SAT

SAT

- Konjunktivní normální forma (KNF) umožňuje snadný test syntaktické správnosti formule, v obecnosti je obtížné určit splnitelnost (NP-úplný problém)
- V disjunktivní normální formě (DNF) je snadné určit splnitelnost
  - Stačí, aby byl splnitelný alespoň jeden disjunkt
  - Disjunkt (konjunkce literálů) je splnitelný, pokud se v něm žádná proměnná nevyskytuje v pozitivní (nenegované) i negativní (negované) formě
  - Každou formuli je možné převést do DNF
  - Při převodu do DNF formule může narůst exponenciálně, tedy polynomiální řešení formule v DNF je exponenciální vůči původní formuli
- I některé speciální tvary formulí v KNF je možné řešit v polynomiálním čase, např.
  - Horn SAT
  - 2-SAT

Horn SAT

- Formule je Hornova, pokud může být generována gramatikou (s počátečním symbolem H)
  - $P ::= \bot \mid \top \mid p$
  - $A ::= P \mid P \wedge A$
  - $C ::= A \rightarrow P$
  - $H ::= C \mid C \wedge H$
- Příklad: $(p \wedge q \wedge s \rightarrow \bot) \wedge (q \wedge r \rightarrow p) \wedge (p \wedge s \rightarrow s)$
- Po převodu Hornovy formule do KNF je v každé klauzuli právě jeden pozitivní literál (je možno prostřednictvím tohoto faktu Hornovy formule alternativně definovat)

Řešení Horn SAT

- Vstup: Hornova formule Φ (ve tvaru konjunkce implikací)
- Algoritmus:
  - označ všechny výskyty $\top$ v Φ
  - dokud exisuje konjunkt $P_1 \wedge P_2 \wedge \ldots P_k \rightarrow P'$, kde všechny P_i jsou označeny ale P' ne, označ P' (všechny jeho výskyty ve formuli)
  - pokud je označen nějaký výskyt $\bot$, vrať nesplnitelná, jinak vrať splnitelná
- Počet průchodů cyklem je lineární vůči počtu proměnných ve formuli, čas jednoho průchodu cyklem je také lineární vůči velikosti formule, algoritmus je tedy kvadratický
- Korektnost algoritmu - indukcí je možno ukázat, že platí: "Všechna označená P jsou true pro všechny valuace, ve kterých se Φ vyhodnotí na $\top$"

2-SAT

- Formule v KNF se 2 literály v každé klauzuli
- Každou formuli je možné polynomiálně převést do KNF se 3 literály v klauzuli, ale již neexistuje obecný polynomiální algoritmus převodu do KNF se 2 literály
- Každá klauzule instance 2-SAT vlastně odpovídá jednoduché implikaci: $x_0 \vee \neg x_3 \equiv (\neg x_0 \rightarrow \neg x_3) \equiv (x_3 \rightarrow x_0)$
- Je proto možné zadávat formule v implikační normální formě (konjunkce implikací)
- Implikační graf - vrchol pro každý literál a orientované hrany podle implikace v implikační normální formě
- Existuje mnoho algoritmů pro řešení 2-SAT, nejefektivnější z nich jsou lineární

Kromův algoritmus

- Je polynomiální
- Navržen již v roce 1967
- Předpokládejme, že ve Φ jsou dvě klauzule, jedna s x a druhá $\neg x$
- Takové klauzule zkombinujeme a vytvoříme novou (rezoluční pravidlo)
- Z $(a \vee b) \wedge (\neg b \vee \neg c)$ vznikne $a \vee \neg c$
- V implikační formě rezoluční pravidlo odpovídá tranzitivitě
- Konzistentní formule je taková, kde nemůžeme současně vytvořit $(x \vee x)$ a $(\neg x \vee \neg x)$
- Když je formule konzistentní, je možné postupně přidat pro každou proměnnou právě jednu z hodnot $(x \vee x)$ (resp. $(\neg x \rightarrow x)$) nebo $(\neg x \vee \neg x)$ (resp. $(x \rightarrow \neg x)$)tak, aby konzistentní zůstala
- Valuace splňující formuli dává true všem proměnným z klauzulí $(x \vee x)$ a false všem z klauzulí $(\neg x \vee \neg x)$

Kromův algoritmus

- Při návrhu se Krom soustředil na úplnost a správnost, nerozebíral složitost
- Dá se ukázat, že pracuje v polynomiálním čase
- Vezmou se všechny klauzule obsahující stejnou proměnnou a aplikují se všechny možné rezoluce v $O(n^3)$, kde n je počet proměnných (pro každou proměnnou může být $O(n^2)$ klauzulí, které ji obsahují a se kterými lze potenciálně dělat rezoluci)
- $O(n)$ testů konzistentnosti, maximálně $O(n^3)$ každý, tedy $O(n^4)$ celkem
- Při důkladnějším zamyšlení se dá odvodit $O(n^2)$

Omezený backtracking

- Even, Itai, Shamir - 1976
- Obecný algoritmus pro CSP s binárními proměnnými a binárními omezeními
- Idea - přiřazovat hodnoty proměnným postupně, při tzv. choice point vybere hodnotu, při backtrackingu se nikdy nevrátí přes poslední choice point
- Na začátku není žádný choice point (CP) a všechny proměnné jsou bez přiřazené hodnoty
- Pak následují kroky
  - Pokud existuje klauzule s oběma proměnnými nastavenými tak, že je klauzule false, vrátit se zpět na nejbližší CP, odebere přiřazení od tohoto CP a změní hodnotu v tomto CP
  - Pokud existuje klauzule, kde je jedna z proměnných nastavená a klauzule může být true i false, tak druhá klauzule je nastavena tak, aby klauzule byla true
  - Pokud všechny klauzule mají garantováno true nebo mají obě proměnné nepřiřazené, vytvoří se CP a některá nepřiřazená proměnná se nastaví na true nebo false

Omezený backtracking

- Na některých vstupech může být často backtracking a dlouhý řetězec přiřazení mezi návraty, takže nemusí být lineární
- Vylepšení - od CP se dělají paralelně obě větve (pro sekvenční algoritmus prolínáním) a když první větev narazí na nový CP, druhá větev končí
- Po vylepšení je možné dokázat lineární složitost

Silně souvislé komponenty

- Aspvale, Plass, Tarjan - 1979
- Jednodušší procedura v linearním čase založená na silně souvislých komponentách (SCC - strongly connected components)
- SCC - v orientovaném grafu jde o rozklad (množina podmnožin, vzájemně mají prázdný průnik a jejich sjednocením je původní množina) množiny vrcholů tž. každý vrchol v komponentě je dosažitelný z každého dalšího vrcholu stejné komponenty orientovanou cestou
- Existuje několik efektivních algoritmů na hledání SCC v grafu v lineárním čase, většinou jsou založené na hledání do hloubky

Silně souvislé komponenty

- Provede se rozklad na SCC na implikačním grafu (kde pro $x_0 \vee \neg x_3$ je $(\neg x_0 \rightarrow \neg x_3)$ i $(x_3 \rightarrow x_0)$)
- Pokud do stejné komponenty patří x i $\neg x$, tak je formule nesplnitelná
- Autoři ukázali, že toto je nutná, ale také postačující podmínka
- Formule 2-SAT je tedy splnitelná právě tehdy, když neexistuje proměnná patřící do stejné komponenty jako její negace
- Lineární algoritmus rozhodující splnitelnost - v lineárním čase provede rozklad na SCC a poté pro každou proměnnou zkontroluje, ve které komponentě se nachází její negace

Silně souvislé komponenty

- Pro nalezení splňujícího ohodnocení
  - Vytvořit implikační graf, udělat SCC
  - Ověřit splnitelnost, pro nesplnitelné formule skončit
  - Vytvořit kondenzovaný graf - za každou klauzuli se dá jeden vrchol a hrana mezi vrcholy je tam, kde mezi komponentami vedla alespoň jedna hrana. Z vlastností SCC je výsledný kondenzovaný graf acyklický
  - Komponenty se uspořádají topologicky (některé algoritmy pro hledání SCC je rovnou nachází podle tohoto uspořádání)
  - Berou se komponenty podle pořadí, komponentě bez přiřazení je přiřazeno false. Doplňková komponenta (obsahující opačné literály stejných proměnných) tak bude true. Všechny předcházející komponenty už musejí mít přiřazeno také false. Pokud je něco true, všem následujícím komponentám se přiřadí také true.

Aplikace 2-SAT

- Bezkonfliktní umístění objektů - popisky grafu nebo mapy, návrh VLSI (very large scale integration) obvodů
- Data clustering - hledá se minimální poloměr clusterů v metrickém prostoru
- Scheduling - např. přiřazení domácích a hostujících celků po nalosování rozpisu sportovních zápasů tak, aby se omezila po sobě jdoucí utkání stejného týmu na domácím hřišti (resp. na soupeřově hřišti)
- Digitální tomografie - rozpoznání tvarů z jejich průsečíků
- V řešičích SAT jako podprocedura, když po částečném přiřazení a úpravě zůstane jen instance 2-SAT
- ...

Obecný SAT

- Rozšířením algoritmu pro Hornovy formule lze získat algoritmus pro obecné formule
- Podformule se označují značkami true, false tak, že všechny označené podformule se vyhodnotí na svou značku pro každé ohodnocení, které celou formuli vyhodnotí na true.

Lineární řešič

- Formule se převedou do fragmentu $\Phi ::= p \mid (\neg\Phi) \mid (\Phi \wedge \Phi)$
- Vytvoříme k takto převedené formuli syntaktický graf, kde stejné podformule reprezentuje jeden podstrom. Jedná se o orientovaný acyklický graf (directed acyclic graph - DAG)
- Převod formule je induktivní: $T(p) = p$, $T(\neg\Phi) = \neg T(\Phi)$, $T(\Phi_1 \wedge \Phi_2) = T(\Phi_1) \wedge T(\Phi_2)$, $T(\Phi_1 \vee \Phi_2) = \neg(\neg T(\Phi_1) \wedge \neg T(\Phi_2))$, $T(\Phi_1 \rightarrow \Phi_2) = \neg(T(\Phi_1) \wedge \neg T(\Phi_2))$
- Φ je splnitelná právě tehdy, když $\neg\Phi$ je splnitelná
- Příklad syntaktického stromu a odpovídajícího DAG je na obrázku 1.12 v souboru satsolvers.pdf
- Je definovaná sada pravidel pro vynucená nové značky - viz. obrázek 1.14 v souboru satsolvers.pdf
- Jen označení grafu vynucenými značkami nestačí
- Je nutné zkontrolovat přepočítáním zdola nahoru, když se shoduje, máme svědka splnitelnosti
- Příklad dokázání splnitelnosti formule je na obrázku 1.13 v souboru satsolvers.pdf

Lineární řešič

- Tento SAT solver má lineární časovou složitost vzhledem k velikosti DAG pro $T(\Phi)$
- Transformace $\Phi \rightarrow T(\Phi)$ zvětší formuli také lineárně, tedy algoritmus je lineární vůči Φ
- Linearita algoritmu je vykoupena tím, že selže na formulích typu $\neg(\Phi_1 \wedge \Phi_2)$

Kubický řešič

- V lineárním řešiči jsme uvažovali 2 možnosti:
  - Vynucený spor - současně vynucené T i F k jednomu uzlu
  - Úplné ohodnocení všech uzlů
- Existuje třetí možnost - všechna vynucená omezení jsou konzistentní, ale ne všechny uzly jsou ohodnocené
- Příklad: $(p \vee q \vee r) \wedge (p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p) \wedge (\neg p \vee \neg g \vee \neg r)$
- Tato formule není splnitelná - první a poslední klauzule říkají, že alespoň jedna z proměnných p, q, r musí být true a alespoň jedna false. Zbylé klauzule říkají, že všechny tyto 3 proměnné mají stejnou hodnotu.
- Lineární řešič nenajde ani spor ani úplné ohodnocení (viz. obrázek 1.17 ze souboru satsolver.pdf)

Kubický řešič

- Když není označeno vše a není spor, vybere se uzel a přiřadí se mu dočasně T.
- Zkoumáme, zda přiřazením nevznikl spor, pokud ano, uzel dostane místo dočasného T permanentní F.
- Když spor nenastane, zkusí se totéž s dočasnou hodnotou F.
- Všechny uzly, které v obou případech dostaly stejnou hodnotu, budou mít tuto hodnotu jako permanentní.
- Stejně se testují další uzly
- Příklad běhu algoritmu je na obrázku 1.18 v souboru satsolver.pdf
- Pořád se ale může stát, že nepřiřadí všem uzlům nějakou permanentní hodnotu - když každá z voleb T,F vede k různým značením ostatních uzlů, ale ne ke sporu.
- Kubická složitost:
  - Každý test je vlastně použití lineárního řešiče
  - Musíme testovat pro každý nepřiřazený uzel
  - Každé nové permanentní přiřazení způsobí, že ostatní nepřiřazené je potřeba testovat znovu

Kubický řešič

- Pokud řešič vrátí, že je formule nesplnitelná nebo splnitelná, je to vždy správně
- Může se stát, že vrátí odpověď "Nevím"

Další přístupy řešení SATu

- Konflikty řízené učení klauzulí (conflict-driven clause learning) - moderní varianta DPLL algoritmu
- Stochastické lokální hledání - WalkSAT
- DPLL - Systematický backtracking procházející exponenciální prostor, pochází z 60. let
- Moderní řešiče SAT:
  - Conflict-driven
  - Look-ahead

Konflikty řízené řešení SAT

- Davis–Putnam–Logemann–Loveland (DPLL) algoritmus s:
  - efektivní analýzou konfliktů
  - učením klauzulí
  - nechronologickým backtrackingem (backjumpingem)
  - adaptivním větvením
  - náhodnými restarty
- Tato vylepšení se empiricky ukázala důležitá pro zvládání velkých instancí SAT

Look-ahead přístup

- Speciální posílená redukce a heuristiky
- Jsou obecně silnější na těžkých instancích

DPLL

- Vybere literál, přiřadí mu hodnotu, zjednoduší formuli a rekurzivně otestuje splnitelnost
- Když je rekurzivní volání neúspěšné, změní hodnotu a znovu zjednoduší formuli a rekurzivně volá
- Zjednodušení
  - Odstraní klauzule, které jsou true
  - Odstraní literály, které jsou false, z ostatních formulí
- Unit propagation - když klauzule obsahuje 1 literál, tak může být splněna jen jeho nastavením na true. Tedy není nutné větvení. V praxi to často vede ke kaskádě jednotek a z toho plyne velké zmenšení prohledávacího prostoru
- Pure literal elimination - když se proměnná vyskytuje ve formuli pouze v jedné polaritě, nazývá se čistá (pure). Tyto proměnné můžeme nastavit vždy tak, aby všechny klauzule, kde se vyskytují, byly true. Čili tyto klauzule lze odstranit.

DPLL

- Nesplnitelnost vyplyne, když některá klauzule bude prázdná, čili všechny proměnné byly nastaveny tak, že literály v této klauzuli nastavovali na false
- Splnitelnost formule - když přiřadí hodnoty všem proměnným a nedojde ke sporu, čili k prázdné klauzuli, nebo také, když všechny klauzule jsou splněny
- Výzkum pro vylepšení algoritmu:
  - Politika výběru literálů pro větvení
  - Nové datové struktury pro urychlení operací na formuli (hlavně operace unit propagation)
  - Varianty základního backtrackingu (backjumping, clause learning)

Stochastické algoritmy pro SAT

- Nejznámější jsou GSAT a WalkSAT
- Oba jsou určeny na formule v KNF
- Přiřadí náhodné ohodnocení
- Když je formule splněna, končí
- Když není splněna, přehodí hodnotu jedné proměnné a opakují postup
- Stochastické algoritmy jsou neúplné - u splnitelné formule najdou často řešení, ale neumí ukázat, že je formule nesplnitelná
- Po nějaké době hledaní se obvykle aplikuje restart - znovu náhodně přiřadí hodnotu všem proměnným a běží na této nové instanci
- Intervaly mezi restarty se často postupně zvětšují

WalkSAT

- Vybere některou nesplněnou klauzuli a v ní přehodí proměnnou
- Klauzule volí náhodně
- Proměnnou v klauzuli volí tak, aby minimální počet splněných klauzulí změnil na nesplněné
- S menší pravděpodobností i proměnnou volí náhodně
- Ukázal se vhodný při instancích vzniklých z automatického plánování (přístup převedení plánování na SAT se nazývá SATplan)

GSAT

- Náhodně přiřadí hodnoty proměnných
- Vybere proměnnou, která maximálně sníží počet nesplněných klauzulí a změní její hodnotu
- Problémem jsou lokální minima - v jiných problémech může nalezení lokálního minima stačit, v SATu ne
- "Krok stranou" (změna hodnoty jiné proměnné, než by určil algoritmus) značně vylepší úspěšnost
- Mezi přístupy k opuštění lokálních minim se často uplatňuje simulované žíhání