Analiza Problemei de Satisfiabilitate (SAT) și Implementarea Algoritmilor Clasici

Nicorescu Isabella-Mariana
Universitatea de Vest din Timișoara
Facultatea de Matematică-Informatică, specializarea Informatică
isabella.nicorescu05@e-uvt.ro

May, 2025

Cuprins

1 Rezumat

Satisfiabilitatea în logică (SAT) este o problemă centrală în informatică și logică teoretică, cu aplicații importante în domenii precum verificarea programelor, inteligența artificială și optimizarea. Problema SAT presupune determinarea dacă există o asignare a valorilor de adevăr care satisface o formulă logică dată în formă normală conjunctivă (CNF). În această lucrare, este prezentată implementarea celor cei trei algoritmi clasici faimoși pentru rezolvarea problemei SAT: algoritmul de rezoluție, algoritmul Davis-Putnam (DP) și algoritmul Davis-Putnam-Logemann-Loveland (DPLL).

2 Introducere

Problema satisfiabilității în logică (SAT) este esențială în domeniul teoriei complexității și are numeroase aplicații în informatică, de la verificarea programelor și diagnosticul automat, la optimizarea combinatorială. SAT este o problemă NP-completă, ceea ce înseamnă că nu există algoritmi eficienți cunoscuți care să rezolve toate instanțele acestei probleme în timp polinomial. Astfel, cercetările și tehnicile de rezolvare a SAT sunt de o importanță majoră în informatică.

Această lucrare se concentrează pe trei dintre cei mai importanți algoritmi pentru rezolvarea problemei SAT: algoritmul de rezoluție, algoritmul Davis-Putnam (DP) și algoritmul Davis-Putnam-Logemann-Loveland (DPLL). În cadrul lucrării sunt incluse implementări practice ale fiecărui algoritm, care reușesc să rezolve problema SAT, fiind prezentate și ideea din spate al fiecărui algoritm (cum a fost gândit).

3 Problema SAT

3.1 SAT - NP-completă

SAT este una dintre cele mai importante probleme din teoria complexității computaționale. SAT este considerată o problemă NP-completă, ceea ce înseamnă că face parte dintr-o clasă de probleme care sunt greu de rezolvat în mod eficient, dar care pot fi verificate rapid (în timp polinomial).

SAT a fost demonstrată ca fiind NP-completă de către Stephen Cook în 1971, în teorema fundamentală cunoscută sub numele de "Teorema lui Cook-Levin". Aceasta a arătat că orice problemă din NP poate fi transformată în echivalentul unei instanțe de SAT, astfel că rezolvarea SAT poate rezolva orice altă problemă NP. Astfel, dacă ar exista un algoritm eficient pentru rezolvarea SAT, acest algoritm ar putea rezolva toate problemele din NP într-un timp polinomial.

3.2 Tranziția de fază

Tranziția de fază este un concept care provine din fizică și care descrie comportamentele unui sistem care trece printr-o schimbare bruscă a stării sale, similar cu modul în care apa se transformă în vapori sau gheață în funcție de temperatură. În contextul SAT, tranziția de fază descrie un comportament similar atunci când instanțele de SAT devin mai greu de rezolvat pe măsură ce parametrii formulei (precum numărul de variabile și clauze) cresc.

Fenomenul de tranziție de fază este important pentru înțelegerea comportamentului algoritmilor care rezolvă SAT, întrucât aceștia pot întâmpina dificultăți semnificative în această zonă de tranziție, unde complexitatea crește semnificativ.

3.3 Implementarea Algoritmilor - Gândirea din spatele implementării

Pentru a asigura că algoritmii implementați pentru rezolvarea problemei SAT funcționează corect și eficient, am avut în vedere câteva aspecte esențiale. Unul dintre principalele obiective a fost ca acești trei algoritmi (rezoluție, DP și DPLL) să poată rezolva măcar câteva exemple importante în SAT. Astfel, am creat o implementare care să funcționeze și să ofere un mod simplu și intuitiv de a verifica satisfiabilitatea unei formule logice.

Pentru a facilita introducerea formulei logice și a asigura o implementare corectă și eficientă, am ales un format standardizat pentru reprezentarea clauzelor. Formulele logice sunt reprezentate în formă normală conjunctivă (CNF), unde o formulă este o conjuncție de clauze, iar fiecare clauză este o disjuncție de literali - un concept simplu și eficient din teoria logicii booleene. Literalii sunt valori booleene sau variabile (care pot fi fie adevărate, fie false) și sunt reprezentate prin numere întregi. Fiecare literar este asociat cu o variabilă, iar negarea unui literar este exprimată prin același număr, dar cu semnul negativ. Spre exemplu, dacă o variabilă este reprezentată ca fiind numărul 2, atunci negarea acesteia va fi exprimată prin numărul -2.

După introducerea întregii formule, utilizatorul poate rula algoritmii implementați pentru a verifica dacă formula este satisfiabilă. Algoritmii de rezoluție, Davis-Putnam și DPLL vor procesa formula și vor determina dacă există o asignare de valori de adevăr pentru variabilele literale care satisface întreaga formulă. În funcție de rezultatul acestora, va fi afișat un mesaj corespunzător indicând dacă formula este satisfiabilă sau nu.

3.4 Reguli pentru Introducerea Formulei

Pentru o introducere corectă a formulelor în program, se aplică următoarele reguli:

1. **Numărul de clauze:** La început, utilizatorul introduce un număr întreg care reprezintă câte clauze are formula.

2. **Scrierea clauzelor:** Fiecare clauză se scrie pe o linie separată, folosind literali întregi.

3. **Literali pozitivi și negativi:** Literalii pozitivi reprezintă variabile, iar cei negativi reprezintă negațiile acestora.

4. **Sfârșitul unei clauze:** Fiecare clauză se termină cu cifra `0`, marcând astfel finalul acesteia.

5. **Trecerea la următoarea clauză:** După apăsarea Enter, se poate introduce clauza următoare.

4 Algoritm

Implementarea celor trei algoritmi a fost realizată în limbajul C++, utilizând standardul C++17, pentru a beneficia de facilitățile moderne oferite de acesta, precum manipularea eficientă a structurii datelor și expresivitatea sporită a codului.

Codul a fost dezvoltat și testat într-un mediu de compilare online, ceea ce a oferit un acces rapid și portabil la testare, însă cu anumite limitări privind monitorizarea precisă a resurselor de sistem, cum ar fi consumul de memorie RAM sau timpul de execuție în milisecunde. Din acest motiv, în cadrul acestei lucrări nu sunt prezentate evaluări cantitative detaliate ale performanței algoritmilor, ci se pune accent pe corectitudinea logică și generalitatea soluției.

Pentru a permite testarea ușoară de către utilizator, citirea formulelor SAT se face de la tastatură, într-un format standardizat. Utilizatorul este invitat să introducă manual fiecare clauză, sub formă de literali întregi, unde variabilele sunt reprezentate prin numere pozitive, iar negațiile acestora prin aceleași numere cu semn negativ. Fiecare clauză se încheie cu 0, iar o nouă clauză este introdusă pe o linie separată.

Codurile sursă complete, împreună cu instrucțiuni de rulare și exemple, sunt disponibile public pe GitHub, la următoarea adresă: Problema SAT - coduri sursă

În cele ce urmează, vor fi prezentate fragmente relevante din implementările celor trei algoritmi utilizați pentru rezolvarea problemei satisfiabilității: Rezoluția, DP și DPLL.

4.1 Algoritmul de Rezoluție

[1] RESOLUTION(formula)
  clauses ← toate clauzele din formulă
  repeat
    new_clauses ← ∅
    for toate clauzele C_1 din *clauses*
      for toate clauzele $C_2 > C_1$ din *clauses*
        resolvent ← resolve(C_1, C_2)
        if resolvent este clauza vidă:
          afișează "Formula este NESATISFIABILĂ"
          return FALS
        else resolvent nu este în *clauses*:
          new_clauses ← new_clauses ∪ {resolvent}
    if new_clauses este vid:
      afișează "Formula este SATISFIABILĂ"
      return ADEVĂRAT
    clauses ← clauses ∪ new_clauses
until nu mai apar clauze noi

4.2 Algoritmul Davis–Putnam

[2] DAVISPUTNAM(formula)
  if formula este goală:
    return ADEVĂRAT
  if formula conține clauză vidă:
    return FALS

alege o variabilă x
elim_x ← elimină toate clauzele care conțin x
resolvenți ← clauze rezultate prin rezoluție pe x
formula_nouă ← (elim_x fără x) ∪ resolvenți
return DAVISPUTNAM(formula_nouă)

4.3 Algoritmul DPLL)

[3] DPLL(formula)
if formula este goală:
return ADEVĂRAT
if formula conține clauză vidă:
return FALS
if există literal unitar l:
atribuie l ca adevărat și propagă
return DPLL(formula modificată)
alege literal l
return DPLL(formula ∪ {l adevărat})
OR DPLL(formula ∪ {l fals})

5 Interacțiunea cu Utilizatorul și Afișarea Rezultatelor

Pentru a asigura o experiență de utilizare intuitivă și accesibilă, interacțiunea cu utilizatorul este realizată exclusiv prin intermediul consolei, în mod text. Utilizatorului i se oferă mesaje clare și succinte pentru fiecare pas al procesului de introducere a datelor și pentru interpretarea rezultatului final.

După rularea programului, utilizatorul este întâmpinat cu mesajul: "Introdu numarul de clauze:". Acest pas permite introducerea numărului total de clauze care alcătuiesc formula logică în formă normală conjunctivă (CNF). După introducerea acestui număr, programul continuă cu solicitarea fiecărei clauze în parte, prin afișarea mesajului: "Introdu fiecare clauza (literali intrgi), terminata cu 0:". Utilizatorul introduce fiecare clauză pe o linie separată, folosind literali întregi (cu semnul negativ pentru negație), și marcând finalul fiecărei clauze prin 0. După ce toate clauzele au fost introduse, programul procesează formula și, în funcție de algoritmul aplicat, afișează unul dintre următoarele mesaje finale, care exprimă verdictul privind satisfiabilitatea formulei:

6 Literatură de Specialitate și o Comparație

În cadrul acestei lucrări au fost implementați trei algoritmi pentru determinarea satisfiabilității formulelor logice în CNF: rezoluția, Davis–Putnam (DP) și Davis–Putnam–Logemann–Loveland (DPLL). Fiecare algoritm are caracteristici și avantaje distincte, iar comparația lor se poate face pe baza următoarelor criterii: corectitudine, generalitate, complexitate și eficiență practică.

- **Rezoluția** – Este o metodă bazată pe logica propozițională care aplică repetat regula de rezoluție până când fie se obține clauza vidă (ceea ce dovedește că formula

este nesatisfiabilă), fie nu se mai pot face pași noi (ceea ce înseamnă că este satisfiabilă). Avantajul său este simplitatea teoretică și faptul că este completă pentru forme CNF. Totuși, în practică, rezoluția poate deveni ineficientă deoarece generează multe clauze noi și poate consuma multă memorie.

- **Davis–Putnam (DP)** – Acest algoritm rafinează rezoluția prin eliminarea variabilelor și simplificarea formulei, aplicând rezoluții doar când este necesar. Este complet și corect, dar poate fi foarte costisitor în practică, deoarece generează toate clauzele posibile prin eliminarea unei variabile, ceea ce duce la o creștere exponențială a numărului de clauze.

- **DPLL** – Este o variantă recursivă îmbunătățită care implementează alegerea unei variabile, aplicarea unui backtracking inteligent și propagarea literalilor unici (unit propagation). DPLL este mult mai eficient în practică și stă la baza solverelor moderne. Algoritmul este adaptabil, poate găsi rapid o soluție sau determina că nu există, fără a explora toate posibilitățile, ceea ce îl face cel mai eficient dintre cei trei în majoritatea cazurilor.

7 Concluzie

Lucrarea de față a avut ca obiectiv explorarea problemei satisfiabilității booleene (SAT), o problemă centrală în teoria complexității și logica computațională, prin analiza și implementarea a trei algoritmi consacrați: rezoluția, Davis–Putnam (DP) și Davis–Putnam–Logemann–Lovela (DPLL).

Prin intermediul implementărilor realizate, s-a urmărit nu doar validarea funcționalității teoretice a fiecărui algoritm, ci și oferirea unui instrument practic, capabil să determine satisfiabilitatea formulelor introduse interactiv de utilizator. Alegerea de a lucra cu literali reprezentați prin numere întregi și organizarea datelor în formă normală conjunctivă (CNF) a permis o structurare clară a inputului și o verificare eficientă a formulelor.

Rezultatele obținute în urma testării indică diferențe semnificative între cele trei metode. Algoritmul bazat pe rezoluție, deși complet din punct de vedere teoretic, devine rapid ineficient în cazul formulelor de dimensiuni mai mari, din cauza generării unui număr mare de clauze intermediare. Davis–Putnam aduce o optimizare prin eliminarea variabilelor, însă complexitatea combinatorică rămâne ridicată. În contrast, DPLL se remarcă prin eficiența sa practică, datorită mecanismelor de propagare a literalilor unitari și a backtracking-ului sistematic, motiv pentru care constituie baza celor mai moderne solvere SAT.

Este important de menționat că, din cauza utilizării unui compilator online, nu a fost posibilă analiza detaliată a consumului de resurse (timp de execuție, memorie RAM etc.). Cu toate acestea, implementările realizate au demonstrat corectitudinea algoritmilor și capacitatea acestora de a răspunde eficient la instanțe diverse ale problemei SAT.

În concluzie, această lucrare confirmă relevanța continuă a metodelor clasice în rezolvarea SAT și oferă o bază solidă pentru extinderea ulterioară către solvere moderne sau aplicații în domenii conexe, precum verificarea formală, inteligența artificială sau optimizarea combinatorică.

8 Bibliografie

Bibliografie

[1] Cook, S. A. (1971). *The Complexity of Theorem-Proving Procedures*. Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC), pp. 151–158.

[2] Karp, R. M. (1972). *Reducibility Among Combinatorial Problems*. In Complexity of Computer Computations, Springer.

[3] Davis, M., Putnam, H. (1960). *A Computing Procedure for Quantification Theory*. Journal of the ACM, 7(3), 201–215.

[4] Davis, M., Logemann, G., Loveland, D. (1962). *A Machine Program for Theorem Proving*. Communications of the ACM, 5(7), 394–397.

[5] Biere, A., Heule, M. J. H., van Maaren, H., & Walsh, T. (Eds.). (2009). *Handbook of Satisfiability*. IOS Press.

[6] Bibel, W. (1987). *Automated Theorem Proving: Theory and Practice*. Vieweg+Teubner Verlag.

[7] Mitchell, D., Selman, B., & Levesque, H. (1992). *Hard and Easy Distributions of SAT Problems*. Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI), pp. 459–465.

A Anexe

A.1 Exemplu de formulă SATISFIABILĂ - Rezoluție

```
Pasul 1:
Introdu numarul de clauze: 3

Pasul 2:
Introdu fiecare clauza (literari intregi), terminata cu 0:
1 2 0
-1 3 0
-2 -3 0

Pasul 3:
Formula este SATISFIABILĂ (nu s-a putut deduce clauza vidă).
```

A.2 Fragmente din codul sursă

```
\\ Citim formula de la tastatură - DP
CNFFormula read_formula_from_user() {
    CNFFormula formula;
    Clause clause;
    int literal;
    int num_clauses;

    cout << "Introdu numarul de clauze: ";
    cin >> num_clauses;
    cout << "Introdu fiecare clauza (literali intregi), terminata cu 0:\n";

    for (int i = 0; i < num_clauses; ++i) {
        clause.clear();
        while (cin >> literal) {
            if (literal == 0) break;
            clause.insert(literal);
        }
        formula.push_back(clause);
    }

    return formula;
}

// Aplicăm Rezoluția pe două clauze

bool resolve(const Clause& c1, const Clause& c2, Clause& resolvent) {
    int count = 0;
    int pivot = 0;
```

```
    for (int lit : c1) {
        if (c2.count(-lit)) {
            ++count;
            pivot = lit;
        }
    }

    if (count != 1) return false;

    resolvent = c1;
    resolvent.insert(c2.begin(), c2.end());
    resolvent.erase(pivot);
    resolvent.erase(-pivot);

    return true;
}

 // 5. Alegere literal arbitrar recursiv - DPLL

 int chosen = *formula[0].begin();

 CNFFormula formula_true = formula;
 remove_clauses_with_literal(formula_true, chosen);
 remove_literal(formula_true, -chosen);
 if (dpll(formula_true)) return true;

 CNFFormula formula_false = formula;
 remove_clauses_with_literal(formula_false, -chosen);
 remove_literal(formula_false, chosen);
 return dpll(formula_false);
}
```