3-SAT beyond brute force

Damijan Randl

21. maj 2024

Povzetek

V tem poročilu je obravnavan problem 3-SAT, njegove implikacije v svetu ter metode za njegovo reševanje. Predstavljeni bodo dve napredni metodi, DPLL in CLCD, izzivi in težave, ki sem jih imel in jih še imam, pri implementaciji le teh ter predvideno nadaljnje delo na projektu.

Kazalo

1 Uvod

Problem 3-SAT (3-Satisfiability) je posebna oblika SAT problema, kjer vsaka klavzula vsebuje natanko tri literale. SAT problem, ki spada v razred NP-polnih problemov, je ključen v teoretičnem računalništvu in ima številne aplikacije v različnih disciplinah, kot so kriptografija, verifikacija strojne opreme in programske opreme, načrtovanje, umetna inteligenca itd. Reševanje 3-SAT problema še dandanes predstavlja velik izziv, saj preproste metode surove sile hitro postanejo neuporabne za večje primere.

2 Problem 3-SAT in njegove implikacije

SAT problem je definiran kot iskanje vrednosti spremenljivk v logični formuli v konjunktivni normalni obliki (CNF), tako da je formula resnična. Logična formula je sestavljena iz logičnih spremenljivk, ki so lahko true (resnične) ali false (napačne), ter logičnih operatorjev kot so AND (in), OR (ali) in NOT (ne). V 3-SAT problemu je vsaka klavzula v formuli sestavljena iz natanko treh literalov (spremenljivk ali njihovih negacij), združenih z operatorjem OR, medtem ko so klavzule med seboj povezane z operatorjem AND. Pomembno je dodati, da je vsak n-SAT problem z redukcijo možno prevesti na 3-SAT. To je pomembna lastnost, saj nam precej poenostavi analizo in razvoj algoritmov za njihovo reševanje.

Primer logične formule v CNF obliki za 3-SAT problem:

$$(l_1 \vee \neg l_2 \vee l_3) \wedge (\neg l_1 \vee l_2 \vee \neg l_3) \wedge (l_1 \vee l_2 \vee \neg l_3) \tag{1}$$

Implikacije reševanja 3-SAT problema so zelo obsežne. Ker je 3-SAT NP-poln problem, je reševanje katerega koli NP problema mogoče reducirati na reševanje 3-SAT. To pomeni, da učinkovita rešitev 3-SAT problema vodi k učinkovitim rešitvam za širok spekter težkih računalniških problemov. Aplikacije vključujejo:

- **Kriptografija**: Mnogo kriptografskih problemov, kot je analiza kriptografskih protokolov in iskanje kolizij v zgoščevalnih funkcijah, je mogoče formulirati kot SAT problem.
- **Verifikacija strojne in programske opreme**: Preverjanje pravilnosti vezij in programov lahko prevedemo v SAT problem, kjer je potrebno preveriti, ali določena logična formula drži za vse možne vhode.
- **Načrtovanje in razporejanje**: Veliko načrtovalskih in razporejevalnih problemov, kot so urniki, logistika in dodeljevanje virov, lahko prevedemo v SAT problem.
- **Umetna inteligenca**: V AI se SAT uporablja za reševanje problemov z omejitvami, logičnih ugank in avtomatsko sklepanje.

3 Rešitve problema

Kot že navedeno, je reševanje 3-SAT problema zaradi njegove NP-polnosti zelo zahtevno. NP-polni problemi so znani po tem, da za njih ni znanih polinomskih algoritmov, kar pomeni, da se čas izvajanja reševanja problema eksponentno povečuje z velikostjo vhoda. Kljub temu so raziskovalci skozi desetletja razvili več naprednih metod, ki presegajo preprosto metodo surove sile in so učinkovitejše pri reševanju praktičnih primerov.

Ena izmed prvih metod za reševanje SAT problemov je bila metoda surove sile, kjer preprosto preizkušamo vse možne dodelitve spremenljivk, kar je časovno izjemno zahtevno. Sčasoma so se razvile bolj prefinjene metode, kot je metoda Davis-Putnam (DP), ki so jo kasneje nadgradili v metodo Davis-Putnam-Logemann-Loveland (DPLL). DPLL je postavila temelje za večino sodobnih SAT rešiteljev. Nadaljnji napredek je prinesel metodo Conflict-Driven Clause Learning (CDCL), ki je še izboljšala učinkovitost reševanja SAT problemov z učenjem iz preteklih konfliktov. Slednja je sedaj sestavni del večine naprednih programov za reševanje SAT problemov.

3.1 Metoda DPLL

Metoda Davis-Putnam-Logemann-Loveland (DPLL) je algoritem za reševanje SAT problemov, ki uporablja tehnike razveji in omeji (branch and bound) ter povratnega iskanja (backtracking). Algoritem izboljšuje osnovno metodo surove sile z uporabo dveh ključnih tehnik: širjenja enot (unit propagation) in čistega literala (pure literal elimination).

Širjenje enot (unit propagation): Če klavzula postane enolična, to pomeni, da vsebuje le en literal, katerega vrednost mora biti resnična, da bo celotna klavzula resnična. Algoritem torej prisilno dodeli to vrednost literalom v enoličnih klavzulah in posodobi preostale klavzule.

Čisti literal (pure literal elimination): Če literal (ali njegova negacija) nastopa v formuli, vendar njegova negacija ne nastopa, se ta literal imenuje čisti literal. Čisti literali se lahko varno dodelijo vrednosti, ki naredi vse njihove pojavitve resnične, saj njihova dodelitev ne vpliva na zadovoljivost drugih klavzul.

Osnovna ideja algoritma: DPLL algoritem deluje tako, da izmenično uporablja tehniki širjenja enot in čistega literala, dokler se ne izčrpajo možnosti za njihovo uporabo. Nato izbere literal in rekurzivno preizkusi obe možni vrednosti literala (resnično in napačno). Če katera od teh dodelitev vodi do zadovoljive rešitve, je formula zadovoljiva. Če vse možnosti odpovejo, je formula nezadovoljiva.

Algorithm 1 DPLL algoritem

Input: Množica klavzul Φ
Output: Logična vrednost, ki pove, ali je Φ zadovoljiva
function DPLL(Φ):
// širjenje enot (unit propagate):
while obstaja enostavna klavzula {l} v Φ **do**
    $\Phi \leftarrow$ unit-propagate(l, Φ)
end while
// čisti literal (pure literal):
while obstaja literal l, ki je čist v Φ **do**
    $\Phi \leftarrow$ pure-literal-assign(l, Φ)
end while
// zaključni pogoji:
if Φ je prazna **then**
    return true
end if
if Φ vsebuje prazno klavzulo **then**
    return false
end if
// DPLL postopek:
$l \leftarrow$ choose-literal(Φ)
return DPLL($\Phi \wedge \{l\}$) **or** DPLL($\Phi \wedge \{\neg l\}$)

Vir: Wikipedia - DPLL algorithm

3.2 Metoda CDCL

Metoda Conflict-Driven Clause Learning (CDCL) je nadgradnja metode DPLL, ki vključuje dodatne tehnike za izboljšanje učinkovitosti reševanja SAT problemov. Ključni tehniki, ki ju uporablja CDCL, sta analiza konflikta in povratni skok (non-chronological backtracking).

Analiza konflikta: Ko algoritem naleti na konflikt, to pomeni, da trenutna dodelitev vrednosti literali povzroči, da je neka klavzula napačna. Namesto da algoritem preprosto povratno sledi do prejšnje odločitvene točke (kot pri DPLL), CDCL analizira konflikt, da ugotovi vzrok konflikta. Na podlagi te analize algoritem ustvari novo klavzulo (imenovano naučena klavzula), ki preprečuje, da bi se isti konflikt ponovno pojavil v prihodnosti.

Povratni skok (non-chronological backtracking): Ko CDCL naleti na konflikt, ne sledi nujno nazaj po zadnjih odločitvah. Namesto tega skoči nazaj na prejšnjo odločitveno raven, ki je pomembna glede na konfliktno klavzulo. Ta tehnika omogoča učinkovitejšo rešitev, saj se algoritem izogne ponavljanju istih napak in hitreje najde pravilno pot do rešitve.

Algorithm 2 CDCL algoritem

Input: Množica klavzul Φ
Output: Logična vrednost, ki pove, ali je Φ zadovoljiva
function CDCL(Φ):
$\tau \leftarrow \emptyset$
while true **do**
  $\tau \leftarrow$ unit-propagate(Φ, τ) // širjenje enot
  if τ napravi kakšno klavzulo napačno **then**
    if na odločitveni ravni 0 **then**
      return *nezadovoljivo*
    end if
    $C \leftarrow$ analyze-conflict(Φ, τ) // analiza konflikta
    $\Phi \leftarrow \Phi \wedge C$ // dodaj novo klavzulo
    Skoči nazaj na prejšnjo odločitveno raven glede na C
  else
    if so vse spremenljivke dodeljene **then**
      return *zadovoljivo*
    end if
    Začni novo odločitveno raven
    Izberi literal l tako, da $\tau(l)$ ni definiran
    $\tau \leftarrow \tau \cup \{l\}$ // "odloči", da je l resničen
  end if
end while

Vir: CS-E3220: Propositional satisfiability and SAT solvers

4 Implementacija

Pri implementaciji algoritmov sem se soočil z različnimi izzivi in pridobil številne ugotovitve, ki jih predstavljam v tem poglavju.

4.1 Implementacija metode DPLL

Do sedaj sem uspel implementirati metodo DPLL. Implementacija je narejena po navedeni pseudo kodi, pri čemer sem si pomagal s člankom na spletni strani [4].

Lotil sem se tako, da sem najprej implementiral osnovno delovanje algoritma (delno interpretacijo in povratno iskanje), nato pa sem postopno dodajal funkcije, kot so širjenje enot (unit propagation), čisti literal (pure literal assign) in na koncu še pametno izbiranje novega literala z metodo VSIDS (Variable State Independent Decaying Sum). VSIDS je hevristična metoda, ki temelji na dodeljevanju višjih prioritet literalom, ki so se nedavno pojavili v konfliktnih klavzulah, kar omogoča hitrejše iskanje rešitev.

To sem naredil zato, ker nameravam predstaviti razlike v časovni zahtevnosti algoritma glede na dodane izboljšave. Pri tem algoritmu večjih težav nisem imel, seveda pa je vedno mogoča optimizacija funkcij (z raznimi hevristikami), s čimer bi bil program še hitrejši pri reševanju SAT problemov.

4.2 Implementacija metode CDCL

Veliko časa sem vložil tudi v implementacijo CDCL algoritma po zgledu iz [5]. Implementacija tega algoritma je veliko zahtevnejša od navadnega DPLL.

Trenutno se ukvarjam z implementacijo funkcionalnosti učenja klavzul, ki zahteva veliko dodatnega razumevanja in je ena ključnih komponent tega algoritma, a mislim, da sem na dobri poti.

Težave vidim predvsem v ne najboljši izbiri podatkovnih struktur. Na primer, tekom učenja klavzul naletimo na pojem *implication graph* (graf implikacij), ki predstavlja odvisnosti med dodelitvami literala. Struktura grafa se uporablja predvsem za razlago problema, za učinkovito implementacijo pa bi bilo potrebno uporabiti naprednejše podatkovne strukture, kot so dinamični seznam (dynamic array) ali povezani seznam (linked list) s katerimi pa postane delo nekoliko bolj abstraktno.

Zaradi navedenih težav mislim, da končni program ne bo ravno konkuriral modernejšim programom, kot so Glucose3, MiniSat, Lingeling, Chaff, a vseeno mislim, da bi se morala poznati očitna razlika z DPLL algoritmom.

4.3 Nadaljnje delo

V nadaljevanju nameravam dokončati implementacijo algoritma CDCL. Implementiral bom tudi nekoliko lepši uporabniški vmesnik, kjer bo možno ročno vnesti formulo v CNF obliki, ali pa izbrati kar .cnf datoteko in bo program vrnil ustrezno rešitev. Prav tako nameravam narediti nekaj primerjav med navedenima algoritmoma na testnih primerih.

Literatura

[1] Knuth, D. E. (2015). *The Art of Computer Programming, Volume 4: Fascicle 6: Satisfiability*. Addison-Wesley.

[2] Marques-Silva, J. and Sakallah, K. A. (1999). GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5), 506-521.

[3] Biere, A., et al., (Eds.). (2009). *Handbook of satisfiability* (Vol. 185). IOS press.

[4] Liberatore, P. DPLL algorithm. Università degli Studi di Roma "La Sapienza". Dostopno na: [`http://www.diag.uniroma1.it/~liberato/ar/dpll/dpll.html`]

[5] Junttila, T. (2020). Notes on the Conflict-Driven Clause Learning (CDCL) algorithm. Aalto University. Dostopno na: [`https://users.aalto.fi/~tjunttil/2020-DP-AUT/notes-sat/cdcl.html`]

[6] Wikipedia. (n.d.). DPLL algorithm. Dostopno na: [`https://en.wikipedia.org/wiki/DPLL_algorithm`]

[7] Wikipedia. (n.d.). Conflict-Driven Clause Learning. Dostopno na: [`https://en.wikipedia.org/wiki/Conflict-Driven_Clause_Learning`]