn3)}$$
$$(\forall x)(\forall y)(\forall c) \ : \ \neg t(x,y,c) \vee \neg t(x{+}2,y{-}1,c) \qquad \text{(Kn4)}$$

- Vyplnené polia obsahujú hodnotu, ktorá je tam vpísaná

$$(\forall x)(\forall y)\,(T(x,y) \neq \emptyset) \ : \ t(x,y,T(x,y)) \qquad \text{(Zd)}$$

Definícia 2.4. Chaos **klauzuly Ch** v CNF budú:

$$\mathrm{Ch} = \mathrm{Pl}{+} \wedge \mathrm{Pl}{-} \wedge \mathrm{Tr} \wedge \mathrm{Kn} \wedge \mathrm{Zd}$$

Definícia 2.5. SAT kódovanie chaosu bude dvojica **(B, Ch)** – premenné a klauzuly.

Ekvivalencia kódovania 2.5 s definíciou chaosu 1.4 vyplýva z konštrukcie, nebudeme ju teda osobitne dokazovať.

Výsledky

Kódovanie z definície 2.5 je v praxi veľmi efektívne, síce veľa možností na výber ani nebolo. SAT-solvery nemajú problém riešiť tabuľky až do veľkosti približne 22×22, kde v prípade relatívne prázdnych, ale dobre ohraničených tabuliek, ako sa stáva pri minimalizácii, môžu občas potrebovať veľa času. Pri veľkosti 30×30 je na minimalizáciu už potrebné vyše niekoľko hodín.

2.2 Sudoku

Transformácia sudoku na SAT problém je téma, ktorou sa zaoberali už viacerí. Väčšinou to bolo intuitívne kódovanie, ktoré aj my uvedieme, ale známe je aj menšie a rýchlejšie kódovanie [KJ06].

Kódovanie v [KJ06] je v podstate redukcia klasického o nepotrebné premenné, ktoré sú očividne nepravdivé na základe predvyplnených polí. Pre veľké tabuľky (9×9 sudoku – $81 \cdot 81$ polí) je rozhodujúce, ale ľudia také veľké sudoku asi riešiť nebudú a teda nepotrebujeme ani také zadania. Navyše, je

užitočné iba na riešenie zadaní. Pri tvorení zadania riešime aj skoro prázdne tabuľky, kde toto kódovanie je skoro ekvivalentné nášmu, ale je zložitejšie a závisí od vyplnených polí, takže by sme ho museli znovu generovať pre každú tabuľku.

Na sudoku použijeme priamočiary prevod, kde každému poľu priradíme premennú pre každú hodnotu a platnosť sudoku vyjadríme ako vzťah týchto premenných.

Premenné

Definícia 2.6. Výšku, **šírku tabuľky** a počet rôznych čísel v nej budeme označovať $\boldsymbol{w}$.

$$w = nm$$

Definícia 2.7. Vektor $\boldsymbol{B}$ je **vektor** $w \cdot w \cdot w$ boolovských **premenných**.

$$B = \{0,1\}^{w^3} \tag{2.4}$$

Definícia 2.8. Funkcia $\boldsymbol{t}$ je bijektívna **funkcia**, ktorá adresuje premenné podľa polohy poľa v tabuľky a čísla na ňom:

$$t : \mathbb{N}_w \times \mathbb{N}_w \times \mathbb{N}_w \to B$$
$$t(x{+}1, y{+}1, c) = B[(x \cdot w + y) \cdot w + c] \tag{2.5}$$

Definícia 2.9. Význam premennej je daný vzťahom:

$$(\forall x)(\forall y)(\forall c) \quad t(x,y,c) \iff T[x,y] = c \tag{2.6}$$

Poznámka: Transformácia S (def. 1.7) bude tiež platiť aj na funkciu t, pre ľahšie vyjadrovanie vzťahov v podtabuľkách.

Klauzuly

Každé pravidlo z definície 1.8 prepíšeme na dve pravidlá. Kladné bude požadovať aby aspoň jedna z premenných bola pravdivá a záporné aby žiadne dve neboli pravdivé súčasne.

- Každé políčko obsahuje jednu z hodnôt 1 až w (relácia 1.4):

$$(\forall x)(\forall y) \ : \ \bigvee_c t(x,y,c) \tag{Pl+}$$

$$(\forall x)(\forall y)(\forall c_1)(\forall c_2)(c1 < c_2) \ : \ \neg t(x,y,c_1) \vee \neg t(x,y,c_2) \tag{Pl−}$$

- V každom riadku sa každé číslo vyskytuje práve raz (relácia 1.5):

$$(\forall y)(\forall c) \ : \ \bigvee_{x} t(x,y,c) \qquad \text{(Rd+)}$$

$$(\forall y)(\forall c)(\forall x_1)(\forall x_2)(x_1 < x_2) \ : \ \neg t(x_1,y,c) \vee \neg t(x_2,y,c) \qquad \text{(Rd−)}$$

- V každom stĺpci sa každé číslo vyskytuje práve raz (relácia 1.6):

$$(\forall x)(\forall c) \ : \ \bigvee_{y} t(x,y,c) \qquad \text{(Sl+)}$$

$$(\forall x)(\forall c)(\forall y_1)(\forall y_2)(y_1 < y_2) \ : \ \neg t(x,y_1,c) \vee \neg t(x,y_2,c) \qquad \text{(Sl−)}$$

- V každom zvýraznenom obdĺžniku sa každé číslo vyskytuje práve raz (relácia 1.7):

$$(\forall s)(\forall c) \ : \ \bigvee_{p} t^S(s,p,c) \qquad \text{(Šr+)}$$

$$(\forall s)(\forall c)(\forall p_1)(\forall p_2)(p_1 < p_2) \ : \ \neg t^S(s,p_1,c) \vee \neg t^S(s,p_2,c) \qquad \text{(Šr−)}$$

- Vyplnené polia obsahujú hodnotu, ktorá je tam vpísaná

$$(\forall x)(\forall y)\,(T(x,y) \neq \emptyset) \ : \ t(x,y,T(x,y)) \qquad \text{(Zd)}$$

Na korektné zadefinovanie sudoku nie sú potrebné všetky pravidlá. Každá z nasledovných troch sád je postačujúca.

- V žiadnom poli nie sú 2 čísla súčasne a v každom stĺpci, rade a zvýraznenej časti je každé číslo aspoň raz.

$$S_1 = \text{Pl}- \wedge \text{Rd}+ \wedge \text{Sl}+ \wedge \text{Šr}+$$

- V každom poli je aspoň jedno a najviac jedno číslo a v každom stĺpci, rade a zvýraznenej časti je každé číslo najviac raz.

$$S_2 = \text{Pl}+ \wedge \text{Pl}- \wedge \text{Rd}- \wedge \text{Sl}- \wedge \text{Šr}-$$

- Všetky definované pravidlá spolu

$$S_3 = \text{Pl}+ \wedge \text{Pl}- \wedge \text{Rd}+ \wedge \text{Rd}- \wedge \text{Sl}+ \wedge \text{Sl}- \wedge \text{Šr}+ \wedge \text{Šr}-$$

sada	počet klauzúl
S_1	$\frac{1}{2}w^4 - \frac{1}{2}w^3 + 3w^2$
S_2	$2w^4 - 2w^3 + \ w^2$
S_3	$2w^4 - 2w^3 + 4w^2$

Tabuľka 2.1: Počet klauzúl v sadách pravidiel

Experimentálne výsledky ukázali, že najlepšia pre naše potreby je tretia sada S_3. S prvou sadou S_1 si SAT-solvery občas nevedeli poradiť ani na klasickom 3×3 sudoku. Druhá sada bola približne o 3% rýchlejšia na 3×3 sudoku, ale na väčších tabuľkách, kde rýchlosť nebola určená primárne počtom klauzúl už viditeľne zaostávala.

Definícia 2.10. Sudoku **klauzuly Sd** v CNF budú:

$$\mathrm{Sd} = S_3 \wedge \mathrm{Zd}$$

Definícia 2.11. SAT kódovanie sudoku bude dvojica $(\boldsymbol{B}, \mathbf{Sd})$.

Výsledky

So SAT kódovaním z definície 2.11 sme boli schopný vytvárať zadania 3×3 sudoku za dve sekundy a 4×4 sudoku za minútu. Považujeme to za dostatočne dobré výsledky, keďže aj tak málokto bude riešiť 4×4 sudoku alebo väčšie.

Výhoda SAT kódovanie je, že sa veľmi ľahko môžu vyvárať rôzne variácie. Väčšina z variácií na tému sudoku, ktoré sa objavujú v časopisoch je sudoku s pridanými ohraničeniami, ktoré je potom k SAT kódovaniu pridať a vytvárať preň zadania.

2.3 Hamilton Snake

Hru hamilton snake nie je možné tak priamočiaro previesť na SAT problém ako sudoku alebo chaos. Ako aj názov napovedá, v hre sa schováva grafový problém nájdenia hamiltonovej cesty. Z NP-úplnosti oboch problémov vieme, že existuje polynomiálne ohraničený prevod z hamiltonovho na SAT problém. Keďže chceme aj prakticky riešiť hamilton snake, nebude nám stačiť hocaký polynomiálny prevod, ale pokúsime sa nájsť prevod rozumnej veľkosti ktorý je riešiteľný dnešnými SAT-solvermi.

2.3.1 Známe kódovania hamiltonovej cesty

V dostupnej literatúre sme našli iba jeden pokus [HHU07] o prevod problému hamiltonovej kružnice na SAT s cieľom riešiť vzniknutý SAT problém, čo môže byť spôsobené tým, že už dlhší čas[Rub74] sú dostupné výkonné algoritmy na hľadanie hamiltonovej kružnice[Van98] a iba v novšej dobe boli vynájdené algoritmy, ktoré SAT-solverom dali popularitu[vMF10] a výkon[ES03] na riešenie všeobecnejších problémov.

V [HHU07] teoreticky skúmali dolné časové ohraničenie algoritmov bežných SAT-solverov, na dôkaz neexistencie hamiltonovej cesty na grafe K_n s pridaným samotným vrcholom, vzhľadom na použité kódovanie.

Kódovanie hamiltonovej cesty bola požiadavka existencie zoradenia vrcholov do postupnosti, tak aby postupnosť tvorila hamiltonovu cestu. Docielené to bolo priraďovaním vrcholov na polohy postupnosti, pričom pre každý vrchol a pre každú polohu bola premenná, či je daný vrchol na danom mieste postupnosti. V kódovaní boli klauzuly pre zaručenie bijektívnosti priradenia a existencie hrán medzi susednými vrcholmi v postupnosti. Skúmané kódovania sa líšili v klauzulách pre bijektívnosť priradenia.

Na kódovanie grafu bolo potrebných $|V|^2$ premenných, na zaručenie bijekcie $O(|V|^3)$ klauzúl a na popis grafu $O(|V|^2 \cdot |V^2 - E|)$ klauzúl.

V práci ukázali, že zložitosť riešenia SAT inštancie silne závisí od kódovania, pričom pri nevhodnom kódovaní môže byť exponenciálna aj pre jednoduchý problém.

Spomínané kódovanie môže byť vhodné pre skoro kompletné grafy, ale pre redšie grafy potrebuje $O(|V|^4)$ klauzúl. Keďže graf v hamilton snake je maximálneho stupňa 4, bolo by veľmi neefektívne z hľadiska počtu klauzúl.

2.3.2 Naše kódovanie hamiltonovej cesty

Na kódovanie hamiltonovej cesty použijeme hybridný prístup. Najprv sa zameriame na jednoducho popísateľné vlastnosti a potom k tomu pridáme dodatočné požiadavky, aby to bola hamiltonova cesta.

V kódovaní sa zameriame na výber hrán. Budeme požadovať, aby každý vrchol mal v riešení stupeň práve dva (krajné jeden), čím zredukujeme možné výsledky na cestu, prípadne spolu s kružnicami, použitím $O(|V|)$ jednoduchých pravidiel, s ktorými veríme, že by SAT-solver nemal mať problémy.

Keďže takto môžu vzniknúť aj nezávislé kružnice, ošetríme to očíslovaním vrcholov. Na číslovanie vrcholov sme sa rozhodli použiť jednoduché priame

kódovanie s $O(|V|^2)$ premenných a $O(|V|^3)$ klauzúl, podobné ako v [HHU07] ale s menej klauzúl vďaka lepšej reprezentácii grafu.

Zvažovali sme použiť binárne kódovanie, na ktoré by stačilo $O(|V| \log |V|)$ premenných a podobne klauzúl, ale na to by boli potrebné sčítačky, ktoré v sebe majú zakódované xor operácie, pre ktoré je známe[ES06], že sú problematické pre SAT-solvery. Nepredpokladáme teda, že by bolo lepšie na kódovanie spojitosti cesty.

Možné riešenie na vyjadrenie numerických problémov, skúmané v [ES06], je použitie triediacich sietí na čísla, vyjadrené v unárnom kódovaní cifier. Triediace siete sú výhodné v tom, že nevyjadrujú paritu a zachovávajú tranzitivitu klauzúl. Mali by tiež potrebovať $O(|V| \log |V|)$ premenných a podobne klauzúl, pre vhodné konštanty, väčšie ako v binárnom kódovaní, ale výrazne menej ako v priamom kódovaní. Triediace siete by mohli zlepšiť výkon SAT-solverov na zadaniach, kde priame kódovanie je problematické svojou veľkosťou, ale rozhodli sme sa nevyužiť ho, keďže je zložité na implementáciu a priame kódovanie poskytuje dostatočný výkon pre naše potreby.

V ďalšom texte formálne vyjadríme naše kódovanie. Kódovanie sa bude vzťahovať na $n \times m$ hamilton tabuľku T.

Premenné

Budeme mať dve množiny premenných. Jedna bude označovať cez ktoré hrany prechádza hamiltonova cesta a druhá očísluje polia podľa cesty.

Riešenie prechádza vnútornými hranami, ktorých je $n(m-1)$ pre horizontálne hrany a $m(n-1)$ pre vertikálne hrany, čo spolu dáva $2nm - n - m$ potrebných premenných.

Definícia 2.12. Vektor $\boldsymbol{B_s}$ je **vektor** $2nm - n - m$ boolovských **premenných zodpovedajúcich** vnútorným **hranám** tabuľky.

$$B_s = \{0,1\}^{2nm-n-m} \tag{2.7}$$

Definícia 2.13. Funkcia $\boldsymbol{s}$ je **funkcia**, ktorá adresuje premenné z B_s polohou poľa a smeru v ktorom je hrana[2].

[2] Doména uvedená v rovnici 2.8 je kvôli prehľadnosti len orientačná, funkcia premenné z B_s priraďuje vnútorným hranám tabuľky.

$$\begin{aligned} s &: \mathbb{N}_n \times \mathbb{N}_m \times D_h \to B_s \\ s(x, y, \vec{H}) &= s(x, y-1, \vec{D}) \\ s(x, y, \vec{L}) &= s(x-1, y, \vec{P}) \\ s(x{+}1, y{+}1, \vec{D}) &= B_s[y \cdot (2w-1) + x + 1] \\ s(x{+}1, y{+}1, \vec{P}) &= B_s[y \cdot (2w-1) + x + n] \end{aligned} \tag{2.8}$$

Postupnosť hamilton riešenia obsahuje nm polí. Pre každé pole a každé poradové číslo postupnosti budeme mať propozičnú premennú, či je dané pole na danom mieste v postupnosti.

Definícia 2.14. Vektor $\boldsymbol{B_p}$ je **vektor** $(nm)^2$ boolovských **premenných**.

$$B_p = \{0, 1\}^{(nm)^2} \tag{2.9}$$

Definícia 2.15. Funkcia $\boldsymbol{p}$ je bijektívna **funkcia**, ktorá adresuje premenné z B_p podľa polohy poľa a indexu v postupnosti riešenia.

$$p(x, y, c) = B_p[(x \cdot m + y) \cdot (nm) + c] \tag{2.10}$$

Pre zjednodušený zápis klauzúl budeme hodnotu funkcií f a s, pre parametre, na ktorých nie sú definované a teda sú mimo tabuľky, považovať za konštantu false ($\bot$).

Klauzuly

Najprv si vyjadríme klauzuly, ktoré musia platiť pre výber hrán, potom pre číslovanie vrcholov a nakoniec podáme ich vzájomné prepojenie.

- Vnútorné vrcholy majú mať stupeň práve dva. Teda zo štyroch možných hrán, pre každú trojicu musí aspoň jedna byť vybraná (H2+) a jedna nie (H2−).

$$(\forall x)(\forall y)(\forall \vec{v})\,(T(x, y) = \emptyset) \;:\; \bigvee_{\vec{u} \neq \vec{v}} s(x, y, \vec{u}) \tag{H2+}$$

$$(\forall x)(\forall y)(\forall \vec{v})\,(T(x, y) = \emptyset) \;:\; \bigvee_{\vec{u} \neq \vec{v}} \neg s(x, y, \vec{u}) \tag{H2−}$$

- Počiatočný a konečný vrchol musia mať stupeň jeden. Teda zo štyroch hrán musí byť vybraná aspoň jedna (H1+) a z každej dvojici nebyť aspoň jedna (H1−).

$$(\forall x)(\forall y)\,(T(x,y) \in \{\mathsf{Z},\mathsf{K}\}) \;:\; \bigvee_{\vec{u}} s(x,y,\vec{u}) \qquad \text{(H1+)}$$

$$(\forall x)(\forall y)(T(x,y) \in \{\mathsf{Z},\mathsf{K}\})(\forall \vec{u})(\forall \vec{v})(\vec{u} \neq \vec{v}) : \neg s(x,y,\vec{u}) \vee \neg s(x,y,\vec{v}) \qquad \text{(H1−)}$$

- Každý vrchol má priradené jedno číslo od 1 do nm. Teda má aspoň jedno a nemá zároveň dve.

$$(\forall x)(\forall y) \;:\; \bigvee_{c} p(x,y,c) \qquad \text{(P+)}$$

$$(\forall x)(\forall y)(\forall c_1)(\forall c_2)(c_1 < c_2) \;:\; \neg p(x,y,c_1) \vee \neg p(x,y,c_2) \qquad \text{(P−)}$$

- Počiatočný vrchol má priradené 1 a koncový nm.

$$(\forall x)(\forall y)\,(T(x,y) = \mathsf{Z}) : p(x,y,1) \qquad \text{(Pz+)}$$

$$(\forall x)(\forall y)\,(T(x,y) = \mathsf{K}) : p(x,y,nm) \qquad \text{(Pk+)}$$

- Vnútorné vrcholy nemajú priradené 1 ani nm.

$$(\forall x)(\forall y)\,(T(x,y) = \emptyset) : \neg p(x,y,1) \qquad \text{(Pz−)}$$

$$(\forall x)(\forall y)\,(T(x,y) = \emptyset) : \neg p(x,y,nm) \qquad \text{(Pk−)}$$

- Každý vrchol okrem koncového má pri sebe vrchol s číslom o jedno väčším.

$$(\forall x)(\forall y)(\forall c)\,(T(x,y) \neq \mathsf{K}) \;:\; \bigvee_{\vec{v}} p((x,y)+\vec{v},c) \qquad \text{(S+)}$$

- Ak majú dva vrcholy susedné čísla, tak sú spojené hranou. Teda vrcholy nie sú spojené hranou alebo nesusedia číslami.

$$(\forall x)(\forall y)(\forall c)(\forall \vec{v}) \;:\; \neg s(x,y,\vec{v}) \vee \neg p(x,y,c) \vee \neg p((x,y)+\vec{v},c+1) \qquad \text{(S−)}$$

- Hrana nemôže prechádzať zakázanou stenou.

$$(\forall x)(\forall y)(\forall \vec{v})\,(T(x,y,\vec{v})) \;:\; \neg s(x,y,\vec{v}) \qquad \text{(Zd)}$$

Definícia 2.16. Hamilton snake **klauzuly Hs** v CNF budú:

$$\mathrm{Hs} = \mathrm{H2} \wedge \mathrm{H1} \wedge \mathrm{P} \wedge \mathrm{Pz} \wedge \mathrm{Pk} \wedge \mathrm{S} \wedge \mathrm{Zd}$$

Definícia 2.17. SAT kódovanie hamilton snake bude dvojica $(\boldsymbol{B_s \cup B_p}, \mathbf{Hs})$.

Ekvivalenciu kódovania z def. 2.17 s riešením hamilton snake v def. 1.14 načrtneme v nasledovných odstavcoch.

Ak je postupnosť riešenie hamilton snake, tak je možné polia očíslovať poradím v akom sa nachádzajú v postupnosti, vybrať hrany, ktoré spájajú polia susedné v postupnosti, a dostaneme ohodnotenie premenných, pre ktoré platia pravidlá Hs, čo je možné ukázať porovnávaním klauzúl s definíciou hamilton riešenia.

Naopak, ak ohodnotenie premenných B_s a B_p spĺňa podmienky Hs, tak na základe pravidla P, premenné B_p definujú permutáciu vrcholov, z Pz a Pk vyplýva, že prvý a posledný vrchol sú v Z a K. Ak pôjdeme z prvého do posledného vrcholu, z S+ vyplýva, že sa vždy budeme pohybovať k susednému vrcholu, z S−, že medzi nimi bude hrana a na základe Zd hrana nebude tam, kde je stena. Teda ohodnotenie premenných spĺňa definíciu hamilton riešenia.

Pozorný čitateľ si mohol všimnúť, že sme v dôkaze vobec nepoužili pravidlá H2 a H1, teda na zaručenie korektnosti kódovania sú zbytočné, ale ako sme v úvode spomínali, tu sú aby SAT-solver mohol ľahšie uvažovať o riešení.

Výsledky

S kódovaním 2.17, náš algoritmus generuje 6×6 zadania za sekundu, 8×8 za 10 sekúnd, 10×10 za 5 minút a 12×12 za približne dve hodiny.

Pri skúšaní rôznych klauzúl na definovanie riešenia sme zistili, že variácie v klauzulách pre počítanie vrcholov veľmi málo ovplyvňujú čas riešenia, ale aj menšie zmeny v kódovaní výberu hrán robili viditeľné zmeny. Domnievame sa teda, že SAT-solver pri riešení používa najmä výber hrán (redundantné klauzuly H2 a H1), ktorými tvorí spojitú cestu, a iba v prípade, že by sa tá cesta spojila do kružnice, prichádza do výrazu číslovanie, ktoré ukáže, že kružnica nie je dobré riešenie.

Naše kódovanie nie je optimálne. Pre každé políčko máme veľa premenných na číslovanie, čo predpokladáme, že spomaľuje SAT-solver, lebo pri zisťovaní, že riešenie nemôže obsahovať nejakú, dosiaľ neskúmanú kružnicu, musí prechádzať $O(nm)$ premennými, ak nepoužíva nejaké prefíkané algoritmy, ktoré si my momentálne nevieme predstaviť.

Bez ohľadu, že kódovanie nie je optimálne, je to krok dopredu v SAT kódovaní hamiltonových ciest, a pre tvorenie zadaní rozumnej veľkosti je postačujúce.

Kapitola 3

Tvorenie zadaní

V tejto kapitole opíšeme náš algoritmus na tvorenie zadaní a ukážeme, ako je možné malými variáciami jednej časti výrazne ovplyvniť distribúciu zložitosti generovaných zadaní.

Základná idea algoritmov na tvorenie zadaní logických hier, väčšinou je, že sa buď začne s riešením, z ktorého sa uberá, alebo s prázdnym zadaním, do ktorej sa postupne pridáva, prípadne kombinácia oboch postupov. Na týchto zadaniach sa občas ešte konajú všelijaké optimalizácie.

Je mnoho algoritmov a implementácií na tvorbu logických hier, ale nenašli sme žiadne, ktoré by používali SAT-solvery. My sme vytvorili a implementovali generický algoritmus, optimalizovaný pre SAT-solvery. Keďže na riešenie používa SAT-solver, použiteľný je na rôzne hry, bez implementovania potenciálne zložitých algoritmov na ich riešenie.

Keďže pri vytváraní zadaní hier, väčšinou je žiadúce vytvoriť zadania, ktoré patria do nejakej kategórie zložitosti riešenia, v sekcii 3.2 uvedieme povrchný prieskum tejto zložitej problematiky, kde sme využili špecifické vlastnosti SAT-solverov na ladenie distribúcie generovaných zadaní.

Na záver v sekcii 3.3 dáme stručný prehľad SAT-solverov, ktoré sme používali.

3.1 Základný algoritmus

Jeden spôsob vytvárania zadaní je, že sa najprv vytvorí nejaké, väčšinou náhodné, riešenie, z ktorého sa vytvorí množina všetkých prípustných ohraničení. Získaná množina sa potom postupne minimalizuje, postupným ubera-

ním a testovaním, či je zadanie aj ďalej jednoznačné. Táto minimálna množina pravdepodobne tvorí netriviálne zadanie. V praxi sa algoritmus ešte rôznymi spôsobmi vylepšuje. Tento algoritmus nie je pre nás vhodný, pretože na riešenie chceme používať výhradne SAT-solver, ktorého riešenia nie sú náhodné, ale iba malá podmnožina veľkého počtu[1] prípustných riešení.

Rozhodli sme sa použiť opačný postup, kde začneme s prázdnou množinou ohraničení do ktorej budeme postupne pridávať ohraničenia, až sa dostaneme k jednoznačnému zadaniu. Konkrétne začíname s prázdnou tabuľkou, do ktorej v chaose a sudoku vpisujeme čísla a v hamilton snake pridávame steny polí, ale algoritmus je nezávislý na tom, čo sú ohraničenia. Náčrt je podaný v algoritme 3.1.

Algorithm 3.1 Idea algoritmu generovania zadaní

Require: $(B, P) \leftarrow$ SAT kódovanie hry
Ensure: $Zd \rightarrow$ zadanie
1: $Zd \leftarrow \emptyset$ {Začneme s prázdnym zadaním}
2: **repeat**
3: {SAT-solverom určíme koľko má riešení (stačia dve)}
4: $c \leftarrow$ solver$(B, P+Zd, 2)$
5: **if** $c > 1$ **then**
6: {ak je nejednoznačné, pridáme ohraničenie}
7: Zd.push(random)
8: **else if** $c = 0$ **then**
9: {ak nie je žiadne, uberieme posledné}
10: Zd.pop()
11: **end if**
12: **until** $c = 1$

Pomocou tohto algoritmu sme schopný generovať ľubovoľné zadanie, keďže výsledné zadanie nezávisí od riešenia, ktoré nám SAT-solver poskytne. Dokonca ani nepotrebujeme vedieť riešenie, stačí nám informácia, či je existuje riešenie a ak áno, či viacej ako jedno. Výsledné zadanie je v úplnosti určené výberom ohraničení (alg. 3.1:7), ktoré môže byť náhodné, ak chceme vytvoriť náhodné zadanie a v sekcii 3.2 ukážeme, ako je možné pozmeniť distribúciu generovaných zadaní.

[1] klasické sudoku má 6,670,903,752,021,072,936,960 rozličných riešení[Enc10]

Algoritmus 3.1 vytvorí zadania, ale s veľkou pravdepodobnosťou sú zbytočne ľahké. Niektoré ohraničenia, ktoré boli pridané, nemuseli vôbec prispieť na redukovanie možných riešení a ak aj, pridaním nových ohraničení sa niektoré staré mohli stať zbytočnými. Preto zadanie po vytvorení algoritmom 3.1 ešte minimalizujeme.

Na minimalizáciu použijeme algoritmus podobný tomu, ktorý sme spomínali v úvode sekcie. Zo sade ohraničení, ktoré nám korektne definujú zadanie, teda také, že definujú práve jedno riešenie hry, budeme pre každé ohraničenie skúšať, či po jeho ubratí zostáva korektné zadanie a ak áno, tak je nepotrebné a zbavíme sa ho. Pseudokód je uvedený v algoritme 3.2.

Algorithm 3.2 Algoritmus na minimalizáciu zadaní

Require: $(B, P) \leftarrow$ SAT kódovanie hry
Require: $Zd \leftarrow$ zadanie
Ensure: $Zd \rightarrow$ minimálne zadanie

```
for all z ∈ Zd do
    {skúsime zadanie bez ohraničenia z}
    c ← solver(B, P+(Zd− z), 2)
    if c = 1 then
        {ak je riešenie aj ďalej jednoznačné, z nie je potrebné}
        Zd ← Zd − z
    end if
end for
```

Počas generovania zadaní, volania SAT-solvera sú zďaleka najdrahšie operácie. Algoritmus uvedený v alg 3.1 nie je optimálny z hľadiska počtu volaní SAT-solvera. Uvedený je pre ilustráciu princípu generovania zadaní. V našej implementácii sme používali algoritmus 3.3. Pri neúspešnom pokuse pridať ohraničenie, po jeho odstránení netestujeme sadu znovu, pamätáme si, ktoré ohraničenia sme skúšali, aby sme ich neskúšali zbytočne viackrát a nezačíname s prázdnou sadou ohraničení, ale nejakou menšou sadou náhodných.

Ďalšia optimalizácia je, že nepotrebujeme vždy overovať, či je zadanie jednoznačné. Stačí nám vedieť či zadanie má riešenie. Jednoznačnosť budeme overovať iba v nejakých intervaloch závislých od počtu aktuálnych ohraničení, ktoré zvolíme na základe pravdepodobnosti, že zadanie už bude jednoznačné. Prípadné zbytočné ohraničenia, ktoré by sme pridali, by boli odstránené v procese minimalizácie.

Algorithm 3.3 Algoritmus na generovanie zadaní

Require: $(B, P) \leftarrow$ SAT kódovanie hry
Ensure: $Zd \rightarrow$ zadanie
1: $good \leftarrow 0$ {ohraničenia, za ktoré vieme, že sú dobré}
2: $Zd \leftarrow$ random {začneme s niekoľko náhodných ohraničení}
3: $tried \leftarrow Zd$ {ohraničenia, ktoré sme skúšali}
4: **loop**
5: $c \leftarrow \text{solver}(B, P + Zd, 2)$
6: **if** $c = 1$ **then**
7: **return** Zd
8: **else if** $c = 0$ **then**
9: $Zd.\text{pop}()$
10: **end if**
11: **if** $\text{len}(Zd) = good$ **then**
12: **repeat** {vyberieme si nejaké nové ohraničenie, ktoré sme neskúšali}
13: $new \leftarrow$ random
14: **until** $new \notin tried$
15: $Zd \leftarrow Zd + new$
16: $tried \leftarrow tried \cup new$
17: **end if**
18: **end loop**

Možná optimalizácia, špecifická pre SAT-solvery, je v prípade, že odpovede SAT-solverov sú s nejakou výraznou pravdepodobnosťou odhadnúteľné, čo sa v prípade väčších zadaní často stáva. V tom prípade je možné viacero dotazov na SAT-solver, na ktoré očakávame určitú odpoveď spojiť do jednej vhodným kódovaním rozdielnych klauzúl, a opýtať sa na všetky súčasne. V prípade, že odpovedí, ktoré sa odchyľujú od očakávaných je málo, oplácalo by ich bolo určiť binárnym vyhľadávaním. Táto optimalizácia by mohla pre veľké zadania, kde sa občas za sebou vyskytujú aj 30 rovnakých odpovedí SAT-solvera, výrazne zrýchliť algoritmus. Pri menších zadaniach, ktoré sme zhodnotili ako rozumné pre riešenie ľuďmi, zrýchlenie by bolo mierne.

Posledné dve optimalizácie sme neimplementovali. Prechádzajúce, ktoré sme implementovali, zrýchlili generovanie zadaní vyše dvakrát. Neimplementovať posledné dve sme sa rozhodli preto, že dobré SAT kódovanie je robí oveľa väčší rozdiel.

3.2 Generovanie špecifických zadaní

V algoritme 3.3 generujeme náhodné zadania, ktoré by nemali mať ďaleko od uniformnej distribúcie. V prípade, že zadania, ktoré nám vyhovujú sú dostatočne veľká podmnožina všetkých možných, náhodné zadania nám môžu stačiť, z ktorých si môžeme povyberať vyhovujúce.

Algoritmus na hodnotenie, aké sú zadania ktoré generujeme ťažké pre ľudí sme nevyvinuli, keďže je to dosť obsiahla téma, ale skúmali sme ako je možné ovplyvniť distribúciu vytvorených zadaní, ak nebudeme pridávať ohraničenia náhodne, ale využijeme pri tom riešenia, ktoré nám poskytne SAT-solver.

Variácie algoritmu

Vytvorili sme štyri variácie výberu ohraničení pri generovaní zadaní algoritmom 3.3.

- `random` - vyberieme náhodné zo všetkých prípustných ohraničenie

  Pôvodný algoritmus. Mal by generovať náhodné zadania.

- `least` - vyberieme ohraničenie, ktoré je pravdivé v najmenej riešení (ale aspoň v jednom).

  Idea `least` spočíva v tom, že budeme vyberať také ohraničenia, ktoré čo najviac redukujú priestor riešení, čím by sme chceli získať ťažšie

zadania. Problém je, že si môžeme dovoliť SAT-solverom vytvoriť iba nejakú malú[2] podmnožinu riešení, takže je optimalizácia iba lokálna.

- `last` - vyberieme ohraničenie, ktoré je pravdivé v poslednom riešení a čo najmenej iných.

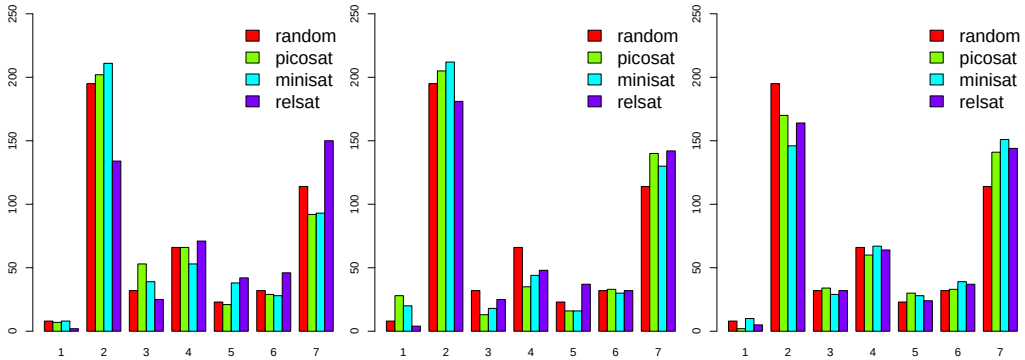
  `last` je pokus o vylepšenie `least`, kde skúsime predpokladať, že pri riešení SAT-solver najprv vygeneruje tie ľahšie riešenia, ktoré bol schopný odvodiť jednoduchými implikáciami a potom bude generovať tie ťažšie. Budeme vyberať ohraničenia tak, aby sme speli k tomu poslednému, ťažšiemu riešeniu. Tieto zadania by mali byť ešte ťažšie ako `least`.

- `first` - vyberieme ohraničenie, ktoré je pravdivé v prvom riešení a čo najmenej iných.

  Podobne ako v `last`, predpokladáme, že SAT-solver bude najprv generovať tie ľahšie riešenia. Budeme vyberať ohraničenia z toho prvého riešenia, aby sme dostali jednoduché zadania.

Výsledky

Pri testovaní sme sa ohraničili na 3×3 sudoku, keďže preňho sme mali dostupný sudoku grader [GF10] - program ktorý je schopný riešiť zadania 3×3 sudoku stratégiami, ktoré bežne používajú ľudia a zhodnotiť asi aké ťažké je zadanie.



(a) `least` (b) `first` (c) `last`

Obr. 3.1: Údaje z tabuľky 3.1 vo forme grafu

[2] napríklad 47 prvkovú

[GF10]	—	picosat			minisat			relsat		
rating	`random`	`first`	`last`	`least`	`first`	`last`	`least`	`first`	`last`	`least`
1	8	28	2	7	20	10	8	4	5	2
2	195	205	170	202	212	146	211	181	164	134
3	32	13	34	53	18	29	39	25	32	25
4	66	35	60	66	44	67	53	48	64	71
5	23	16	30	21	16	28	38	37	24	42
6	32	33	33	29	30	39	28	32	37	46
7	114	140	141	92	130	151	93	142	144	150
8	0	0	0	0	0	0	0	1	0	0
9	0	0	0	0	0	0	0	0	0	0

Tabuľka 3.1: Hodnotenie sudoku graderom 470 zadaní pre každú stratégiu a solvery relsat, picosat a minisat

Pre stratégie `least`, `last` a `first` sme pre každý so SAT-solverov relsat, picosat a minisat vyvorili 470 zadaní, pričom pri každej iterácii algoritmu bolo vytvorených 47 rozdielnych riešení. Zadania sme ohodnotili sudoku graderom a porovnávali ich počty so stratégiou `random`. Údaje sú prezentované v tabuľky 3.1 a tiež v podobe grafu na obr. 3.1.

Konkrétne hodnoty nás nezaujímajú, ale zaujímavé sú rozdiely distribúcií, ktoré vznikli medzi náhodne generovaným sudoku a nejakou inou stratégiou.

Vidno je, že relsat so stratégiou `least` vyváral ťažšie zadania, ale ostatné solvery ani nie, čo je pravdepodobne spôsobené tým, že pre relsat sme používali interné generovanie viacero riešení a ostatné solvery to nepodporovali, takže boli spúštané viacej krát s pridanými klauzulami, že predchádzajúce riešenia sú neprípustné.

Z tabuľky je tiež vidno, že picosat a minisat stratégiou `first` vytvárali výrazne viacej najľahších zadaní, ako stratégia `random`.

Ukázali sme teda, že podobným spôsobom je možné ovplyvniť ťažkosť zadania, ale bol by potrebný ďalší výskum v tejto oblasti, aby to bolo užitočné.

Problémy

Problém pre všetky spomínané metódy vyberania ohraničení, okrem `random` je, že vytvorené zadania môžu mať nejakú nežiadúcu štruktúru (obr. 3.2). Výsledky závisia od konkrétnej hry, použitého SAT-solvera a metódy gene-

	2	2	3		3		1
		3				4	
			1				1
2							
3							2
1							1
3				4		1	3

(a) Zadanie

4	2	2	3	2	3	2	1
3	2	3	4	3	4	4	2
4	1	4	1	2	1	4	1
1	2	1	2	1	2	1	2
2	3	4	3	4	3	4	3
3	4	3	2	3	2	1	2
1	1	2	1	4	1	2	1
3	4	3	2	4	4	1	3

(b) Riešenie

Obr. 3.2: Pravidelný chaos vytvorený metódou `least` a solverom picosat

2	1	4	3		1		3
3		3				2	
4							
3						1	
					1		
2							4
2				4		3	

(a) Zadanie

2	1	4	3	4	1	4	3
3	2	3	2	3	2	2	4
4	1	4	1	4	1	2	3
3	4	1	2	3	4	1	4
2	3	2	3	4	1	2	3
3	4	1	2	3	2	1	4
2	4	1	1	4	1	2	4
2	2	3	2	4	2	3	3

(b) Riešenie

Obr. 3.3: Pravidelný chaos vytvorený metódou `least` a solverom relsat

rovania. Chaos generovaný inými SAT-solverami (obr. 3.3) preukazuje iba menšiu mieru štruktúrovanosti.

Predpokladáme, že je chaos je problematický pre tieto metódy preto, že pravidlá sú malé, lokálne, všade rovnaké a SAT-solver má tendenciu rozhodovať sa rovnakým spôsobom, keď má na výber viacero možností, čo má za následok vytváranie pravidelných riešení.

Pravidelnosti sme si nevšimli v ostatných hrách, ale nemôžeme tvrdiť, že žiadne nie sú. V sudoku a hamilton snake každé ohraničenie komplexne interaguje so všetkými ostatnými a vždy pridávame iba po jedno z mnohých ohraničení, ktoré sú potrebné na určenie konkrétneho riešenia, takže veríme, že pre tieto hry môžu byť osožné metódy `least`, `last` a `first`.

Dobré výsledky by mohla priniesť kombinácia metód `random` a `least`, ak by sa najprv pridali nejaký počet ohraničení a iba pri konci by sa zadanie doladilo inou metódou. Osožné by mohlo byť aj obmieňanie viacerých solverov počas generovania.

3.3 Použité SAT-solvery

V dnešnej dobe sú voľne dostupné výkonné SAT-solvery.

Ako dobrý zdroj výkonných SAT-solverov na všeobecné problémy považujeme Medzinárodnú súťaž SAT-solverov[vMF10]. Je to otvorená súťaž, v ktorej sa SAT-solvery súťažia v niekoľkých kategóriách riešením rôznych SAT-problémov. V posledných rokoch je organizovaná v dvojročných intervaloch.

Nevýhoda týchto solverov je, že väčšinou nie sú zamerané na aspekty, ktoré sa v súťaži nehodnotia, ako napríklad generovanie viacej riešení, ktoré interne podporuje iba relsat.

O relsat-e sme sa dozvedeli od Michala Foriška, vedúceho práce. K solverom picosat, precosat a minisat sme sa dostali na základe súťaže [vMF10].

- picosat - solver z roku 2007 Armina Bierea, ktorý v 2007. roku získal jedno prvé a niekoľko druhých miest SAT súťaže [vMF10]. Prezentovaný je v článkoch [Bie08] a [Bie09]. Pre naše účely sa ukázal ako relatívne rýchly, občas mierne pomalší ako minisat.

  Dostupný je na stránke: `http://fmv.jku.at/picosat/`

- precosat - solver z roku 2009 Armina Bierea[JBH10], ktorý v 2009. roku získal jedno prvé a niekoľko druhých miest SAT súťaže [vMF10]. Prezentovaný je v článkoch [JBH10] a [Bie09]. Svoju silu sčasti berie z predspracovania a funkčnej substitúcie, o ktorej tvrdí, že rozpoznáva xor a niektoré iné operácie. V našom kódovaní podobné operácie nevyskytovali, čo je pravdepodobne dôvod, prečo sa v praxi ukázal ako mierne pomalší od minisat a picosat. Mohol by byť dobrá voľba, pre kódovania so zložitejšími operáciami.

  Dostupný je na stránke: http://fmv.jku.at/precosat/

- minisat - populárny minimalistický SAT-solver. Vytvorili ho Niklas Eén a Niklas Sörensson [ES03]. V 2005. a 2006. roku vyhral niekoľko druhých miest SAT súťaže [vMF10]. Pre naše úlohy sa ukázal ako najrýchlejší, trochu rýchlejší ako picosat. Prevahu pravdepodobne získal svojou jednoduchosťou, keďže naše kódovanie bolo relatívne priame.

  Dostupný je na stránke: http://minisat.se/

- relsat - solver ktorý vytvoril Roberto Bayardo. Prezentovaný je 1997. roku v článku [BS97]. Pri obyčajnom riešení SAT inštancií, kde nás zaujíma iba jedno riešenie je pomalší. Veľkosť vstupov, s ktorými začína mať problémy je nižšia ako u ostatných. Na druhej strane, má mnohé príjemné vlastnosti, z ktorých najdôležitejšia pre nás je interná podpora pre vytvárania viacero riešení. S ním sme boli schopný generovať veľké množstvo riešení v rozumnom čase, pričom ostatné solvery by sme museli spúšťať viackrát, čím by veľakrát opakovali rovnakú prácu. Má aj optimalizovaný algoritmus určovanie počtu riešení.

  Dostupný je na stránke: http://www.bayardo.org/resources.html

Záver

V našej práci sme ukázali, že tvorenie zadaní logických hier pomocou SAT-solvera je možné a dokonca aj výhodné, ak je cieľ ľahko urobiť zadania pre rôzne hry.

Ukázali sme jednoduchosť kódovania niektorých logických hier na príklade chaosu a sudoku, kde stačilo korektne formálne zapísať pravidlá hry ako inštanciu SAT problému a trochu prihliadať na to, že aj redundantné pravidlá môžu byť osožné, ak ich odvodenie z ostatných pravidiel je pre SAT-solvery ťažké.

Pre hru hamilton snake sme vyvinuli SAT kódovanie hamiltonovej cesty, idea ktorého sa môže aplikovať aj na kódovanie spojitosti grafu. Naše kódovanie nie je optimálne, ale pre potreby hamilton snake, a veríme, že aj mnoho iných logických hier, je úplne postačujúce.

Vyvinuli sme generický algorimus na tvorenie inštancií logických hier, ktorý je aplikovateľný na rôzne hry, pokiaľ je pre ňu dostupné rozumné SAT kódovanie.

SAT kódovanie hamiltonovej cesty a spojitosti grafu sú zaujímavé problémy, ktoré zatiaľ zostávajú nedostatočne preskúmané a sú dobrou témou na ďalší výskum.

Zadanie vytvorené naším algoritmom sú náhodné a síce sme ukázali spôsob, ako je môžné pozmeniť distribúciu generovaných zadaní, často je potrebné vedieť, aké ťažké na riešenie sú konkrétne zadania. Prieskum možných generických riešení, ktoré by boli aplikovateľné na rôzne hry by bol tiež zaujímavou témou na ďalší výskum.

Zdrojový kód programu je v prílohe a neurčitú dobu bude spolu s touto prácou dostupný aj na niektorých z týchto stránok: http://people.ksp.sk/~ivan/, http://matrix.dre.am/ alebo http://www.st.fmph.uniba.sk/~labath3/

Literatúra

[Bie08] Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008. 3.3

[Bie09] Armin Biere. Precosat, picosat at sat competition '09, 2009. System Description for the SAT'09 SAT Competition. 3.3

[BS97] Roberto J. Bayardo and Robert C. Schrag. Using csp look-back techniques to solve real-world sat instances. In *Proceedings of the National Conference on Artificial Intelligence*, pages 203–208. AAAI Press, 1997. 3.3

[Cha08] World Puzzle Championship. Instruction booklet, 2008. www.forsmarts.com/pdf/WPC2008_Instruction_Booklet.pdf. 1.2

[Enc10] Encyclopædia Britannica. Sudoku. In *Encyclopædia Britannica Online*, 2010. http://www.britannica.com/EBchecked/topic/1214841/sudoku. 1.3, 1

[ES03] Niklas Een and Niklas Sörensson. An Extensible SAT-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing*, 2003. 2.3.1, 3.3

[ES06] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006. 2.3.2

[GF10] AT&T research Glenn Fowler. A 9x9 sudoku solver and generator, 2010. http://public.research.att.com/~gsf/sudoku/. 3.2, ??

[HHU07] Alexander Hertel, Philipp Hertel, and Alasdair Urquhart. Formalizing dangerous sat encodings. In *SAT'07*, pages 159–172, 2007. 2, 2.3.1, 2.3.2

[ILO06] Ines Lynce Ist, Inês Lynce, and Joël Ouaknine. Sudoku as a sat problem. In *Proceedings of the 9 th International Symposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale*. Springer, 2006. 2

[JBH10] Matti Järvisalo, Armin Biere, and Marijn Heule. Blocked clause elimination, 2010. Tools and Algorithms for the Construction and Analysis of Systems. 3.3

[KJ06] Gihwon Kwon and Himanshu Jain. Optimized cnf encoding for sudoku puzzles, 2006. 2, 2.2

[Rub74] Frank Rubin. A search procedure for hamilton paths and circuits. *J. ACM*, 21(4):576–580, 1974. 2.3.1

[Van98] Basil Vandegriend. Finding hamiltonian cycles: Algorithms, graphs and performance, 1998. 2.3.1

[vMF10] Hans van Maaren and John Franco. The international sat competitions web page, 2010. http://www.satcompetition.org/. 2.3.1, 3.3

[Web05] Tjark Weber. A SAT-based Sudoku solver. In Geoff Sutcliffe and Andrei Voronkov, editors, *LPAR-12, The 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, Short Paper Proceedings*, pages 11–15, December 2005. 2

[Yat03] Takayuki Yato. Complexity and completeness of finding another solution and its application to puzzles, 2003. (document)

Dodatok A: Príklady zadaní

	2		3			1	3
3	1					1	4
1							
4					3		
1	3	2				4	

1	1						3
1	2						
2						2	1
						3	3
3							
		1					1
	3						3

Obr. 4: Chaos

4	1	2	1	4					2				2					1	3
1										3		4						2	3
2								1											
3																			
	3			2										2					
2	1		3														3	2	
2	4			3	4					4			2	3	1	3			2

Obr. 5: Chaos

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

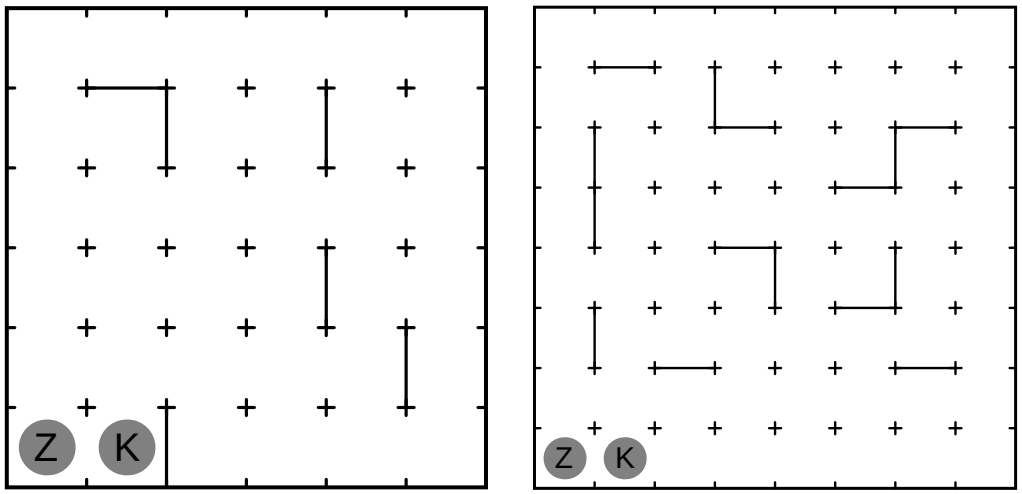
		3	2	1				6
					7			5
8				6			2	
	8			2				4
6				8				7
		7				9		
	1							
9							5	8
					4		9	3

Obr. 6: Sudoku

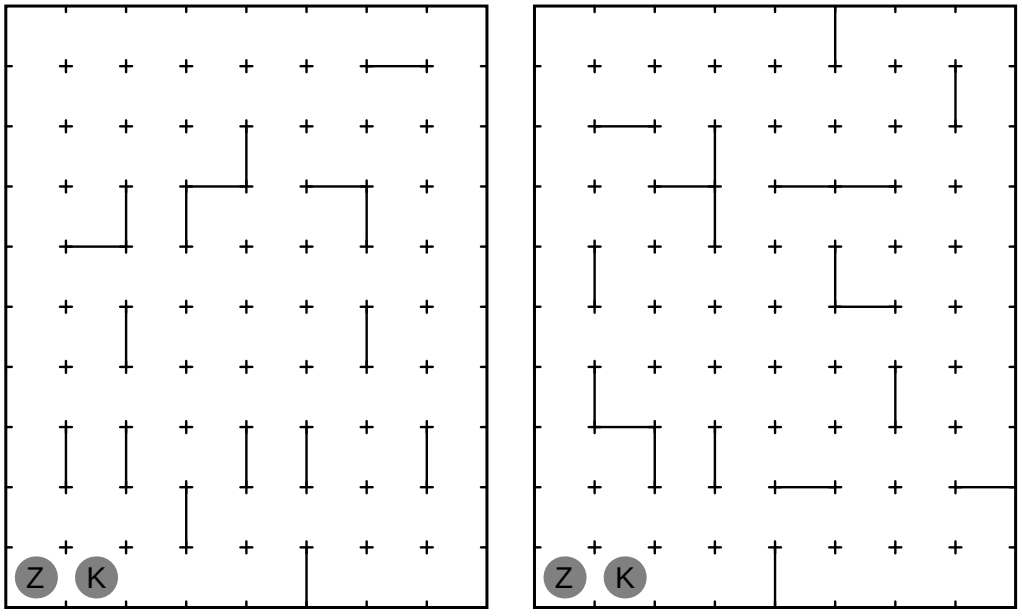
					5		a				
					7						b
		7					0		8		
				9	3		6				
9	5	b						a		8	
		2		a				5	0		
					6	b		3	7		
1					9		3				2
			4	5		1				a	
2			0		4					1	
	7	5							b		
4		a	9		0		2	7		5	

4				6			8		3
		0	2				9	7	
			5		8				
		8					4		
7						2			
	9			0		3		8	6
	5			3					
	6				1		2		
		9			4				
3			4		6			0	

Obr. 7: Sudoku



Obr. 8: Hamilton



Obr. 9: Hamilton