Ohjelman yleisrakenne

Ohjelman yleisrakenne on yksinkertainen. Tiedosto main.py toimii pääohjelmana ja tiedosto satsolver.py sisältää varsinaisen algoritmin toteutuksen tarvittavine funktioineen. Hakemisto test sisältää kaiken testamiseen liittyvän, kuten tarvittavat Python-skriptit ja Dimacs-tiedostot.

Aika- ja tilavaativuusanalyysi

Aikavaativuus

DPLL-algoritmin aikavaativuutta on vaikea arvioida tarkasti, koska se riippuu syötteen rakenteesta ja heuristiikan toimivuudesta. Pahimmassa tapauksessa se voi olla eksponentiaalinen muuttujien määrän n suhteen.

1. **Yksikköpropagaatio ja puhtaan literaalin poisto**
    - Jokainen muuttujan määritys voi vaikuttaa kaikkiin lausekkeisiin, joten pahimmillaan nämä operaatiot vievät O(m), missä m on lausekkeiden määrä.
    - Jos propagoinnit jatkuvat rekursiivisesti, niiden vaikutus kasvaa.
2. **Haarautuminen (Muuttujan valinta ja rekursio):**
    - Algoritmi valitsee muuttujan ja kokeilee molempia arvoja (True/False), mikä voi johtaa eksponentiaaliseen kasvuun.
    - Pahimmassa tapauksessa syntyy täysin binäärinen hakupuu, jonka syvyys on O(n), eli aikavaativuus on O(2^n)
3. **Käytännön suorituskyky:**
    - Heuristiikat, kuten MOM (Maximum Occurrences in Minimum-sized clauses), voivat parantaa keskimääräistä tapausta.
    - Yksikköpropagaatio ja puhtaan literaalin poisto voivat vähentää muuttujien määrää merkittävästi, jolloin todellinen suorituskyky on usein huomattavasti parempi kuin pahimmassa tapauksessa.

Pahin tapaus:

- Eksponentiaalinen: O(2^n), koska SAT-ongelma on NP-täydellinen.

Keskimääräinen tapaus käytännössä:

- Joissakin erityistapauksissa (esim. helposti ratkaistavat ongelmat) algoritmi voi toimia jopa polynomisessa ajassa
- Haastavimmat tapaukset ovat vaiheensiirtymässä (clause-to-variable ratio c ≈ 4.25, jolloin ratkaisu on tyypillisesti vaikein [5]

Tilavaativuus

1. **Tallennetut lausekkeet ja muuttujat:**
    - Algoritmi tallentaa syötteessä olevat lausekkeet, jolloin perusmuistinkulutus on O(m)
    - Muuttujien määritykset vievät tilaa O(n)
2. **Backtracking:**
    - Jokainen rekursiivinen kutsu lisää uuden haaran hakupuuhun
    - Pahimmillaan rekursion syvyys on O(n), jolloin pinoon kuluva tila on O(n)
3. **Kopioitavat lausekkeet:**
    - Jokaisessa rekursiohaarassa algoritmi tallentaa osittaisen tilan (eli lausekkeet ja muuttujien arvot)
    - Pahimmillaan jokaisessa rekursiohaarassa on O(m) lausekkeita, mikä johtaa kokonaismuistin kulutukseen $O(n \cdot m)$

Parannusehdotukset

Algoritmin toiminnan tehostaminen esim. hyödyntämällä kehittyneempiä heuristiikoita, tietorakenteita ja optimointitekniikoita.

Kielimallien käyttö

ChatGPT:tä on käytetty tiedon etsintään aiheeseen liittyen. Kyselin myös ideatason vinkkejä algoritmin toiminnan tehostamiseksi ja ehdin niihin joihinkin teoriatasolla tutustuakin, mutta kooditasolle ne eivät tämän kurssin puitteissa ehtineet. Heti alkuun pyysin esim. vinkkejä hyvistä aihetta käsittelevistä tieteellisistä lähteistä ja niitä löytyikin esim. kirja [1] oli erittäin hyvä.

Lähdeluettelo

[1] Handbook of Satisfiability 3 Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsh (Eds.) IOS Press, 2009

[2] https://github.com/arminbiere/cadical.

[4] https://massimolauria.net/cnfgen

[5] https://dl.acm.org/doi/10.5555/2832249.2832300

[6] https://users.aalto.fi/~tjunttil/2020-DP-AUT/notes-sat/dpll.html