Algoritmy

Speciální případy řešitelné v ~~CNF~~ polynomiálním čase

A) 2-SAT (~~Každá~~ klausule má nejvýše 2 literály)

Algoritmus pro formuli F

1. opakuj rezoluční uzávěr formule F.

   pokud ~~nějaká~~ $|C_1| \le 2, |C_2| \le 2$ pak $|r(x, C_1, C_2)| \le 2$. Velikost rezolučního uzávěru je tak omezena $|Var(F)|^2$.

2. pokud ~~prázdná~~ do uzávěru patří $\square$, není F splnitelná. Jinak je splnitelná.

B) Horn-SAT

F je Hornova formule, pokud každé $C \in F$ obsahuje nejvýše jeden pozitivní literál.

klausule $(x \vee \bar{y}_1 \vee \bar{y}_2 \vee \bar{y}_3 \dots \bar{y}_k)$ interpretujeme jako (= je ekvivalentní) $(y_1 \wedge y_2 \wedge y_3 \dots y_k) \to x$

(x) $\quad\vdots\quad$ $1 \to x$

$(\bar{y}_1 \vee \bar{y}_2 \vee \dots \bar{y}_k)$ $\quad\vdots\quad$ $(y_1 \wedge y_2 \wedge y_3 \dots y_k) \to 0$

Teoretický a algoritmický základ logického programování a prologu. (spolu s rezolucí).

Algoritmus: vstup je F

1. dokud existuje $\{x\} \in F$, $F \in F\{x=1\}$. // $x \in Var(F)$.
2. pokud $\square \in F$, pak je F nesplnitelná
3. F je splnitelná, lze je splnit ohodnocením proměnných na 0

Algoritmus pracuje v polynomiálním čase: cyklus v kroku 1 proběhne max $|Var(F)|$ krát, v každém ~~cyklu kroku~~ iteraci projdeme F jednou.

Korektnost: na vstupu 3 F neobsahuje klausuli bez negativního literálu. Na výstupu jedna dělané „vymazaná ohodnocení"; případ 2: $\square \in F$ pk. jsme v nějaké klausuli ohodnotily všechny literály na 0.

Algoritmy založené na backtrackingu

Jednoduchý backtracking

BACKTRACK(F)

1. pokud $\square \in F$, vrať 0
2. pokud $\emptyset = F$, vrať 1
3. vyber $x \in Var(F)$
4. pokud BACKTRACK$(F[x=1]) = 1$, vrať 1
5. vrať BACKTRACK$(F[x=0])$

Věta: Nechť F je kontradikce. Nechť k je minimální možný počet rekurzivních volání BACKTRACK, spustíme-li algoritmus pro F a uvažujeme-li všechny možné výběry na řádku 3 (tj. všechy kontextově algoritmy lišící se řádkem 3).
Potom existuje rezoluční zamítnutí F s nejvýše k klauzulemi.

Důkaz: Představme si strom rekurzivních volání. každé hraně v něm můžeme přiřadit ohodnocení nějaké proměnné (a to ta, která je vybrána na ř. 3).
Po cestě z kořene do vrcholu tak můžeme sečíst částečné ohodnocení proměnných tak, že spojíme ohodnocení nalezená na hranách na cestě. Označme toto ohodnocení PATH(S).

Důkaz provedeme tak, že každému vrcholu s přiřadíme klauzuli C tak, že C PATH$(S) = 0$. Tím pádem musíme kořeni přiřadit klauzuli $\square$, protože ta je nepravdivá pro prázdné ohodnocení.

Začneme u listů: je-li s list, musí existovat klauzule $C \in F$ taková, že C PATH$(S) = 0$ (jinak by s nebyl list, viz řádek 1 algoritmu, kde $\square \in F$; a $\square$ se dostala do F tak, že jsme z nějaké $C \in F$ odebrali nepravdivé literály).

Obecně: Nechť s je vnitřní vrchol a s potomky s_1, s_2 které již mají přiřazeny klauzule C_1, C_2; dále nechť x je proměnná vybraná v kroku 3 v rek. volání odpovídajícím s.

Pokud $x \in Var(C_1) \cap Var(C_2)$, potom musí být $x \in C_1$ a $\bar{x} \in C_2$, protože podle předpokladu C_1 PATH$(S_1) = 0$ a C_2 PATH$(S_2) = 0$.
Vrcholu s přiřadíme $r(x, C_1, C_2)$.

Pokud $x \notin Var(C_1) \cap Var(C_2)$, přiřadíme s tu klauzuli z C_1, C_2, která neobsahuje proměnnou x.

V obou případech platí, že klauzule přiřazená s je nepravdivá pro ohodnocení PATH(S).

Rezoluční zamítnutí pro F dostaneme tak, že vrcholy stromu seřadíme v opačném pořadí, než v jakém bychom je potkali průchodem do šířky. Vyrecháme opakující se výskyty klauzulí (zprava).

∎

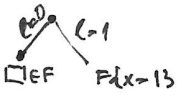
DPLL

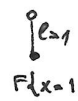
1. pokud $\square \in F$, return 0
2. pokud $F = \emptyset$, return 1
3. pokud $\{l\} \in F$ pro nějaký literál l, vrať DPLL($F\{l=1\}$) // unit propagace
4. pokud l je čistý literál, potom vrať DPLL($F\{l=1\}$)
5. s nějakou strategií vyber proměnnou $x \in Var(F)$
6. pokud DPLL($F\{x=1\}$), vrať 1
7. vrať DPLL($F\{x=0\}$).

Poznámky:

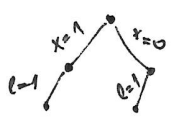
Jedná se o backtracking s úpravou volby proměnné a jejího ohodnocení:

→ řádek 3: Pokud $\{l\} \in F$, je $\{l\}$ tzv. unit klauzule.
Formule $F\{l=0\}$ je kontradikce, obsahuje $\square$ (ta vzniklá z $\{l\}$).

v algoritmu takto: 

máme přímo 

Navíc je výhodné, abychom unit klauzule splnily co nejdříve, protože se tak vyhneme jejich pluením v různých podstromech:



nahradíme: 

[v obecné případě může být rozdíl mnohem výraznější]

→ řádek 4: Literál l je čistý, pokud se $\bar{l}$ ve formuli F nevyskytuje. Je tak bezpečné ohodnotit $l=1$. Opět je výhodné učinit tak co nejdříve (viz komentář k unit klauzulím).

→ řádek 5: Pro výběr proměnné a toho, jestli ji nejdříve ohodnotíme 0 nebo 1 [tj. vybíráme literál] existují různé heuristiky.

Uvedeme několik příkladů; v nich je $f_k(u)$ ... počet výskytů literálu u v klauzulích velikosti k.

$$F(u) = \sum_{k=2}^{n} f_k(u)$$ ... počet výskytů u ve formuli; nepočítáme unit klauzule.

DLIS — literál u s maximálním $F(u)$, ohodnotíme $u = 1$

DLCS — proměnná x s maximálním $f(x) + F(\bar{x})$. Pokud $F(x) \geq F(\bar{x})$, nejdříve ohodnotíme $x=1$, pak $x=0$

MOM: k ... velikost nejmenší klauzule, vybereme proměnnou x, pro kterou je maximální

$$(f_k(x) + f_k(\bar{x})) \cdot p + f_k(x) \cdot f_k(\bar{x}), \text{ kde } p \gg \frac{f_k(x) \cdot f_k(\bar{x})}{f_k(x) + f_k(\bar{x})}.$$

etc. existují další příklady.

Monien - Speckenmeyer

Algoritmus pro k-SAT

Jednoduchá verze:

MS(F)

1. if $\square \in F$, then return 0
2. if $F = \emptyset$, then return 0
3. vyber nejkratší klausuli $C = (l_1 \vee l_2 \vee \dots l_m)$ // $m \le k$ pro k-SAT
4. For $i = 1 .. m$
   if MS($F\{l_1=0, l_2=0, \dots, l_{i-1}=0, l_i=1\}$) = 1, return 1
5. return 0

Lze se dívat jako na speciální verzi DPLL, kde vybíráme k ohodnocení proměnné z krátkých klausulí (a nebereme v úvahu čisté literály).

Pokud má C krátká klausule v F, potom má $C\{l\}$ krátká klausule v $F\{l=0\}$.

Automaticky dělá unit propagaci.

Složitost:

= počet rekurzivních zavolání

$T(n)$ .... počet rekurzivních zavolání pro fci s k proměnnými

z algoritmu vidíme $T(n) \le T(n-1) + T(n-2) + \dots + T(n-m)$, kde m je délka vybrané klausule.

Pokud $m \le k$ a předpokládáme $T(n) = a^n$ pro nějaké $a \ge 1$, potom pro $n = k$ dostaneme

$$a^k = a^{k-1} + \dots + a + 1$$

ze vzorečku $a^{k-1} + a^{k-2} + \dots + a + 1 = \frac{1-a^k}{1-a}$ máme po dosazení $a^{k+1} + 1 = 2a^k$

Řešení této rovnice pro různá k nám dá např.

k	3	4	5
a	1.839	1.928	1.966

Zajímají nás také $b = \log_2 a$, protože $a^n = 2^{b \cdot n}$. Vidíme tak podíl proměnných, jejichž ohodnocení „je vyřešeno"

k	3	4	5
b	0.879	0.94	0.97

~~Zjednodušení~~ Vylepšení pomocí autarkního ohodnocení

def: α je autarkní ohodnocení pro F, pokud pro každé $C \in F$ platí, že $var(\alpha) \cap var(C) \neq \emptyset \Rightarrow$ implikuje $C\alpha = 1$

pozorování: Pro F a autarkní α platí, že F a $F\alpha$ jsou sat-ekvivalentní!

Důkaz: Pokud $(F\alpha)\beta = 1$, potom $F(\alpha \cup \beta) = 1$, protože $VAR(\alpha) \cap VAR(\beta) = \emptyset$ a α splní všechny klauzule, které ohodnotí literál.

Naopak, pokud $F\beta = 1$ pro nějaké β. Potom ustřižení β na proměnné $var(F) - var(\alpha)$ splní klauzule, které neobsahují literál ohodnocený α, tj. splní $F\alpha$ (klauzule, které obsahují literál ohodnocený α jsou α splněny). ∎

MS2(F)

1. if $\square \in F$, return 0
2. if $F = \emptyset$, return 1
3. vyber nejkratší klauzuli $C = \{l_1, l_2, \dots l_m\}$
4. for i in $1..m$
   $\alpha = \{l_1 = 0, l_2 = 0, \dots l_i = 1\}$
   if α je autarkní // lze vždy otestovat
   return MS2($F\alpha$)
5. for i in $1..m$
   if MS2($F\{l_1 = 0, \dots, l_i = 1\}$) then return 1
6. return 0

Autarkní ohodnocení používáme analogicky unit propagaci

Složitost (počet rekurzivních volání)

Musíme vzít v úvahu to, jestli jsme našli autarkní ohodnocení

$T(n) \leq T(n-1)$ // našli jsme autarkní ohodnocení, které ~~splnilo alespoň jednu proměnnou~~ ohodnotilo alespoň jednu proměnnou.

$T(n) = T'(n-1) + \dots + T'(n-k)$ // Tady T' je situace, kdy víme, že F obsahuje klauzuli s nejvíce $k-1$ literály

Máme tedy

$T(n) = \max\{T(n-1), T'(n-1) + \dots + T'(n-k)\}$

Podobné úvahy provedeme pro T' a dostaneme

$$T'(n) = \max\{T(n-1), T'(n-1) + \dots + T'(n-k+1)\}$$

proč $T(n-1)$ a ne $T'(n-1)$ v prvním členu? Protože aplikací ekvivalentního ohodnocení ztratíme klauzuli s $k-1$ literály!

Protože $T(n-1) \le T'(n-1) + \dots + T'(n-k+1)$ máme // to by se dokázalo indukcí přes n

$$T(n) = T'(n-1) + T'(n-k)$$
$$T'(n) = T'(n-1) + T'(n-k+1)$$

Podobně jako u MS předp. $T'(n) = a^n$ a z druhé nerovnosti dostaneme

$$a^{k-1} = a^{k-2} + \dots + a + 1$$

a použitím vlastnosti vzorečku jako předtím

$$a^k + 1 = 2a^{k-1}$$

Dále určíme, že $T(n) = O(T'(n))$ [konstanta v O notaci je $\le k$]

$\leftrightarrow$ Řešením rovnice dostaneme:

k	3	4	5
a	1.618	1.839	1.928
$\log_2 a$	0.694	0.879	0.947

Paturi – Pudlák – Zane algoritmus

PPZ(F)

1. $\alpha = \{\}$
2. Vybereme náhodnou permutaci π množiny $[n]$
3. For i in $1..n$
   - $j = \pi(i)$
   - pokud $\{x_j\} \in F$, pak $\alpha = \alpha \cup \{x_j = 1\}$
   - pokud $\{\bar{x}_j\} \in F$, pak $\alpha = \alpha \cup \{x_j = 0\}$
   - jinak náhodně vybereme $a \in \{0,1\}$ a $\alpha = \alpha \cup \{x_j = a\}$
   - $F = F\alpha$
4. Vrátíme α.

Jedná se o pravděpodobnostní verzi DPLL, kde již pořádí proměnných i teprve ohodnocení, mimo ~~unit resolution~~ unit propagaci, vybírá náhodně.

Algoritmus může vrátit ohodnocení α tak, že $F\alpha = 0$, i v případě, že F je splnitelná. Také v opačném směru chybu nedělá.

Pokud je pravděpodobnost chyby p, pak po provedení $\frac{1}{p} \cdot \epsilon$ opakování algoritmu je pravděpodobnost, že žádné opakování nevrátilo ohodnocení, které splní F, rovno $(1-p)^{\frac{1}{p}\epsilon}$. Protože můžeme omezit $(1-p)^{\frac{1}{p}\epsilon} \leq e^{-\epsilon p}$, abychom dostali pst. chyby menší než e^{-c}, stačí provést $\frac{c}{p}$ opakování.

Věta: Pravděpodobnost toho, že PPZ vrátí pro splnitelnou F ohodnocení α, tak které je splní, je větší než $2^{-n(1-1/k)}$, kde F je v k-CNF a $|Var(F)| = n$.

Složitost algoritmu nalezení s konstantní chybou (viz úvaha výše) je tak $O(2^{n(1-1/k)} \cdot poly(n))$.

Důkaz: Vynecháme.

□

k	3	4	5
$2^{1-1/k}$	1.587	1.682	1.741
$1-1/k$	0.667	0.75	0.8

Pomocí dalšího vylepšení: • ~~to~~ k F doplníme všechny rezolventy klauzulí z F, které mají méně než s literálů, kde $s = o(n/\log n)$ a až poté spustíme algoritmus; dostaneme ~~O(1.308^n)~~ $O(1.308^n)$ pro 3-SAT.

Lokální prohledávání a náhodná procházka (random walk)

Pouze příkladově, hlavně vyučujeme techniky, které vyžadují znalosti pravděpodobnosti.

Hlavní algoritmická myšlenka, pro formuli F:

1. zvolíme počáteční ohodnocení α $[\text{var}(\alpha) = \text{Var}(F)]$
2. Dostatečný počet-krát opakujeme: // toto je dáno např. počtem zahrnutým na $|\text{Var}(F)|$.
   a) pokud $F\alpha = 1$, vrátíme 1
   b) lokálně změníme ohodnocení α tak, aby bylo lepší
3. Vrať 0

Bod 2b) vyžaduje vysvětlení:

- Lokální změnou myslíme změnu v ohodnocení jedné proměnné
- kvalitu ohodnocení můžeme měřit například
  - počtem nesplněných klauzulí
  - Hammingovou vzdáleností k ohodnocení, které F splní (pokud je F splnitelná).
    Hammingova vzdálenost α a α' je počet proměnných, v jejichž ohodnocení se α a α' liší. [Tady je technický výhodné brát ohodnocení jako slova z $\{0,1\}^{|\text{Var}(F)|}$.]

Algoritmus může být tzv. neúplný: pokud je F splnitelná, může se stát, že algoritmus vrátí 0.

Deterministický lokální ~~prohledávání~~ vyhledávání pro 3SAT

Řešíme otázku: Pro formuli F, ohodnocení α a $d \in \mathbb{N}$; existuje ohodnocení α', které splní F a jeho Hammingova vzdálenost od α je nejvýše d?

```
localsearch (F, α, d)
1. if Fα = 1, then return 1
2. if d = 0, then return 0
3. vybereme C ∈ F tak, že Cα = 0, C = {u1, u2, u3}
4. for i in 1..3
       if localsearch (F, α[ui := 1], d-1) = 1, then return 1
5. return 0
```

V kroku 3 k je klíčová myšlenka: pokud lze α' najít, pak se k němu lze přiblížit splněním jednoho z literálů u_1, u_2, u_3.

složitost je dána rekurencí $T(d) = 3 \cdot T(d-1)$, jejímž řešením je 3^d.

Můžeme ovšem provést následující trik:

$\alpha_0 = 0^n$ [tj. ohodnocení, které položí všechny proměnné na 0]

$\alpha_1 = 1^n$ [tj. ohodnocení, které položí všechny proměnné na 1]

$d = n/2$

Každé ohodnocení $\alpha \in \{0,1\}^n$ má vzdálenost od α_1, nebo α_2 nejvýše $n/2$.

Zavoláme tedy

local search $(F, \alpha_1, n/2)$

local search $(F, \alpha_2, n/2)$

a vrátíme 1, pokud jedno z volání vrátilo 1.

Složitost je tak $O^*(3^{n/2}$ ~~polynom~~$) = O(1.733^n) = O(2^{0.793n})$

Algoritmus je navíc úplný: je-li F splnitelná, vrátí 1.

Randomizovaný algoritmus 1

RANDI (F)

1. For i in $1..t$
   - náhodně ohodnocení α
   - if local search $(F, \alpha, d) = 1$, then return 1
2. return 0

Lze ukázat následující:

- algoritmus má pst. chyby omezenou e^{-c}, musí být $t = \frac{c \cdot 2^n}{\sum_{i=1}^{d} \binom{n}{i}}$

- optimální složitost pro je pro $d = n/4$. Pro 3SAT je to $O^*(1.5^n)$.

- Složitosti pro k-SAT

k	3	4	5	...
základ	1.5	1.5	1.667	
$\log_2$ základ	0.585	0.678	0.737	

Náhodná procházka (random-walk)

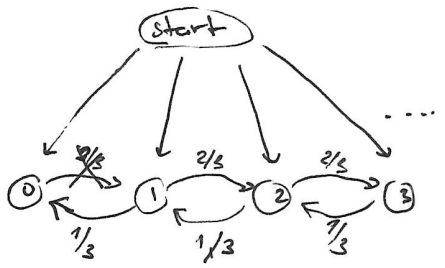
pravděpodobnostní verze local search

1. Výběr náhodné ohodnocení α
2. ~~For jistý počet~~
   For j in $1..n$
   a) IF $F\alpha = 1$, then return 1
   b) výběr $C \in F$, $C\alpha = 0$, $C = (u_1, u_2, u_3)$
   c) náhodný výběr $i \in \{1,2,3\}$
   d) $\alpha = \alpha[u_i := 1]$
3. return 0

tady si d z local-search rovno n, můžeme si to dovolit, protože jedna iterace má polynomiální složitost.

Nechť F je splnitelná. Vyberme α_0 tak, že $F\alpha_0 = 1$ a budeme analyzovat pst, že náhodná procházka najde toto ohodnocení.

Pro každé $C \in F$ máme $C\alpha_0 = 1$, tj. existuje aspoň jeden literál $\ell \in C$ tak, že $\ell\alpha = 1$. Pro každou C pevně vybereme jeden takový literál. Pravděpodobnost, že v kroku 3 vybereme tento literál je $1/3$. V tomto případě řekněme, že se Hammingova vzdálenost aktuálního ohodnocení k α_0 zmenší o 1. Při výběru zbylých dvou literálů budeme předpokládat, že se zvětší (i když to nemusí být pravda).
Máme tak:



čísla v kolečkách jsou vzdálenosti od α_0, pouze odhadnuté (viz odstavec výše)

popisky na hranách jsou pst, odpovídají př. 2c) 2d).

že start přijdeme krokem 1. do stavu j s pstí $\binom{n}{j}2^{-n}$ (binomické rozložení)

Prozkoumáme pst. dvou náhodných jevů

- E_1: v kroku 1 přijdeme do $n/3$: $\quad pst(E_1) = \binom{n}{n/3}2^{-n}$

- E_2: $n/3$ kroků ~~doprava~~ je doprava, $2n/3$ kroků je doleva a skončíme v (0)
  $pst(E_2) = \binom{n}{n/3}\left(\frac{1}{3}\right)^{2n/3}\left(\frac{2}{3}\right)^{n/3}$

$pst(E_1 \wedge E_2) \leq \ldots$ technický výpočet $\leq \left(\frac{3}{4}\right)^n$

Pokud tedy lokálním prohledáváním provedeme $20 \cdot \left(\frac{4}{3}\right)^n$ kroků, omezíme pst. chyby na e^{-20}.