Comparație Teoretică și Experimentală a Algoritmilor de Rezolvare a Problemei Satisfiabilității în Logica Propozițională

Haniș Manase
Universitatea de Vest Timișoara

4/04/2025

1. Introducere

Problema satisfiabilității (SAT) este una dintre cele mai fundamentale probleme din informatică teoretică. Se pune întrebarea dacă o formulă logică propozițională are o atribuire a variabilelor care o face adevărată. În această lucrare vom analiza trei metode importante de rezolvare a acestei probleme: algoritmul de rezoluție, algoritmul Davis-Putnam (DP) și algoritmul Davis-Putnam-Logemann-Loveland (DPLL).

2. Prezentare teoretică a algoritmilor

2.1 Algoritmul de rezoluție

Algoritmul de rezoluție este o tehnică de inferență folosită pentru a verifica **nesatisfiabilitatea** unei formule în logica propozițională. El funcționează doar pe formule exprimate în **formă normală conjunctivă (CNF)**, adică o conjuncție de clauze, fiecare fiind o disjuncție de literali.

Regula de rezoluție:
Dacă avem două clauze:

$$(A \vee C_1), \quad (\neg A \vee C_2) \Rightarrow (C_1 \vee C_2)$$

atunci se poate deduce o clauză nouă, eliminând variabila A și combinând restul literalilor.

Exemplu:
Fie formula:

$$(A \vee B), \quad (\neg A \vee C), \quad (\neg B), \quad (\neg C)$$

Pași de rezoluție:

- Aplicăm rezoluția între $(A \vee B)$ și $(\neg B)$: obținem (A)
- Aplicăm rezoluția între (A) și $(\neg A \vee C)$: obținem (C)

- Aplicăm rezoluția între (C) și $(\neg C)$: obținem clauza vidă $\square$

Apariția clauzei vide indică faptul că formula este **nesatisfiabilă**.

Proprietăți:

- Complet pentru demonstrări de nesatisfiabilitate (dacă formula e nesatisfiabilă, rezoluția va deduce clauza vidă).
- Nu produce un model satisfăcător (nu spune ce valori fac formula adevărată).
- Poate produce un număr exponențial de clauze intermediare.
- Mai mult interes teoretic decât practic.

Complexitate:

- În cel mai rău caz, rezoluția generează un număr exponențial de clauze.
- Fără strategii de control, este ineficientă pentru formule mari.

Utilizare practică: Deși rar folosită în SAT-solvers moderni, rezoluția este importantă în demonstrarea automată a teoremelor și ca bază teoretică pentru alți algoritmi mai eficienți precum Davis-Putnam și DPLL.

2.2 Algoritmul Davis–Putnam (DP)

Algoritmul Davis–Putnam este o metodă de rezolvare a formulelor SAT exprimată în formă CNF, care îmbunătățește rezoluția prin eliminarea sistematică a variabilelor din formulă.

Ideea principală: Algoritmul alege o variabilă și aplică rezoluția între toate perechile de clauze care conțin respectiv variabila și negația ei. Apoi elimină toate clauzele ce conțin acea variabilă.

Pași principali:

- Se alege o variabilă x.
- Se aplică rezoluția pe x: pentru fiecare clauză care conține x, și fiecare clauză care conține $\neg x$, se generează o clauză rezolvată.
- Se elimină toate clauzele care conțin x sau $\neg x$.
- Se repetă procesul până când:
  - se obține clauza vidă ($\square$) $\Rightarrow$ formula este nesatisfiabilă;
  - sau nu mai rămân clauze $\Rightarrow$ formula este satisfiabilă.

Proprietăți:

- Algoritm complet: determină dacă formula este satisfiabilă.
- Elimină variabilele în mod ireversibil.

- Nu construiește un model satisfăcător.
- În practică, produce multe clauze noi, ceea ce duce la o explozie de spațiu.

Complexitate:

- Timpul de rulare este exponențial în numărul de variabile.
- Cost mare de memorie din cauza numărului de clauze generate prin rezoluție.

Observație: Algoritmul Davis–Putnam a fost înlocuit în practică de DPLL, care utilizează backtracking și strategii de propagare mai eficiente.

2.3 Algoritmul Davis–Putnam–Logemann–Loveland (DPLL)

Algoritmul DPLL este o versiune îmbunătățită a algoritmului Davis–Putnam, propusă în 1962. Acesta introduce două idei-cheie: **backtracking** și **propagarea literalelor unitare**, ceea ce îl face mult mai eficient în practică.

Ideea principală: Algoritmul explorează recursiv spațiul posibil al atribuirilor, încercând să găsească o configurație satisfăcătoare a variabilelor. În fiecare nod al arborelui de decizie, se face o alegere pentru o variabilă, iar apoi se aplică regulile de simplificare.

Pași principali:

1. Dacă formula este goală (toate clauzele au fost satisfăcute), returnează **SAT**.
2. Dacă există o clauză vidă, returnează **UNSAT**.
3. Se aplică:
   - **Unit propagation** – dacă există o clauză cu un singur literal, atribuie valoarea care o face adevărată.
   - **Pure literal elimination** – dacă o variabilă apare doar cu un semn, atribuie valoarea care o satisface.
4. Se alege o variabilă și se face recursie pentru ambele valori posibile (true/false).

Pseudo-cod simplificat:

```
function DPLL(formula, assignment):
    formula := simplify(formula, assignment)
    if formula is empty: return True
    if formula contains empty clause: return False
    var := choose_variable(formula)
    return DPLL(assign(formula, var, True)) or
           DPLL(assign(formula, var, False))
```

Proprietăți:

- Produce un model satisfăcător (dacă există).

- Poate evita multe ramuri inutile prin propagare.
- Eficient în practică și folosit ca bază pentru multe SAT-solvers moderne.

Complexitate:

- În cel mai rău caz, tot exponențial, dar mult mai eficient decât rezoluția sau DP în cazuri reale.

Importanță practică: DPLL este piatra de temelie pentru SAT-solvers moderni. Algoritmi mai avansați, precum CDCL (Conflict-Driven Clause Learning), sunt extinderi directe ale DPLL.

2.4 Tabel comparativ teoretic

Criteriu	**Rezoluție**	**Davis-Putnam**	**DPLL**
Anul apariției	1960	1960	1962
Idee de bază	Refutare logică	Eliminare variabile	Backtracking + propagare
Complexitate	Exponențială	Exponențială	Exponențială
Formă necesară	CNF	CNF	CNF
Model SAT	Nu	Nu	Da
Utilizare modernă	Rar	Aproape deloc	Foarte frecvent

3. Strategii de alegere a pasului următor

Algoritmii de tip DPLL și derivații săi trebuie să decidă, în fiecare pas, **ce variabilă să aleagă** și **ce valoare să îi atribuie**. Aceste decizii au un impact major asupra performanței, motiv pentru care au fost dezvoltate numeroase strategii (heuristici) de alegere.

3.1 Heuristici pentru selecția variabilei

Strategii simple:

- **First unassigned** – se alege prima variabilă neatribuită. Este o metodă simplă, dar ineficientă în practică.
- **Random** – se selectează aleatoriu o variabilă. Utilizată rar, din cauza comportamentului instabil.
- **Static order** – se respectă o ordine prestabilită de variabile. Utilă în unele cazuri speciale.

Heuristici dinamice eficiente:

- **DLIS (Dynamic Largest Individual Sum)**: se alege variabila care apare cel mai frecvent în clauzele nesatisfăcute, ținând cont separat de aparițiile pozitive și negative.
- **VSIDS (Variable State Independent Decaying Sum)**: fiecare variabilă are un scor care crește atunci când apare în conflicte. Scorurile „se degradează" în timp pentru a oferi prioritate problemelor recente. Este folosită în solvers precum MiniSAT.
- **Jeroslow–Wang (JW)**: se calculează un scor pentru fiecare literal bazat pe frecvența sa și lungimea clauzelor în care apare. Se preferă literalii care apar în clauze scurte și frecvente.

3.2 Alegerea valorii (branching decision)

După alegerea variabilei, algoritmul trebuie să decidă ce valoare să încerce mai întâi:

- **Ordine fixă** – de exemplu, întâi `True`, apoi `False`.
- **Ordine ghidată** – pe baza analizelor anterioare, algoritmul poate încerca mai întâi valoarea care a dus la mai puține conflicte.

3.3 Propagare și învățare

- **Propagare unitară (Unit Propagation)**: dacă o clauză conține un singur literal neatribuit, atunci acel literal trebuie să fie adevărat pentru a satisface clauza. Acest pas este aplicat automat după fiecare alegere.
- **Învățare din conflicte (Clause Learning)**: în algoritmi mai avansați (ex: CDCL), dacă se ajunge la un conflict, algoritmul învață o clauză nouă care împiedică revenirea în același punct problematic.

3.4 Impactul asupra performanței

Alegerea unei strategii eficiente:

- Reduce numărul de ramuri explorate în arborele de căutare.
- Poate scurta drastic timpul de rezolvare.
- Este esențială în solvers SAT moderni.

4. Implementare

Se pot implementa algoritmii în C++:

```
{https://github.com/Manase05/MPI.git} {Link catre Github}

Pentru a rula codul este RECOMANDAT folosirea programului "Codeblocks".
Datele sunt generate aleatoriu iar pe baza lor se produc rezultatele dorite.
```

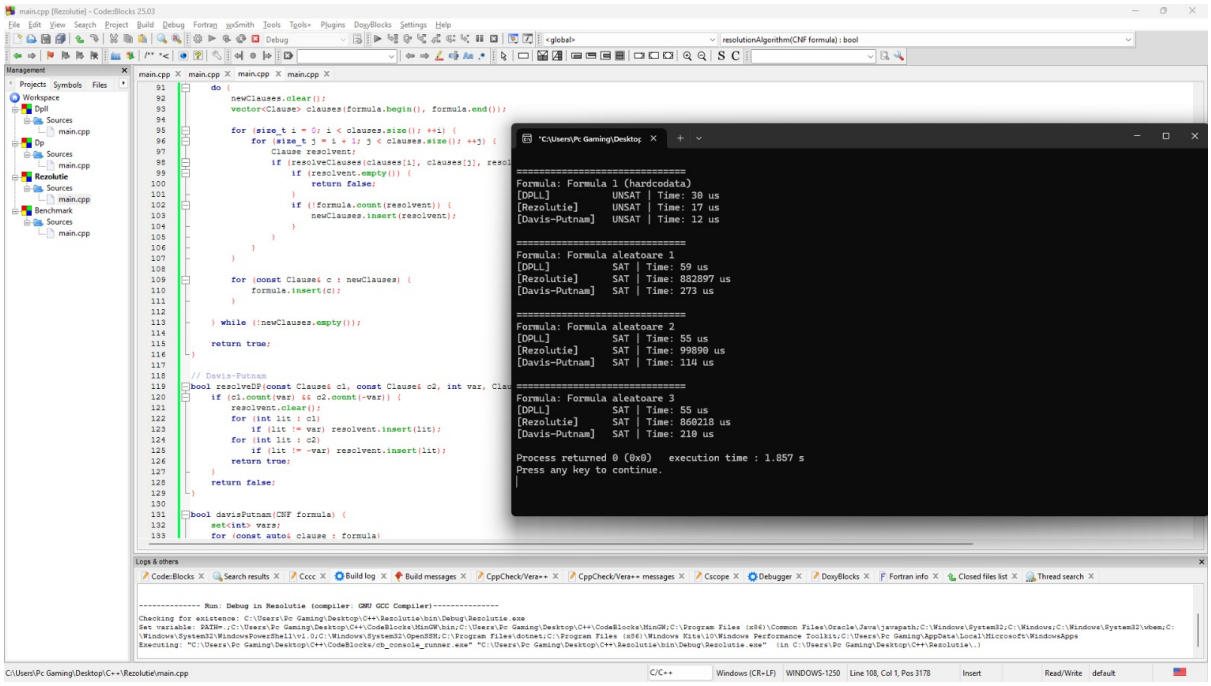

Figura 1: Caption

5. Experimente și rezultate

5.1 Măsurători

Pentru fiecare algoritm:

- Timp de rulare
- Număr de pași/decizii
- Număr de clauze procesate

5.2 Rezultate experimentale

Algoritm	Timp (us)	SAT/UNSAT
DPLL	59	SAT
Rezolutie	882897	SAT
Davis-Putnam	273	SAT

6. Concluzii

DPLL se dovedește a fi cel mai eficient algoritm în cazul formulelor testate. Rezoluția este teoretic interesantă, dar ineficientă practic. Alegerea variabilelor și propagarea sunt cruciale pentru performanță.

Cuvinte cheie: satisfiabilitate, SAT, DPLL, rezoluție, CNF, algoritmi, logică propozițională.