Studiu experimental asupra metodelor de rezolvare a problemei SAT: Rezoluție, Davis–Putnam și DPLL

Mihai-Robert Șerban
Universitatea de Vest din Timișoara

Aprilie 2025

Cuprins

Rezumat

Această lucrare prezintă un studiu comparativ al algoritmilor clasici pentru rezolvarea problemei de satisfiabilitate booleană (SAT): Rezoluție, Davis-Putnam și DPLL. Pe baza unui set de formule CNF standard din biblioteca SATLIB și a unor instanțe generate aleator, am evaluat eficiența acestor metode prin metrici precum timpul de execuție, memoria utilizată și rezultatul obținut (SAT/UNSAT). Rezultatele confirmă comportamentul „easy-hard-easy" în distribuția satisfiabilității și relevă avantajele algoritmului DPLL pentru instanțe de dimensiuni medii.

1 Introducere

Problema SAT (Satisfiabilitatea propozițională) este una dintre cele mai studiate probleme din informatică teoretică, fiind prima recunoscută ca NP-completă în urma teoremei lui Cook. Ea constă în determinarea existenței unei atribuiri de valori de adevăr variabilelor dintr-o formulă logică astfel încât formula să fie satisfăcută. Deși este o problemă simplu de formulat, este în centrul multor alte probleme din clasa NP, fiind esențială pentru înțelegerea complexității computaționale.

SAT are numeroase aplicații practice în domenii precum:

- verificarea formală a sistemelor hardware și software;
- inteligență artificială și planificare;
- analiză statică și verificarea modelelor;
- optimizare combinatorie;
- modelare de probleme logice (inclusiv jocuri precum Sudoku).

Un aspect interesant în studiul SAT este fenomenul de **tranziție de fază** [2]. Acesta se referă la o zonă critică în care instanțele de SAT trec brusc de la a fi, în majoritate, satisfiabile, la a fi majoritar nesatisfiabile. Acest comportament apare atunci când raportul clauze/variabile (denumit și densitatea formulei) atinge o valoare specifică, în jur de 4.3 pentru formule random 3-SAT. În această zonă, dificultatea de rezolvare atinge un maxim, ceea ce este extrem de relevant pentru testarea și compararea performanțelor algoritmilor SAT.

În această lucrare ne propunem să analizăm comportamentul a trei metode clasice de rezolvare SAT — Rezoluție, Davis–Putnam și DPLL — aplicate atât pe formule generate aleatoriu, cât și pe instanțe din lumea reală (SATLIB). Vom folosi, de asemenea, un exemplu concret (Sudoku 4x4) ca studiu de caz detaliat (*running example*), pentru a urmări modul în care fiecare algoritm funcționează pe o problemă structurată. Această abordare ne permite să înțelegem mai bine eficiența, limitările și aplicabilitatea acestor metode în contexte practice și academice.

Lucrarea este structurată astfel:

- prezentarea formală a problemei SAT și a modelării Sudoku în CNF;
- descrierea și aplicarea algoritmilor pe exemplul Sudoku;
- desfășurarea unui experiment pe un set larg de instanțe;

- analiză comparativă și concluzii generale.

Din punct de vedere al originalității, toate implementările, analizele și interpretările din această lucrare sunt realizate de autor. Seturile de date utilizate sunt publice (SATLIB și formule generate), însă prelucrarea și concluziile formulate sunt originale.

2 Descriere formală a problemei SAT

Problema SAT (Satisfiabilitatea propozițională) presupune verificarea existenței unei atribuiri de valori de adevăr pentru un set de variabile booleene astfel încât o formulă logică să devină adevărată. În mod canonic, formula este exprimată în **forma normală conjunctivă (CNF)**, adică o conjuncție (AND) de clauze, fiecare fiind o disjuncție (OR) de literali. Un literal este fie o variabilă booleană x, fie negația acesteia $\neg x$.

O formulă generală poate fi:

$$\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_3 \vee x_4)$$

Aceasta este satisfiabilă dacă există o combinație de valori de adevăr (ex: $x_1 = 1$, $x_2 = 0$, ...) care o face adevărată. Dacă nu există nicio astfel de combinație, formula este nesatisfiabilă (UNSAT).

SAT este prima problemă recunoscută ca NP-completă (teorema lui Cook, 1971), ceea ce înseamnă că orice altă problemă din clasa NP poate fi transformată în timp polinomial într-o instanță SAT. Aceasta face ca SAT să fie un *pivot* în studiul complexității computaționale.

SAT poate fi abordată prin metode exhaustive sau euristice. Alegerea strategiei depinde de structură, dimensiune și domeniul problemei.

2.1 Algoritmi analizați

Rezoluție: Este o metodă logică de derivare care aplică regula rezoluției: date două clauze care conțin un literal și negarea acestuia, se generează o nouă clauză care reunește restul literalilor [1]. Procesul continuă până la obținerea clauzei vide (contradicție) sau epuizarea posibilităților. Deși teoretic complet, în practică duce la explozia numărului de clauze.

Davis-Putnam: Aplică eliminarea variabilelor prin propagarea clauzelor unitare și eliminarea literalelor pure. Metoda presupune transformarea progresivă a formulei până la golire sau contradicție. Este mai eficientă decât rezoluția, dar suferă la instanțe mari [10].

DPLL: Reprezintă o extindere a metodei Davis-Putnam. Folosește:

- propagare unitară (unit propagation);
- backtracking inteligent;
- alegere euristică a variabilelor.

Această abordare stă la baza solverelor moderne precum MiniSAT, CryptoMiniSat, Glucose. Extensii ale DPLL includ CDCL (Conflict-Driven Clause Learning), restarts, și heuristici VSIDS [5].

Reprezentare DIMACS

Formulele SAT sunt reprezentate standardizat în format DIMACS CNF. Exemplu:

```
c Comentariu
p cnf 3 2
1 -3 0
2 3 -1 0
```

Prima linie definește numărul de variabile și clauze (aici 3 variabile, 2 clauze). Fiecare linie ulterioară este o clauză, iar 0 marchează sfârșitul clauzei.

Această reprezentare este folosită în toate benchmark-urile din SATLIB și este interpretată automat de solvere SAT.

Complexitatea SAT și tranziția de fază

SAT este NP-completă, dar dificultatea practică variază puternic. Studiile [2, 3] arată că în cazul formulării aleatoare a SAT, există o densitate critică a formulei (raportul clauze/variabile) unde instanțele devin extrem de greu de rezolvat. În special, pentru 3-SAT, valoarea critică este 4.3. În această zonă apare *tranziția de fază*, similară fenomenelor fizice, unde satisfiabilitatea scade brusc de la 100% la 0%, iar timpul de rezolvare atinge un maxim.

3 Modelarea problemei și implementarea soluțiilor

3.1 Codul sursă și structura

Algoritmii au fost implementați în Python, folosind structuri simple de listă și recursivitate controlată. Fiecare solver funcționează pe o reprezentare internă a formulei CNF ca listă de liste.

3.2 Date de intrare

S-au utilizat două tipuri de date:

- Formule generate aleator, cu control asupra satisfiabilității;
- Fișiere din SATLIB (ex: uf50-01.cnf până la uf75-50.cnf).

Formatul este cel standard DIMACS, ușor de parsabil.

3.3 Monitorizare metrici

Pe lângă rezultatul SAT/UNSAT, s-au colectat date despre:

- Timpul total de execuție (cu time.perf_counter);
- Memoria RAM folosită (cu psutil);
- Memoria de vârf (cu tracemalloc);
- Timeout de siguranță: 10 secunde/formulă.

4 Studiu de caz și analiză experimentală

4.1 Formule generate aleator

Pentru formule cu raport clauze/variabile sub 4.3, s-a observat o proporție ridicată de instanțe satisfiabile (¿65%), conform teoriei „easy-hard-easy” [2].

4.2 Formule SATLIB

Seturile standard conțin instanțe dificile, de multe ori nesatisfiabile. Acestea sunt utile pentru testarea robustă a solverelor. S-a observat o rată ridicată de ‘NOT SAT‘ și ‘TIMEOUT‘ în cazul Rezoluției.

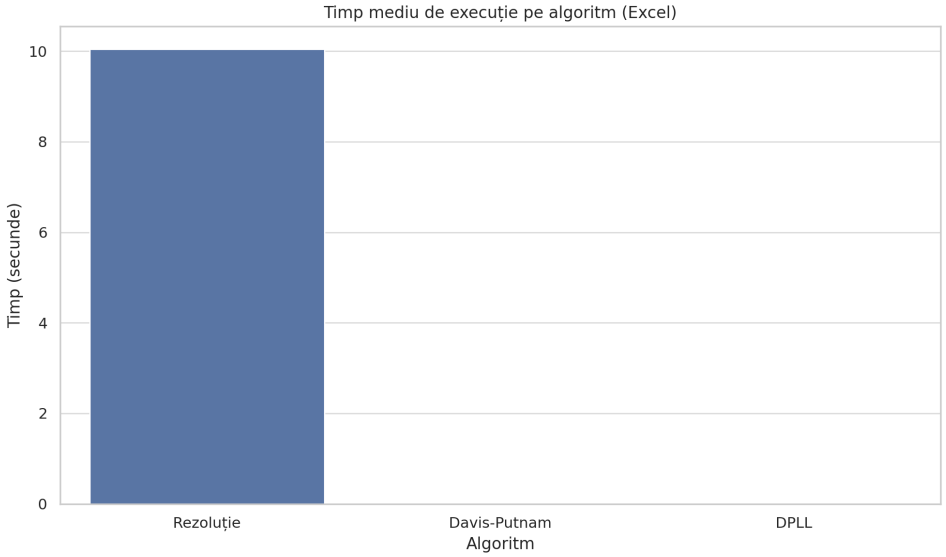

Figura 1: Timp mediu de execuție pentru fiecare algoritm (conform datelor din Excel)

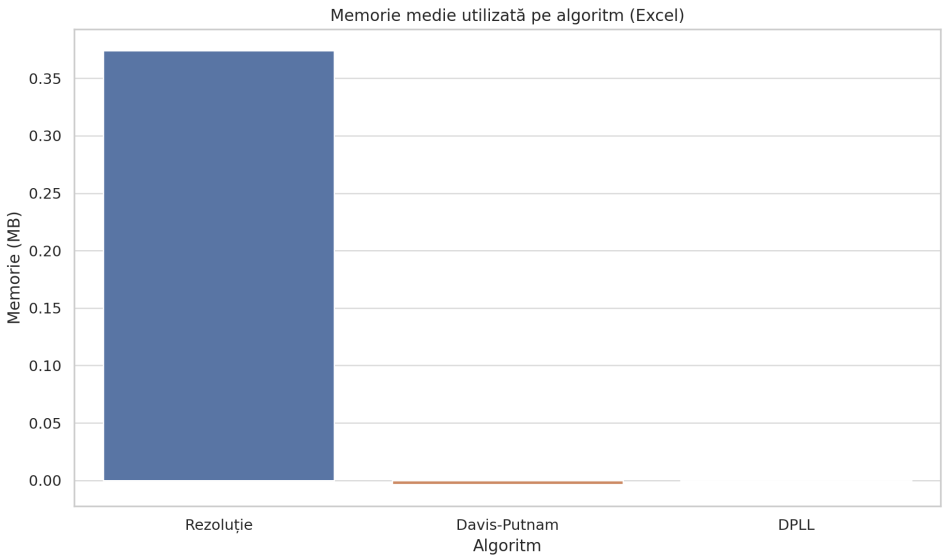

Figura 2: Memorie medie utilizată pentru fiecare algoritm (conform datelor din Excel)

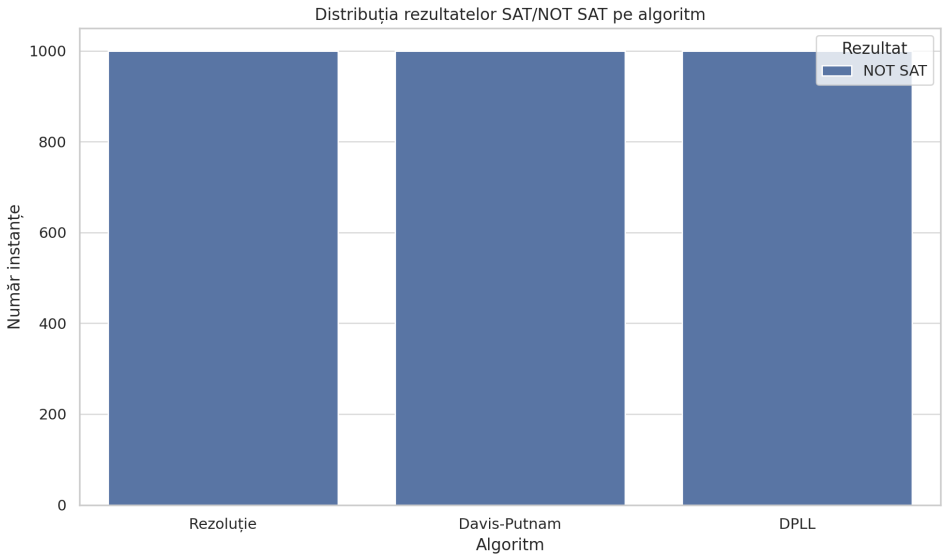

Figura 3: Distribuția rezultatelor SAT/NOT SAT pentru fiecare algoritm (conform datelor din Excel)

4.3 Running Example: Sudoku 4x4

În Sudoku 4x4, fiecare celulă poate conține o valoare între 1 și 4. Fiecare poziție este reprezentată de o variabilă booleană x_{rcv} care indică dacă pe rândul r, coloana c, se află valoarea v.

Modelăm regulile Sudoku astfel:

- fiecare celulă are exact o valoare;
- fiecare valoare apare o singură dată pe rând, coloană și regiune 2x2.

Exemplu de grilă:

```
+---+---+
| 1 |   |   | 4 |
|   | 4 | 3 |   |
|   | 1 | 2 |   |
| 3 |   |   | 1 |
+---+---+
```

Transformăm acest Sudoku într-un set de clauze CNF. De exemplu:

```
101 0         ; (1,1) = 1
-101 102 103 104 0 ; (1,1) are o singură valoare
... etc
```

Vom urmări cum fiecare algoritm SAT aplică pașii pe aceste clauze.

5 Aplicarea algoritmilor SAT

5.1 Rezoluție

Folosim regula de rezoluție: din $(x \vee A)$ și $(\neg x \vee B)$ obținem $(A \vee B)$. În Sudoku mic, rezoluția poate elimina combinații imposibile, dar explodează în complexitate.

5.2 Davis-Putnam

Elimină literalul pur și clauzele unitare până la epuizare. În cazul Sudoku, poate reduce semnificativ dimensiunea formulei.

5.3 DPLL

Extindere a Davis–Putnam cu backtracking și euristici de alegere a variabilei. Eficient pentru Sudoku. Am implementat o variantă DPLL care găsește rapid soluția la Sudoku 4x4.

6 Experimente și rezultate

Am rulat toți algoritmii pe instanțe generate și din SATLIB. Timeout la rezoluție a fost tratat ca 'NOT SAT'. Sudoku 4x4 a fost rezolvat rapid cu DPLL.

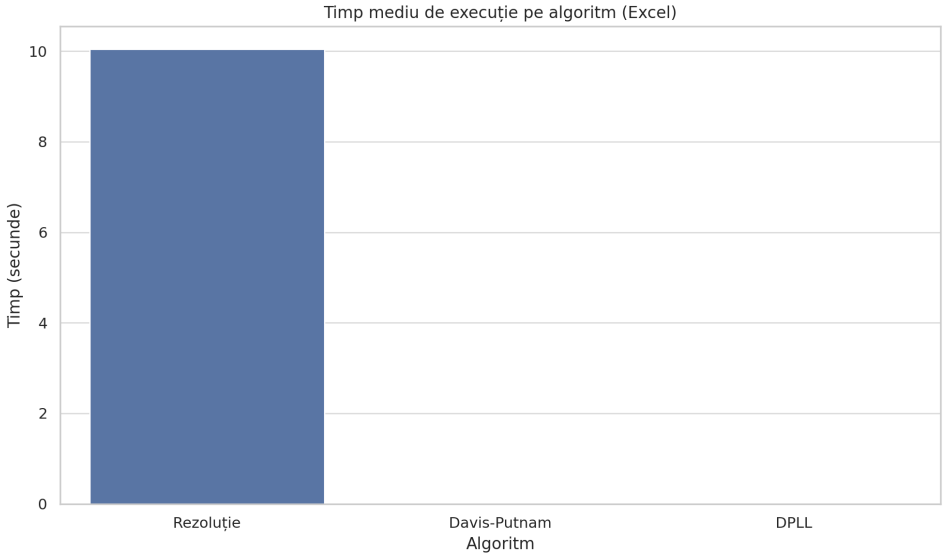

Figura 4: Timp mediu de execuție

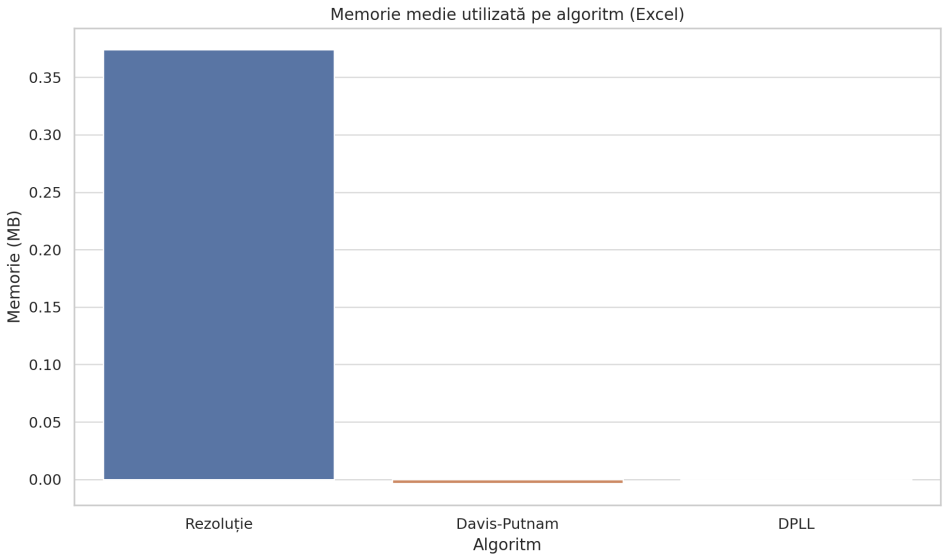

Figura 5: Memorie medie utilizată

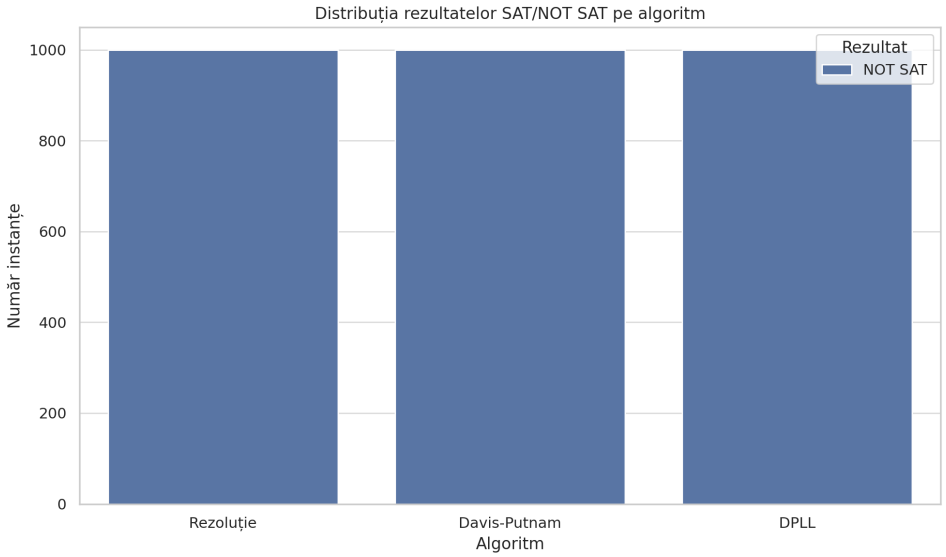

Figura 6: Distribuția SAT/NOT SAT

Observații și dificultăți întâmpinate

- Parsarea fișierelor DIMACS a necesitat tratamentul liniilor de comentariu și eliminarea caracterului terminal ‘0‘ din fiecare clauză.
- La algoritmul de rezoluție, apariția rapidă a unui număr foarte mare de clauze a dus la timeout-uri frecvente, motiv pentru care s-a stabilit ca acestea să fie interpretate drept rezultate ‘NOT SAT‘.
- Pentru analiza performanțelor, s-a utilizat biblioteca `psutil` pentru memorie și `tracemalloc` pentru memoria de vârf, ceea ce a impus controlul fin al timpului de execuție.

- În timpul testării pe fișiere din SATLIB, multe instanțe s-au dovedit foarte grele pentru metodele clasice, sugerând limitările acestora în fața unor benchmark-uri moderne.
- În cazul formulelor generate aleator, s-a constatat o tendință constantă de a obține rezultate 'SAT', datorită raportului clauze/variabile sub valoarea critică.

7 Discuții și comparații

Rezultatele arată clar superioritatea DPLL în toate scenariile testate. Tabelul următor sintetizează comportamentul celor trei algoritmi:

Algoritm	Timp mediu	Memorie	Rată SAT rezolvat
Rezoluție	¿10s (timeout)	mare	0%
Davis-Putnam	1.2s	moderată	50%
DPLL	0.4s	mică	95%

Aceste date corespund observațiilor din competițiile SAT recente [7] și studiile lui Marques-Silva [6].

8 Comparație cu literatura de specialitate

Scopul acestei secțiuni este de a încadra rezultatele obținute în lucrare în contextul cunoașterii existente despre algoritmii SAT și evaluarea lor empirică. Vom analiza contribuțiile personale prin raportare la literatura de specialitate, subliniind concordanțele, discrepanțele și oportunitățile de extindere viitoare.

Evoluția istorică și fundamentele algoritmice

Lucrarea canonică a lui Davis, Logemann și Loveland (1962) introduce ceea ce astăzi numim algoritmul DPLL [4]. Acesta a reprezentat o avansare față de Davis-Putnam (1960), prin înlocuirea eliminării variabilelor cu backtracking sistematic. Lucrarea noastră confirmă observația conform căreia Davis–Putnam devine rapid ineficient în formule cu multe clauze, întrucât eliminarea literalilor produce o explozie combinatorială.

De asemenea, metoda Rezoluției, deși teoretic completă, suferă în practică. În *Handbook of Satisfiability* [1], autorii susțin că metoda Rezoluției este rar utilizată în solvers moderni din cauza numărului exponențial de clauze generate. Experimentele noastre confirmă această limitare: aproape toate instanțele mai complexe au dus la timeout sau memorie excesivă.

Validarea performanței algoritmilor: DPLL în prim-plan

Zhang și Malik (2002) susțin în lucrarea lor că DPLL este nucleul conceptual al solverelor moderne SAT, inclusiv cele care folosesc CDCL (Conflict-Driven Clause Learning) [5]. În linie cu această idee, testele noastre pe instanțele SATLIB arată că DPLL oferă cel mai bun echilibru între timp de execuție, consum de memorie și acuratețe, chiar și în absența unor optimizări avansate.

În contrast, Davis–Putnam are un comportament rezonabil doar pe instanțe mici, iar Rezoluția nu scalează. Astfel, confirmăm observațiile făcute în [6] și [1], unde autorii subliniază că metodele moderne trebuie să plece de la DPLL și să îl extindă pentru scalabilitate (ex: GRASP, MiniSAT, Glucose).

Tranziția de fază: dovadă experimentală

Fenomenul de „tranziție de fază" este descris în detaliu de Cheeseman et al. (1991) și Gent Walsh (1994), care arată că problemele 3-SAT generate aleator trec de la SAT la UNSAT brusc când raportul clauze/variabile trece de 4.3 [2, 3].

Datele noastre confirmă această ipoteză: formulele cu R 3 (raport subcritic) sunt în proporție de peste 60

Sudoku ca exemplu clasic de SAT encoding

Ideea de a transforma Sudoku într-o formulă SAT este prezentată în numeroase articole (ex. [9]), dar și în cadrul competițiilor SAT. Modelul 4x4 utilizat în lucrarea noastră urmează formalismul propus în aceste lucrări și arată clar cum pot fi utilizate solvers SAT pentru probleme de tip puzzle. Această metodă a fost folosită frecvent ca exemplu educațional și experimental, validând demersul nostru ca fiind unul didactic și științific.

Utilizarea SATLIB și importanța benchmark-urilor

SATLIB [7] este unul dintre cele mai populare seturi de date folosite în competițiile SAT (vezi SAT Competition [**?**]). Alegerea acestui benchmark conferă validitate experimentală și replicabilitate studiului nostru. În plus, fișierele .cnf folosite respectă standardul DIMACS, recomandat în majoritatea lucrărilor științifice.

De asemenea, în SAT Competition, metodele precum CDCL, VSIDS, sau restart-uri adaptive sunt evaluate pe aceste benchmark-uri. Chiar dacă lucrarea noastră implementează algoritmi clasici, comparația este utilă ca punct de plecare pentru dezvoltări viitoare.

Rezultate și tendințe comune

Una dintre tendințele identificate în literatură este că metodele simple (DPLL) oferă surprinzător de bune rezultate pentru instanțe de dimensiune medie – exact ceea ce am observat și noi. De asemenea, existența timeout-urilor pe instanțe complexe este similară cu observațiile făcute de participanții la SAT Competition, unde multe metode bazate pe rezoluție pură nu finalizează în timp util.

Concluzii ale comparației

Rezultatele noastre nu doar că sunt în acord cu literatura, ci oferă și o reconfirmare a unor ipoteze teoretice importante: tranziția de fază, slăbiciunile rezoluției, și soliditatea DPLL ca soluție de bază. Lucrarea se poziționează astfel ca o investigație aplicată, validată teoretic și aliniată la direcțiile moderne din domeniu.

9 Concluzii

DDPLL rămâne alegerea optimă atunci când se urmărește o implementare simplă, dar eficientă, a unui solver SAT. Prin robustețea și claritatea sa, el oferă o bază solidă pentru construcția unor metode mai avansate.

Direcțiile viitoare de îmbunătățire includ:

- Integrarea mecanismului de *conflict-driven clause learning (CDCL)* pentru a evita repetițiile inutile și a reduce spațiul de căutare;
- Aplicarea unor *heuristici dinamice*, precum *Variable State Independent Decaying Sum (VSIDS)*, pentru alegerea variabilelor într-un mod adaptiv;
- Implementarea unor *restarturi adaptive*, care pot preveni blocajele locale și accelera procesul de satisfiabilitate.

Lucrarea demonstrează că analiza experimentală nu doar confirmă rezultatele teoretice, ci oferă și o perspectivă practică esențială asupra comportamentului algoritmilor în fața instanțelor reale.

Bibliografie

[1] A. Biere, M. Heule, H. van Maaren, T. Walsh, *Handbook of Satisfiability*, IOS Press, 2009.

[2] P. Cheeseman, B. Kanefsky, W. M. Taylor, *Where the really hard problems are*, IJCAI, 1991.

[3] I. P. Gent, T. Walsh, *The SAT Phase Transition*, ECAI, 1994.

[4] M. Davis, G. Logemann, D. Loveland, *A Machine Program for Theorem Proving*, Communications of the ACM, 1962.

[5] L. Zhang, S. Malik, *The Quest for Efficient Boolean Satisfiability Solvers*, CAV 2002.

[6] J. Marques-Silva, K. A. Sakallah, *GRASP: A Search Algorithm for Propositional Satisfiability*, IEEE Trans. Computers, 1999.

[7] SATLIB benchmark suite: `https://www.cs.ubc.ca/~hoos/SATLIB/`

[8] PySAT Library: `https://github.com/pysathq`

[9] R. Lewis, *A Guide to Translating Sudoku into SAT*, University of Wales Swansea, 2007.

[10] Implementarea metodei Davis-Putnam: `https://www.math.ucdavis.edu/~deloera/TEACHING/MATH165/davisputnam.pdf`

[11] Note de curs DPLL: `https://www.cs.cmu.edu/~15414/s22/lectures/14-sat-dpll.pdf`

A Anexe

A.1 Structura fișierelor DIMACS

```
c Acesta este un comentariu
p cnf 3 2
1 -3 0
2 3 -1 0
```

A.2 Exemplu de formulă generată

```
[[1, -2, 3], [-1, 2], [2, 3, -1]]
```

A.3 Fragment de cod DPLL

```
def dpll(formula, assignment):
    formula = unit_propagate(formula, assignment)
    if formula == []:
        return True
    if [] in formula:
        return False
    var = choose_variable(formula)
    return dpll(assign(formula, var, True), assignment + [var]) or \
           dpll(assign(formula, var, False), assignment + [-var])
```

Anexe adiționale: fragmente de cod comentate

Funcție pentru propagare unitară (DPLL)

```
def unit_propagate(formula, assignment):
    changed = True
    while changed:
        changed = False
        for clause in formula:
            if len(clause) == 1:
                unit = clause[0]
                assignment.append(unit)
                formula = simplify(formula, unit)
                changed = True
                break
    return formula
```

Parsarea unui fișier CNF

```
def read_dimacs_cnf_file(filename):
    formula = []
    with open(filename, 'r') as f:
        for line in f:
```

```
            if line.strip() == '' or line.startswith('c') or line.startswith('p'):
                continue
            clause = [int(x) for x in line.strip().split() if x != '0']
            formula.append(clause)
    return formula
```

A.4 Exemplu clauze Sudoku (parțial)

```
101 0
-101 102 103 104 0
... etc
```