Moderné SAT-solvery ako najefektívnejší nástroj na riešenie kombinatorických problémov

Ján Mazák

FMFI UK Bratislava

23. 9. 2019

- teoretická vs. praktická zložitosť
- problém splniteľnosti a aplikácie
- verifikácia hardvéru
- algoritmy v moderných SAT solveroch
- využitie v kombinatorike

Ako merať zložitosť výpočtov — teória

- zložitosť je počet vykonaných krokov ako funkcia veľkosti vstupu n (nezávislé od hardvéru)

Ako merať zložitosť výpočtov — teória

- zložitosť je počet vykonaných krokov ako funkcia veľkosti vstupu n (nezávislé od hardvéru)
- zložitosť problému vs. konkrétneho algoritmu

Ako merať zložitosť výpočtov — teória

- zložitosť je počet vykonaných krokov ako funkcia veľkosti vstupu n (nezávislé od hardvéru)
- zložitosť problému vs. konkrétneho algoritmu
- pri porovnávaní uvažujeme n idúce do nekonečna

Ako merať zložitosť výpočtov — teória

- od cca 1970 delíme problémy na „ľahké“ (existuje polynomiálny algoritmus, trieda P) a „ťažké“ (nik nepozná polynomiálny algoritmus, triedy NP, PSPACE atď.)
- zvláštny dôraz na triedu NP: problémy, kde vieme v polynomiálnom čase overiť riešenie

Ako merať zložitosť výpočtov — teória

- od cca 1970 delíme problémy na „ľahké“ (existuje polynomiálny algoritmus, trieda P) a „ťažké“ (nik nepozná polynomiálny algoritmus, triedy NP, PSPACE atď.)
- zvláštny dôraz na triedu NP: problémy, kde vieme v polynomiálnom čase overiť riešenie
- napriek rozsiahlemu úsiliu nevieme ani len to, či $P \neq NP$
- ale vieme, že existujú nerozhodnuteľné problémy

Zložitosť výpočtov — prax

- asymptotické porovnávanie ignoruje konštanty
- mnohé teoreticky najlepšie algoritmy nemá význam implementovať, pretože veľkosť vstupu je obmedzená pamäťou

Zložitosť výpočtov — prax

- asymptotické porovnávanie ignoruje konštanty
- mnohé teoreticky najlepšie algoritmy nemá význam implementovať, pretože veľkosť vstupu je obmedzená pamäťou
- na hardvéri záleží, často viac ako na algoritme
- špecializovaný hardvér, ak je to ekonomicky atraktívne (ťažba kryptomien)

Zložitosť výpočtov — prax

- asymptotické porovnávanie ignoruje konštanty
- mnohé teoreticky najlepšie algoritmy nemá význam implementovať, pretože veľkosť vstupu je obmedzená pamäťou
- na hardvéri záleží, často viac ako na algoritme
- špecializovaný hardvér, ak je to ekonomicky atraktívne (ťažba kryptomien)
- aj keď NP-úplné problémy sú teoreticky ekvivalentné, v praxi sa neriešia rovnako rýchlo (hranové farbenie celými vs. racionálnymi číslami)

Zložitosť výpočtov — prax

- asymptotické porovnávanie ignoruje konštanty
- mnohé teoreticky najlepšie algoritmy nemá význam implementovať, pretože veľkosť vstupu je obmedzená pamäťou
- na hardvéri záleží, často viac ako na algoritme
- špecializovaný hardvér, ak je to ekonomicky atraktívne (ťažba kryptomien)
- aj keď NP-úplné problémy sú teoreticky ekvivalentné, v praxi sa neriešia rovnako rýchlo (hranové farbenie celými vs. racionálnymi číslami)
- strojový čas je lacnejší ako ľudský

Zložitosť výpočtov — prax

- zdanlivo niet dôvod preferovať ľubovoľný polynóm pred exponenciálnou funkciou (n^{100} je horšie ako 2^n)
- ale takmer vždy ak nájdeme *nejaký* polynomiálny algoritmus, do pár rokov máme aj *dostatočne rýchly* polynomiálny algoritmus

Zložitosť výpočtov — prax

- algoritmy s nevábnou teoretickou zložitosťou môžu dobre fungovať v praxi, najmä ak sa podarí nájsť dobrú heuristiku
- klasická teória zložitosti neprihliada na nerovnomerné zastúpenie praktických inštancií

Zložitosť výpočtov — prax

- algoritmy s nevábnou teoretickou zložitosťou môžu dobre fungovať v praxi, najmä ak sa podarí nájsť dobrú heuristiku
- klasická teória zložitosti neprihliada na nerovnomerné zastúpenie praktických inštancií
- napr. pre problém splniteľnosti sú spracované vstupy často 10- až 100-krát väčšie než by zodpovedalo teoretickej zložitosti

Problém splniteľnosti

- rozhodnutie, či je daná boolovská formula splniteľná, čiže či možno ohodnotiť jednotlivé premenné tak, aby bola pravdivá
- prvý problém, pre ktorý bola dokázaná NP-úplnosť
- teoreticky najlepšie algoritmy cca 1.3^n
- v praxi riešiteľné pre formuly s tisíckami až miliónmi premenných

Súťaže

- výrazný pokrok v rokoch 1996–2001, keď sa SAT solvery stali dostatočne rýchle pre praktické využitie
- od r. 2002 každoročne SAT Competition
- o.i. kategória „Glucose hack“ — modifikácia existujúceho solvera nesmie presiahnuť 1000 znakov
- desiatky SAT solverov s otvoreným zdrojovým kódom
- 2013+ SAT configuration competition: pre obmedzený okruh vstupov možno dosiahnuť zrýchlenie typicky 2-10x (4.5x pre verifikáciu hardvéru)

Aplikácie problému splniteľnosti

- verifikácia hardvéru (procesor i7 od Intelu)
- verifikácia softvéru (ovládače vo Windows 7)
- manažment závislostí (pluginy v Eclipse)
- konfigurácia produktu pre zákazníka (Daimler)
- expertné systémy, letová kontrola, kryptológia atď.

Aplikácie problému splniteľnosti

- verifikácia hardvéru je azda najvýznamnejšia oblasť využitia — bez moderných procesorov nevieme robiť žiadne iné výpočty
- softvér sa vymení ľahko, vymieňať hardvér je prakticky nemožné alebo neekonomické; nedá sa opraviť časť procesora
- pri desiatkach miliónov tranzistorov nemáme inú dostatočne výkonnú alternatívu

Metódy verifikácie hardvéru a softvéru

1. Simulácia

Metódy verifikácie hardvéru a softvéru

1. Simulácia
   - užitočná, ale nič nezaručuje
   - je ťažké až nemožné zachytiť všetky možné stavy, v ktorých sa má systém používať

Metódy verifikácie hardvéru a softvéru

1. Simulácia
   - užitočná, ale nič nezaručuje
   - je ťažké až nemožné zachytiť všetky možné stavy, v ktorých sa má systém používať
2. Formálna verifikácia
   - v princípe úplný matematický dôkaz správnosti

Metódy verifikácie hardvéru a softvéru

1. Simulácia
   - užitočná, ale nič nezaručuje
   - je ťažké až nemožné zachytiť všetky možné stavy, v ktorých sa má systém používať
2. Formálna verifikácia
   - v princípe úplný matematický dôkaz správnosti
   - nedá sa použiť pre fyzickú vrstvu, ale ideálna pre logickú

Metódy verifikácie hardvéru a softvéru

1. Simulácia
   - užitočná, ale nič nezaručuje
   - je ťažké až nemožné zachytiť všetky možné stavy, v ktorých sa má systém používať
2. Formálna verifikácia
   - v princípe úplný matematický dôkaz správnosti
   - nedá sa použiť pre fyzickú vrstvu, ale ideálna pre logickú
   - používa sa zriedka — drahá a vyžaduje vysokú odbornosť
   - atómové elektrárne, vesmírne lety, veľké série procesorov

Logická vrstva

AND: $a \wedge b, \quad a \cdot b$

a	b	ab
0	0	0
0	1	0
1	0	0
1	1	1

Logická vrstva

AND: $a \wedge b, \quad a \cdot b$

a	b	ab
0	0	0
0	1	0
1	0	0
1	1	1

OR: $a \vee b, \quad a + b$

a	b	$a + b$
0	0	0
0	1	1
1	0	1
1	1	1

Logická vrstva

AND: $a \wedge b, \quad a \cdot b$

a	b	ab
0	0	0
0	1	0
1	0	0
1	1	1

OR: $a \vee b, \quad a + b$

a	b	$a + b$
0	0	0
0	1	1
1	0	1
1	1	1

- máme tiež NOT, NOR, NAND, XOR, ...
- hradlové obvody počítajú *boolovské funkcie*, napr. $a \vee (b \wedge c)$

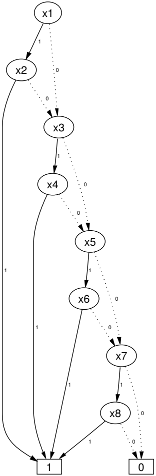
Verifikácia hardvéru

Príklady úloh:

- dôkaz ekvivalencie výpočtových okruhov (napr. po optimalizácii)
- kontrola zachovania invariantu
- *safety*: môže systém dosiahnuť daný stav?
- *liveness*: dosiahne sa stav T vždy po dosiahnutí S?

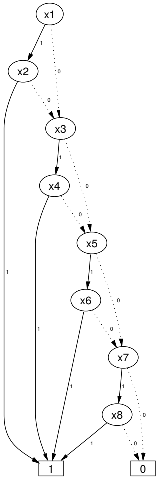
Binary decision diagrams (BDDs)

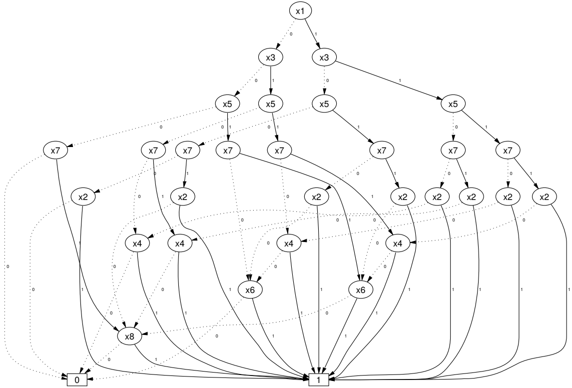
$$f(x_1, \ldots, x_8) = x_1x_2 + x_3x_4 + x_5x_6 + x_7x_8$$



Binary decision diagrams (BDDs)

$$f(x_1, \ldots, x_8) = x_1 x_2 + x_3 x_4 + x_5 x_6 + x_7 x_8$$





Verifikácia: BDDs

- logický jazyk sa dá doplniť o nové prvky, ktoré trebárs umožňujú vyjadriť, že niečo nastane v budúcnosti (napr. dôjde k prideleniu požadovaného prostriedku procesu)

Verifikácia: BDDs

- logický jazyk sa dá doplniť o nové prvky, ktoré trebárs umožňujú vyjadriť, že niečo nastane v budúcnosti (napr. dôjde k prideleniu požadovaného prostriedku procesu)
- používané desiatky rokov

Verifikácia: BDDs

- logický jazyk sa dá doplniť o nové prvky, ktoré trebárs umožňujú vyjadriť, že niečo nastane v budúcnosti (napr. dôjde k prideleniu požadovaného prostriedku procesu)
- používané desiatky rokov
- nevýhody:
  - diagramy môžu byť exponenciálne veľké, zvyšuje pamäťovú aj časovú náročnosť

Verifikácia: BDDs

- logický jazyk sa dá doplniť o nové prvky, ktoré trebárs umožňujú vyjadriť, že niečo nastane v budúcnosti (napr. dôjde k prideleniu požadovaného prostriedku procesu)
- používané desiatky rokov
- nevýhody:
  - diagramy môžu byť exponenciálne veľké, zvyšuje pamäťovú aj časovú náročnosť
  - poradie premenných musí byť vo všetkých vetvách rovnaké
  - veľmi záleží na poradí premenných

Alternatíva: bounded model checking (BMC)

Redukuje vyššie uvedené problémy na overenie (ne)splniteľnosti boolovskej formuly. [Biere et al. 1999]

Alternatíva: bounded model checking (BMC)

Redukuje vyššie uvedené problémy na overenie (ne)splniteľnosti boolovskej formuly. [Biere et al. 1999]

- porovnanie okruhov: $f_1 \Leftrightarrow f_2 \ldots (f_1 \wedge f_2) \vee (\neg f_1 \wedge \neg f_2)$

Alternatíva: bounded model checking (BMC)

Redukuje vyššie uvedené problémy na overenie (ne)splniteľnosti boolovskej formuly. [Biere et al. 1999]

- porovnanie okruhov: $f_1 \Leftrightarrow f_2 \ldots (f_1 \wedge f_2) \vee (\neg f_1 \wedge \neg f_2)$
- kontrola splnenia podmienky: popíšeme formulou, že v k-tom kroku výpočtu došlo k porušeniu, a postupne zvyšujeme k

Alternatíva: bounded model checking (BMC)

Redukuje vyššie uvedené problémy na overenie (ne)splniteľnosti boolovskej formuly. [Biere et al. 1999]

- porovnanie okruhov: $f_1 \Leftrightarrow f_2 \ldots (f_1 \wedge f_2) \vee (\neg f_1 \wedge \neg f_2)$
- kontrola splnenia podmienky: popíšeme formulou, že v k-tom kroku výpočtu došlo k porušeniu, a postupne zvyšujeme k

Prepíšeme formulu do *konjunktívnej normálnej formy* (CNF) a overíme SAT solverom.

Alternatíva: bounded model checking (BMC)

Redukuje vyššie uvedené problémy na overenie (ne)splniteľnosti boolovskej formuly. [Biere et al. 1999]

- porovnanie okruhov: $f_1 \Leftrightarrow f_2 \ldots (f_1 \wedge f_2) \vee (\neg f_1 \wedge \neg f_2)$
- kontrola splnenia podmienky: popíšeme formulou, že v k-tom kroku výpočtu došlo k porušeniu, a postupne zvyšujeme k

Prepíšeme formulu do *konjunktívnej normálnej formy* (CNF) a overíme SAT solverom.
Riešiteľné inštancie: 0.4 mil. premenných, 7 mil. klauzúl [2004].

Alternatíva: bounded model checking (BMC)

Redukuje vyššie uvedené problémy na overenie (ne)splniteľnosti boolovskej formuly. [Biere et al. 1999]

- porovnanie okruhov: $f_1 \Leftrightarrow f_2 \ldots (f_1 \wedge f_2) \vee (\neg f_1 \wedge \neg f_2)$
- kontrola splnenia podmienky: popíšeme formulou, že v k-tom kroku výpočtu došlo k porušeniu, a postupne zvyšujeme k

Prepíšeme formulu do *konjunktívnej normálnej formy* (CNF) a overíme SAT solverom.
Riešiteľné inštancie: 0.4 mil. premenných, 7 mil. klauzúl [2004].
Prehľadanie priestor veľkosti 2^{50000} trvá sekundy [IBM 2011].

Ako testovať splniteľnosť formuly v CNF

Máme formulu s n premennými.

Ako testovať splniteľnosť formuly v CNF

Máme formulu s n premennými.

- hrubou silou: 2^n možností

Ako testovať splniteľnosť formuly v CNF

Máme formulu s n premennými.

- hrubou silou: 2^n možností
- *backtracking*: prehľadáme celý strom (stále exponenciálne)

Ako testovať splniteľnosť formuly v CNF

Máme formulu s n premennými.

- hrubou silou: 2^n možností
- *backtracking*: prehľadáme celý strom (stále exponenciálne)
- DPLL (1960)

Ako testovať splniteľnosť formuly v CNF

Máme formulu s n premennými.

- hrubou silou: 2^n možností
- *backtracking*: prehľadáme celý strom (stále exponenciálne)
- DPLL (1960)
- conflict-driven clause learning — CDCL (1996)

Ako testovať splniteľnosť formuly v CNF

Máme formulu s n premennými.

- hrubou silou: 2^n možností
- *backtracking*: prehľadáme celý strom (stále exponenciálne)
- DPLL (1960)
- conflict-driven clause learning — CDCL (1996)
- CDCL vylepšené o heuristiku VSIDS (2001)

Ako testovať splniteľnosť formuly v CNF

Máme formulu s n premennými.

- hrubou silou: 2^n možností
- *backtracking*: prehľadáme celý strom (stále exponenciálne)
- DPLL (1960)
- conflict-driven clause learning — CDCL (1996)
- CDCL vylepšené o heuristiku VSIDS (2001)
- kombinácia VSIDS a strojového učenia (Maple 2016+)

DPLL

základný backtracking:

1. zvoľ premennú
2. zvoľ ohodnotenie
3. odstráň splnené klauzuly a nepravdivé literály
4. ak splniteľná, koniec
5. ak nesplnená, späť k 2.
6. inak opäť 1. s inou premennou

DPLL

základný backtracking:

1. zvoľ premennú
2. zvoľ ohodnotenie
3. odstráň splnené klauzuly a nepravdivé literály
4. ak splniteľná, koniec
5. ak nesplnená, späť k 2.
6. inak opäť 1. s inou premennou

vylepšenia v DPLL:

unit clause propagation: ak je v klauzule len jeden literál, vieme, ako ho ohodnotiť, aplikujeme vždy po kroku 3.; ak vznikne prázdna klauzula, je to dôsledok poslednej voľby v kroku 2.
— *chronologický* backtracking

DPLL

základný backtracking:

1. zvoľ premennú
2. zvoľ ohodnotenie
3. odstráň splnené klauzuly a nepravdivé literály
4. ak splniteľná, koniec
5. ak nesplnená, späť k 2.
6. inak opäť 1. s inou premennou

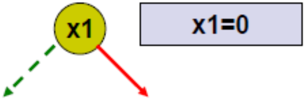
vylepšenia v DPLL:

unit clause propagation: ak je v klauzule len jeden literál, vieme, ako ho ohodnotiť, aplikujeme vždy po kroku 3.; ak vznikne prázdna klauzula, je to dôsledok poslednej voľby v kroku 2.
— *chronologický* backtracking
pure literal elimination: ak sa nevyskytuje negácia literálu, možno ho ohodnotiť

CDCL — conflict-driven clause learning

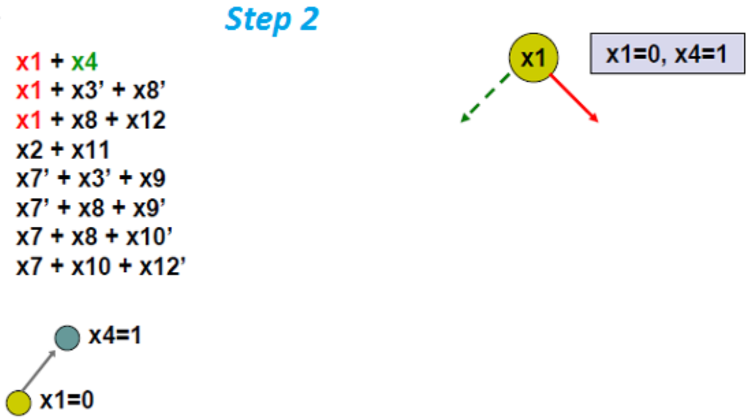
Step 1

x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’

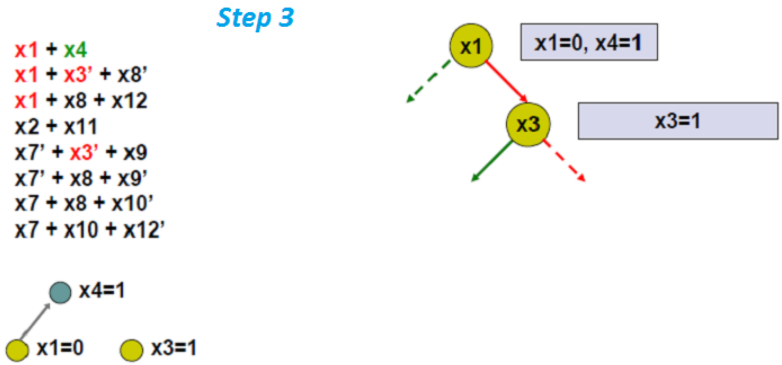


x1=0

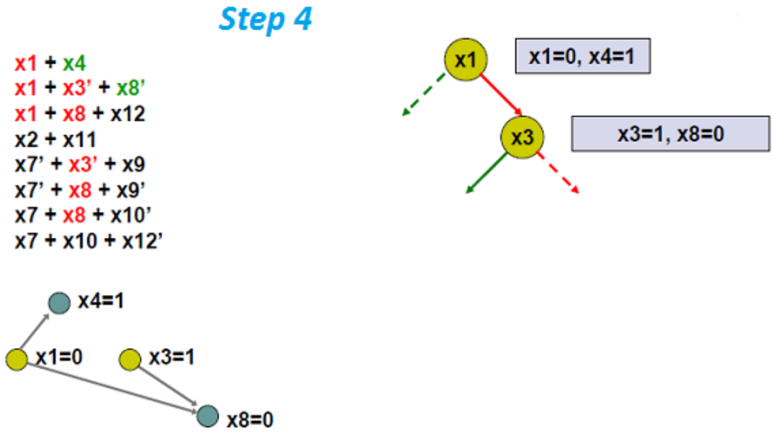
CDCL — conflict-driven clause learning



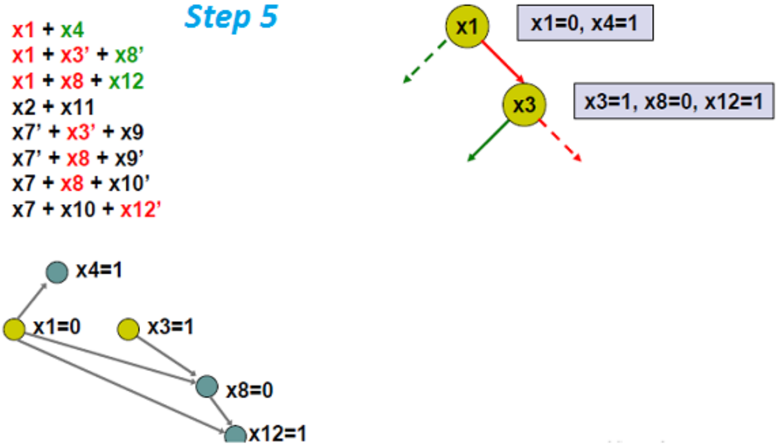
CDCL — conflict-driven clause learning



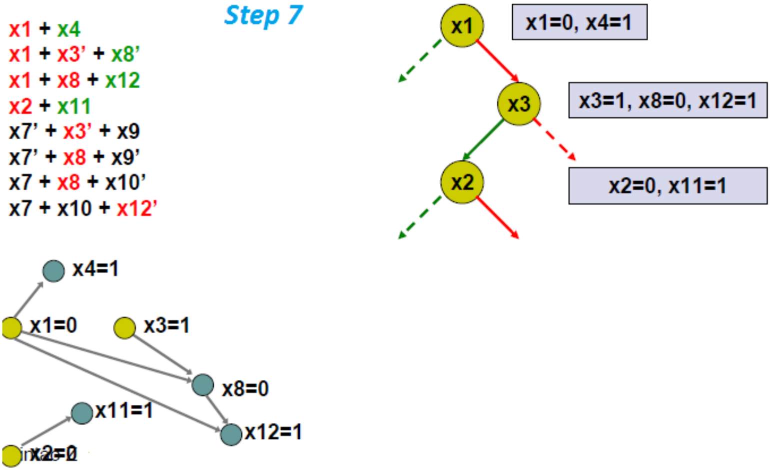
CDCL — conflict-driven clause learning



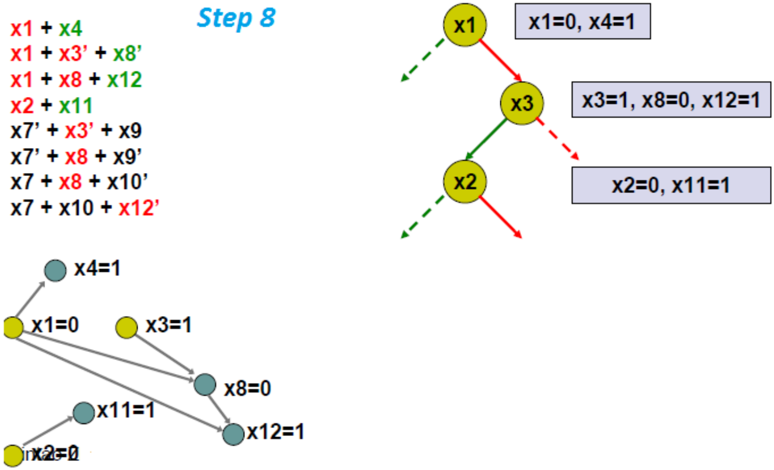
CDCL — conflict-driven clause learning



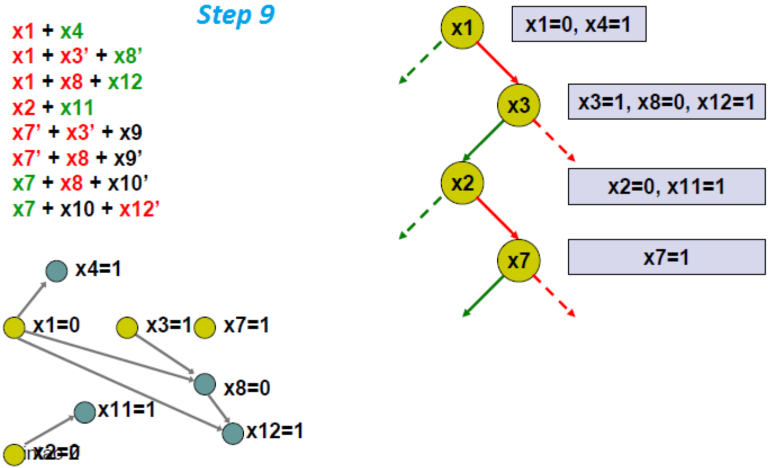
CDCL — conflict-driven clause learning



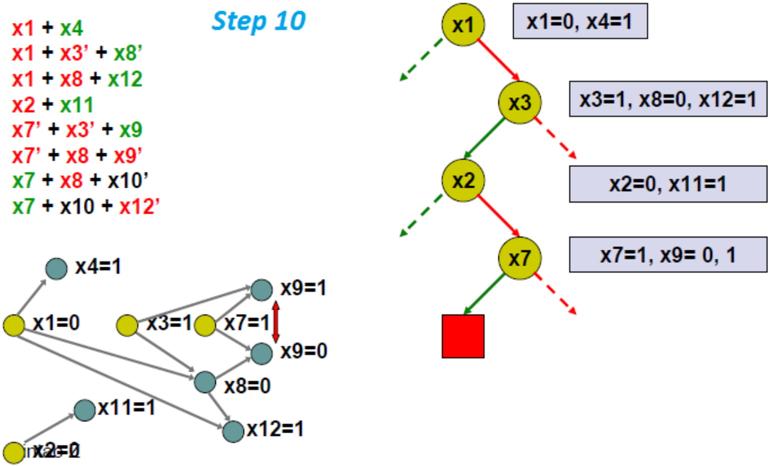
CDCL — conflict-driven clause learning



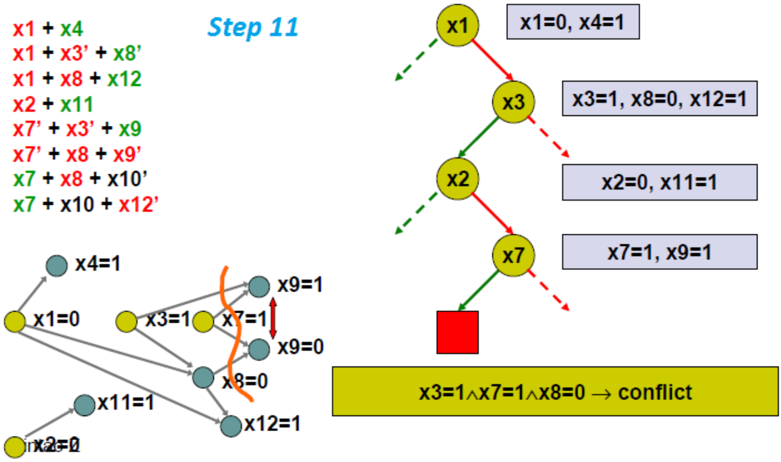
CDCL — conflict-driven clause learning



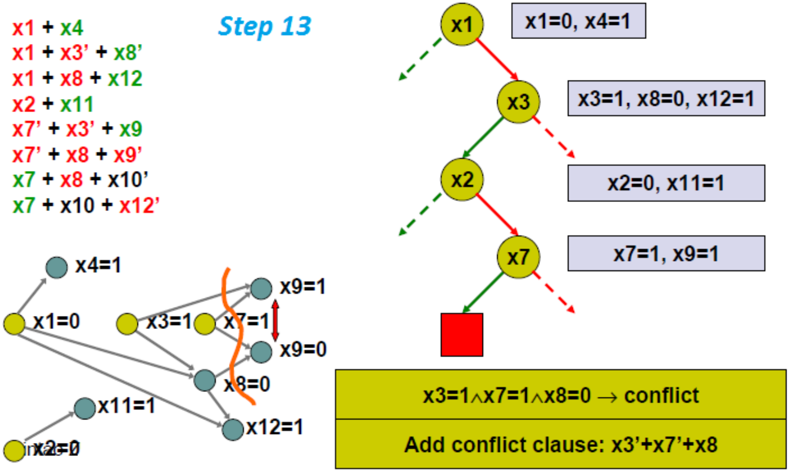
CDCL — conflict-driven clause learning



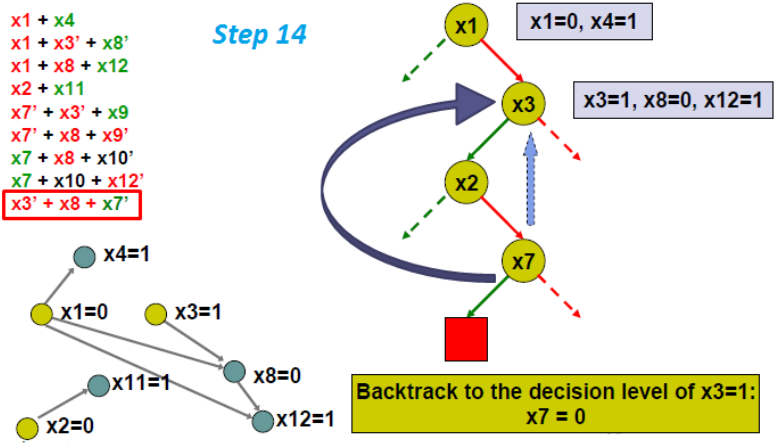
CDCL — conflict-driven clause learning



CDCL — conflict-driven clause learning



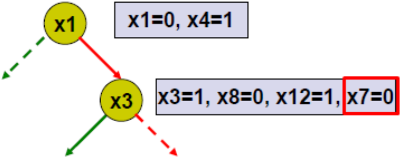
CDCL — conflict-driven clause learning

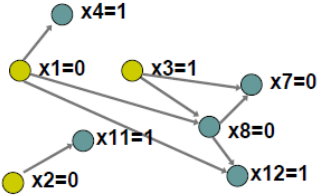


CDCL — conflict-driven clause learning

x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’
x3’ + x8 + x7’

Step 15





CDCL — conflict-driven clause learning

- vytvárame implikačný graf

CDCL — conflict-driven clause learning

- vytvárame implikačný graf
- ak nájdeme konflikt, zvolíme rez medzi rozhodnutiami a konfliktmi; z rezu odvodíme novú klauzulu — čosi sme sa *naučili*

CDCL — conflict-driven clause learning

- vytvárame implikačný graf
- ak nájdeme konflikt, zvolíme rez medzi rozhodnutiami a konfliktmi; z rezu odvodíme novú klauzulu — čosi sme sa *naučili*
- vrátime sa nie chronologicky, ale do miesta, kde sme volili predposlednú hodnotu pre premennú v nájdenom konflikte

CDCL — conflict-driven clause learning

- existuje exponenciálne veľa odvoditeľných klauzúl; ktoré a kedy zahodiť?
- v 2009 zvíťazil Glucose s novým agresívnym prístupom k zahadzovaniu

CDCL — conflict-driven clause learning

- existuje exponenciálne veľa odvoditeľných klauzúl; ktoré a kedy zahodiť?
- v 2009 zvíťazil Glucose s novým agresívnym prístupom k zahadzovaniu
- čas výpočtu má distribúciu s ťažkými chvostmi (občas trvá výpočet extrémne dlho)
- riešenie: občasné reštartovanie (mnoho metód, niektoré založené na hlbokej štatistickej analýze náhodných procesov)

Ako zvoliť nasledujúcu premennú a ohodnotenie?

- voľba premennej a ohodnotenia: potenciálne veľký priestor na zlepšenie

Ako zvoliť nasledujúcu premennú a ohodnotenie?

- voľba premennej a ohodnotenia: potenciálne veľký priestor na zlepšenie
- heuristika VSIDS: additive bumping, multiplicative decay
- počíta pre každý literál, koľkokrát sa zjavil v „naučených“ klauzulách
- periodicky predelí skóre konštantou (dôraz na novoobjavené klauzuly)

Ako zvoliť nasledujúcu premennú a ohodnotenie?

- voľba premennej a ohodnotenia: potenciálne veľký priestor na zlepšenie
- heuristika VSIDS: additive bumping, multiplicative decay
- počíta pre každý literál, koľkokrát sa zjavil v „naučených“ klauzulách
- periodicky predelí skóre konštantou (dôraz na novoobjavené klauzuly)
- jednoduchá, ale veľmi efektívna; väčšina súčasných solverov využíva nejaký variant VSIDS

Ako zvoliť nasledujúcu premennú a ohodnotenie?

- voľba premennej a ohodnotenia: potenciálne veľký priestor na zlepšenie
- heuristika VSIDS: additive bumping, multiplicative decay
- počíta pre každý literál, koľkokrát sa zjavil v „naučených“ klauzulách
- periodicky predelí skóre konštantou (dôraz na novoobjavené klauzuly)
- jednoduchá, ale veľmi efektívna; väčšina súčasných solverov využíva nejaký variant VSIDS
- heuristika LRB [Maple 2016]: vychádza z učenia so spätnou väzbou (multi-armed bandit problem); odhaduje, aké časté budú konflikty pre danú premennú a vyberá tú, kde ich očakáva najviac
- striedavé použitie VSIDS a LRB

Ďalšie vylepšenia

- drobné vylepšenia majú veľký vplyv na efektivitu v praxi

Ďalšie vylepšenia

- drobné vylepšenia majú veľký vplyv na efektivitu v praxi
- väčšinu času výpočtu SAT solvera tvorí unit propagation (keď vznikne klauzula s jedinou premennou, je jasné, ako má byť ohodnotená, a toto ohodnotenie môžeme dosadiť do všetkých ostatných klauzúl)
- watched literals [2001]: pre každú zatiaľ nesplnenú klauzulu evidujeme dva neohodnotené literály; keď priradíme literálu x pravdivú hodnotu, pozrieme sa na len na klauzuly, kde $\overline{x}$ je evidovaný literál

Ďalšie vylepšenia

- drobné vylepšenia majú veľký vplyv na efektivitu v praxi
- väčšinu času výpočtu SAT solvera tvorí unit propagation (keď vznikne klauzula s jedinou premennou, je jasné, ako má byť ohodnotená, a toto ohodnotenie môžeme dosadiť do všetkých ostatných klauzúl)
- watched literals [2001]: pre každú zatiaľ nesplnenú klauzulu evidujeme dva neohodnotené literály; keď priradíme literálu x pravdivú hodnotu, pozrieme sa na len na klauzuly, kde $\overline{x}$ je evidovaný literál
- súčasné solvery sú ako programy zložité — dátové štruktúry musia zvládnuť návrat pri backtrackingu a informácia o redukovanej formule je neúplná

Predspracovanie

- všetky moderné SAT solvery venujú značnú pozornosť predspracovaniu formuly
- počet premenných je zvyčajne podstatnejší ako veľkosť formuly

Predspracovanie

- všetky moderné SAT solvery venujú značnú pozornosť predspracovaniu formuly
- počet premenných je zvyčajne podstatnejší ako veľkosť formuly
- rezolvenciou možno znížiť počet klauzúl (ale narastie ich veľkosť)

Predspracovanie

- všetky moderné SAT solvery venujú značnú pozornosť predspracovaniu formuly
- počet premenných je zvyčajne podstatnejší ako veľkosť formuly
- rezolvenciou možno znížiť počet klauzúl (ale narastie ich veľkosť)
- rezolvenciou možno znížiť počet premenných (ale výrazne narastie počet klauzúl)

Predspracovanie

- všetky moderné SAT solvery venujú značnú pozornosť predspracovaniu formuly
- počet premenných je zvyčajne podstatnejší ako veľkosť formuly
- rezolvenciou možno znížiť počet klauzúl (ale narastie ich veľkosť)
- rezolvenciou možno znížiť počet premenných (ale výrazne narastie počet klauzúl)
- poradie klauzúl zvyčajne nemá zásadný vplyv na dĺžku výpočtu
- redundantné klauzuly môžu pomôcť

Predspracovanie

- desiatky rôznych techník, často doménovo špecifických
- napr. neúplné BDD reprezentácie umožňujú získať klauzuly, ktoré nemožno odvodiť počas CDCL

Predspracovanie

- desiatky rôznych techník, často doménovo špecifických
- napr. neúplné BDD reprezentácie umožňujú získať klauzuly, ktoré nemožno odvodiť počas CDCL
- cryptominisat akceptuje XOR-klauzuly a pri predspracovaní sa na ne díva ako na sústavu lineárnych rovníc nad $\mathbb{Z}_2$ a používa Gaussovu elimináciu

Predspracovanie

- desiatky rôznych techník, často doménovo špecifických
- napr. neúplné BDD reprezentácie umožňujú získať klauzuly, ktoré nemožno odvodiť počas CDCL
- cryptominisat akceptuje XOR-klauzuly a pri predspracovaní sa na ne díva ako na sústavu lineárnych rovníc nad $\mathbb{Z}_2$ a používa Gaussovu elimináciu
- pri „ľahkých“ inštanciách môže predspracovanie zabrať viac času než následné riešenie, treba nájsť vhodný kompromis
- v niektorých prípadoch zase predspracovanie zvyšuje dobu následného riešenia

Neúplné riešenie

- *incomplete* solver negarantuje detekciu nesplniteľnosti v konečnom čase

Neúplné riešenie

- *incomplete* solver negarantuje detekciu nesplniteľnosti v konečnom čase
- rôzne solvery založené na náhodných prechádzkach, genetických algoritmoch, simulovanom žíhaní či heuristikách využívaných v štatistickej fyzike, napr. survey propagation má výborné výsledky pre 3-SAT (SAT vykazuje *threshold* a *clustering phenomenon*)

Neúplné riešenie

- *incomplete* solver negarantuje detekciu nesplniteľnosti v konečnom čase
- rôzne solvery založené na náhodných prechádzkach, genetických algoritmoch, simulovanom žíhaní či heuristikách využívaných v štatistickej fyzike, napr. survey propagation má výborné výsledky pre 3-SAT (SAT vykazuje *threshold* a *clustering phenomenon*)
- v automatizovanom plánovaní je efektívna kombinácia neúplného a úplného solvera

Ďalšie aspekty

- existujúce CDCL solvery nie sú veľmi paralelizovateľné: niektoré bežia výlučne v jedinom vlákne, iné vedia využiť aj 24 vlákien, lenže na výkone to veľmi nevidno (ak chceme vyriešiť viac inštancií, viac sa oplatí riešiť každú v osobitnom vlákne)
- využitie strojového učenia na voľbu vhodného solvera, resp. jeho konfigurácie (lingeling: 300 parametrov)

Formulácia vstupu

- pre niektoré problémy je prirodzené vytvoriť disjunktívnu normálnu formu
- ekvivalentná CNF je exponenciálne veľká

Formulácia vstupu

- pre niektoré problémy je prirodzené vytvoriť disjunktívnu normálnu formu
- ekvivalentná CNF je exponenciálne veľká
- možno však vytvoriť formulu, ktorá je *equisatisfiable*:

$$\bigvee_i (a_i \wedge b_i \wedge c_i)$$

$$\left(\bigvee_i z_i\right) \wedge \bigwedge_i \left[(\overline{z_i} \vee a_i) \wedge (\overline{z_i} \vee b_i) \wedge (\overline{z_i} \vee c_i)\right]$$

- vo všeobecnosti možno očakávať polynomiálnu veľkosť formuly

Kombinatorické problémy

- pre viaceré kombinatorické problémy je redukcia na SAT a využitie existujúceho solvera najlepšie, čo možno v súčasnosti spraviť (často je to jediná realistická možnosť)
- napr. hľadanie najmenšieho k-chromatického grafu s daným obvodom [Goedgebeur 2018] či určovanie cirkulárneho chromatického indexu [Kunertová 2017]
- nie vždy: napr. pre problém obchodného cestujúceho existujú špecializované solvery s efektívnymi heuristikami

Hranové 3-farbenie 3-regulárneho grafu

- premenné $x_{e,c}$
- každá hrana má aspoň jednu farbu $\bigwedge_{e \in E} (x_{e,1} \vee x_{e,2} \vee x_{e,3})$
- každá hrana má najviac jednu farbu
  $$\bigwedge_{e \in E} \left((\overline{x_{e,1}} \vee \overline{x_{e,2}}) \wedge (\overline{x_{e,2}} \vee \overline{x_{e,3}}) \wedge (\overline{x_{e,3}} \vee \overline{x_{e,1}}) \right)$$

Hranové 3-farbenie 3-regulárneho grafu

Regularita farbenia:

1. susedné hrany majú rôznu farbu
2. v každom vrchole je každá farba použitá práve raz
3. farby v danom vrchole vytvárajú prípustnú trojicu
4. formula vytvorená z nullstellensatz

Hranové 3-farbenie 3-regulárneho grafu

- $O(n)$ premenných, $O(n)$ klauzúl, veľkosť formuly $O(n)$
- čas riešenia: konštantný faktor (do 4), závisí od konkrétneho solvera a jeho konfigurácie
- veľkosť formuly len mierne koreluje s časom riešenia

Hranové 3-farbenie 3-regulárneho grafu

- $O(n)$ premenných, $O(n)$ klauzúl, veľkosť formuly $O(n)$
- čas riešenia: konštantný faktor (do 4), závisí od konkrétneho solvera a jeho konfigurácie
- veľkosť formuly len mierne koreluje s časom riešenia
- všetky možnosti fungujú lepšie než formulovanie problému ako lineárneho programu a vyriešenie pomocou GLPK či Gurobi

Hranové 3-farbenie 3-regulárneho grafu

- $O(n)$ premenných, $O(n)$ klauzúl, veľkosť formuly $O(n)$
- čas riešenia: konštantný faktor (do 4), závisí od konkrétneho solvera a jeho konfigurácie
- veľkosť formuly len mierne koreluje s časom riešenia
- všetky možnosti fungujú lepšie než formulovanie problému ako lineárneho programu a vyriešenie pomocou GLPK či Gurobi
- pre malé grafy (cca do 40–50 vrcholov) vyhráva backtracking, najmä ak sú zafarbiteľné
- riešiteľné aj pre tisíce vrcholov

Hamiltonovská kružnica

1. premenné $x_{v,i}$ — v je i-ty vrchol
2. na každej pozícii je vrchol
3. na žiadnej poziícii nie sú dva vrcholy
4. každý vrchol je použitý najviac raz
5. každé dva susedné vrcholy sú spojené hranou

Hamiltonovská kružnica

1. premenné $x_{v,i}$ — v je i-ty vrchol
2. na každej pozícii je vrchol
3. na žiadnej poziícii nie sú dva vrcholy
4. každý vrchol je použitý najviac raz
5. každé dva susedné vrcholy sú spojené hranou

- $O(n^2)$ premenných, $O(n^3)$ klauzúl!
- pre kubické grafy funguje do 40–50 vrcholov, zhruba ako backtracking
- poučenie: „efektívna“ redukcia na SAT má lineárne veľa premenných

Hamiltonovská kružnica

- hamiltonovská kružnica je súvislý 2-faktor
- je jednoduché lokálne popísať 2-faktor
- CNF pre 2-faktor má veľkosť $O(n)$

Hamiltonovská kružnica

- hamiltonovská kružnica je súvislý 2-faktor
- je jednoduché lokálne popísať 2-faktor
- CNF pre 2-faktor má veľkosť $O(n)$
- stačí preveriť všetky 2-faktory — AllSAT
- napr. kubické grafy na 30–40 vrcholoch majú rádovo milióny 2-faktorov

Hamiltonovská kružnica

- hamiltonovská kružnica je súvislý 2-faktor
- je jednoduché lokálne popísať 2-faktor
- CNF pre 2-faktor má veľkosť $O(n)$
- stačí preveriť všetky 2-faktory — AllSAT
- napr. kubické grafy na 30–40 vrcholoch majú rádovo milióny 2-faktorov
- oveľa menej solverov: clasp, BDD_MINISAT_ALL
- rýchlejšie než vyššie uvedená redukcia na SAT, lenže počet 2-faktorov rastie exponenciálne

AllSAT

- každý SAT solver možno upraviť na hľadanie všetkých riešení: stačí pre každé nájdené riešenie pridať novú klauzulu a pustiť solver znova

AllSAT

- každý SAT solver možno upraviť na hľadanie všetkých riešení: stačí pre každé nájdené riešenie pridať novú klauzulu a pustiť solver znova
- značne neefektívne: riešení môže byť exponenciálne veľa a formula rastie; naučené informácie o pôvodnej formule sa zakaždým zahodia (cryptominisat: do 20 000 riešení)

AllSAT

- každý SAT solver možno upraviť na hľadanie všetkých riešení: stačí pre každé nájdené riešenie pridať novú klauzulu a pustiť solver znova
- značne neefektívne: riešení môže byť exponenciálne veľa a formula rastie; naučené informácie o pôvodnej formule sa zakaždým zahodia (cryptominisat: do 20 000 riešení)
- iná možnosť je obmedziť non-chronological backtracking

AllSAT

- zatiaľ najlepšie je *formula-BDD caching* [Toda 2015]
- tento mechanizmus sa dá pridať do existujúceho backtrackingu, je však nutné vopred zafixovať poradie premenných
- to citeľne oslabí výhody heuristík typu VSIDS (vyberáme už len ohodnotenie, nie premennú), ale celkovo je výkon uspokojivý, najmä ak existuje veľmi veľa riešení
- BDD_MINISAT_ALL má značné nároky na pamäť, ale jediný zvláda inštancie s miliardami riešení
- v niektorých prípadoch neriešiteľných inštancií bol dokonca rýchlejší ako top SAT solvery