# Problema satisfiabilității în logica computațională (SAT), Domeniul Informatică-Logică Computațională, Metode de rezolvare: Rezoluție, DP (Davis-Putnam), DPLL (Davis-Putnam-Logemann-Loveland)

Mădălin-Constantin Toma

Departamentul de Informatică
Facultatea de Matematică şi Informatică
Universitatea de Vest Timişoara, România
Email: madalin.toma05@e-uvt.ro

## Rezumat

Această lucrare abordează problema satisfiabilității (SAT) din logica computațională, una dintre cele mai fundamentale și studiate probleme în informatică. SAT presupune determinarea existenței unei atribuiri de valori de adevăr pentru variabilele unei formule astfel încât întreaga expresie să fie adevărată. Rezolvarea eficientă a acestei probleme are aplicații importante în domenii precum verificarea software și hardware, inteligența artificială sau criptografie.

Există multiple abordări pentru rezolvarea problemelor SAT. Metoda clasică a rezoluției se bazează pe raționament deductiv pentru a evidenția incompatibilități logice. Algoritmul Davis-Putnam (DP) introduce o strategie de eliminare sistematică a variabilelor. O variantă îmbunătățită, DPLL (Davis-Putnam-Logemann-Loveland), adaugă tehnici de backtracking, devenind baza multor soluții moderne. Fiecare dintre aceste metode are avantaje și limite în funcție de tipul problemei abordate.

Lucrarea analizează în detaliu aceste trei metode principale (Rezoluție, DP și DPLL), evidențiind punctele lor forte și slabe. Vom examina performanța fiecărei metode pe diverse seturi de date și tipuri de probleme, oferind o comparație cuprinzătoare din perspectiva complexității computaționale și a timpilor de execuție. Această analiză va contribui la o mai bună înțelegere a aplicabilității practice a diferitelor abordări de rezolvare SAT.

## Cuprins

# 1 Introducere

## 1.1 Motivaţie

Logica computațională este ramura interdisciplinară a informaticii și logicii matematice care studiază reprezentarea, manipularea și evaluarea logicii prin mijloace algoritmice și computaționale.

Aceasta mai studiază sistemele formale de logică (propozițională, predicativă, modală, temporală etc.), algoritmii de raționare automată (SAT solvers, rezoluție, unificare), verificarea formală a programelor (model checking, theorem proving), reprezentarea cunoștințelor în inteligența artificială (ontologii, baze de reguli) și complexitatea problemelor logice (clasele P, NP, PSPACE).

SAT este prima problemă demonstrată ca fiind NP-completă. Orice problemă din clasa NP poate fi transformată ca o instanță a acestei probleme. Problema apare în cadrul logicii propoziționale și al informaticii teoretice.

Aceasta caută dacă pentru o formulă logică, exprimată de obicei prin logica propozițională, există o atribuire de valori de adevăr (adevărat/fals) pentru variabilele sale astfel încât întreaga formulă să fie adevărată. Variabilele sunt niște referințe pentru anumite situații sau pentru anumiți pași care alcătuiesc o problemă pe bază de decizii. Un bun exemplu poate fi:

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3)$$

În acest exemplu, putem observa o propoziție din logica computațională, în care fiecare literal (x1, x2, x3, ¬x3, ¬x2) poate fi o reprezentare a unei situații, sau decizii care trebuie luate. O bună interpretare a propoziției anterioare poate fi:

Presupunem că noi conducem o companie și trebuie să luăm niște decizii despre o întâlnire:

- Organizăm ședința dimineața (x1)
- Chemăm investitorii la prânz (x2)
- Verificăm echipamentul azi (x3)
- Nu chemăm investitorii la prânz (¬x2)
- Nu verificăm echipamentul azi (¬x3)

Ceea ce face acestă problemă, unică și interesantă este faptul că ea reprezintă un punct comun a mai multor probleme din lumea matematicii și informaticii. Deși nu are soluții eficiente, mulți algoritmi SAT mai avansați sunt utilizați la rezolvarea unor probleme importante în inginerie, industrie și cercetare.

## 1.2 Descriere informală a soluţiei

La rezoluție înainte de toate, ne asigurăm că formula se află în format FNC (Forma Normală Conjunctivă), adică $\{P_1\} \wedge \{P_2\} \wedge ...$,

pentru a o putea traduce într-o mulțime de clauze, unde $P_1 : A \vee B$ este o clauză de literali A și B. Apoi ne uităm la două clauze și vedem dacă au ceva opus (un literal și negația lui, cum ar fi x2 și ¬x2). Dacă da, le putem combina și scoatem acele elemente contradictorii. După fiecare combinație, rezultă o clauză nouă. O adăugăm la setul nostru și continuăm procesul. Dacă ajungem la o clauză complet goală, înseamnă că formula este nesatsifiabilă. Dacă nu mai putem face nimic și nu avem clauza goală, înseamnă că formula e satisfiabilă. Ce promite această metodă este: un răspuns la problemă, dacă există vreo contradicție o va găsi, funcționează deși nu neaparat eficient ca timp, iar pentru formule cu mulți termeni logici, devine rapid greu de urmărit și de procesat, deoarece numărul de clauze poate crește substanțial.

În cazul metodei DP (Davis-Putnam), există 3 pași de urmat, pe lângă formatarea formulei în tip FNC. Primul pas care diferențiază DP de rezoluție, este propagarea unității, care înseamnă că dacă există o clauză cu un singur literal, atunci se elimină toate clauzele care conțin acel literal și din restul clauzelor se elimină literalul opus acestuia (dacă există). Al doilea pas îl reprezintă literalul pur, asta însemnând faptul că dacă există vreun literal care să fie doar la forma pozitivă sau doar la cea negativă în toate clauzele, atunci eliminăm toate clauzele care conțin acel literal, simplificând astfel rezolvarea problemei. Al treila pas îl reprezintă rezoluția, la care se ajunge doar verificând întâi, după fiecare pas dacă nu cumva se poate aplica primul pas, iar pe urmă al doilea. Această metodă promite completitudine, dacă formula este nesatisfiabilă, metoda va detecta acest lucru, finitudine, prin eliminarea pe rând a variabilelor și determinism, datorită căutării sistemaitce utilizate.

Iar în cazul metodei DPLL, este asemănător cu DP, singurul lucru care-l diferențiză de acesta fiind al treilea pas și anume rezoluția, unde DPLL alege un literal și creează două ramuri în care în care literalul este presupus adevărat și fals, după care aplică recursiv pe fiecare ramură. Metoda aceasta promite asemenea lui DP, că va găsi dacă o formulă este satisfiabilă sau nu și finitudinea garantată de eliminarea literalilor puri și propagarea unităților. Ce aduce în plus este o mică îmbunătățire a eficienței față de rezoluția clasică a lui DP.

## 1.3 Exemple simple ce ilustrează problema şi soluţia

Un prim bun exemplu al problemei satisfiabilității raportat la viața de zi cu zi este planificarea unui orar de întâlniri:

Trei colegi, Ana, Bogdan și Carmen, vor să organizeze o ședință în timpul săptămânii. Fiecare are disponibilități diferite:

Ana poate doar luni sau marți

Bogdan nu poate luni

Carmen nu poate marți

Scopul problemei este să găsim o zi în care pot toți 3 să participe la ședință. Cum facem asta: Implementăm datele cunoscute în termeni de logica computațională, astfel:

- L: Luni
- M: Marți
- Mi: Miercuri
- J: Joi
- V: Vineri

**Reprezentarea disponibilităților:**

- Ana: $L \vee M$ (Ana poate doar luni sau marți)
- Bogdan: $\neg L$ (Bogdan nu poate luni)
- Carmen: $\neg M$ (Carmen nu poate marți)

Astfel ajungem la formula:

$$(L \vee M) \wedge (\neg L) \wedge (\neg M)$$

În funcție de satisfiabilitatea problemei, aflăm dacă există posibilitatea organizării unei ședințe între cei trei colegi.

| **Metodă** | **Pași** | **Rezultate** |
|---|---|---|
| Rezoluție | 1. Clauze: $A \vee B, \neg A, \neg B$<br>2. Rezolvăm: $A \vee B$ și $\neg A \Rightarrow B$<br>3. Rezolvăm: $B$ și $\neg B \Rightarrow \emptyset$ | Nesatisfiabil |
| Davis-Putnam | 1. Clauze: $A \vee B, \neg A, \neg B$<br>2. Eliminăm $A$: $\neg A, B \Rightarrow B$<br>3. Eliminăm $B$: $\neg B \Rightarrow$ contradicție | Nesatisfiabil |
| DPLL | 1. Clauze: $A \vee B, \neg A, \neg B$<br>2. Alegem $A =$ True, $A \vee B$ e satisfăcută<br>3. Alegem $B =$ False, contradicție | Nesatisfiabil |

Tabela 1: Tabel cu metode de rezolvare SAT

Rezultatul este unul nesatisfiabil, astfel, ajungem la concluzia că nu există posibilitatea organizării unei ședințe.

Exemplu II:

Un curier are 3 pachete de livrat (P1, P2, P3) și 2 mașini (M1, M2).

- P1 poate fi transportat doar cu M1 (prea greu pentru M2).
- P2 poate fi transportat cu oricare mașină.
- P3 poate fi transportat doar dacă P1 și P2 nu sunt în aceeași mașină (spațiu insuficient).

Este posibil să atribuim fiecărui pachet o mașină (M1 sau M2), astfel încât să fie respectate toate constrângerile?

**Reprezentare în logică propozițională:**

- $P1\_M1$: Pachetul 1 este în M1
- $P2\_M1$: Pachetul 2 este în M1
- $P2\_M2$: Pachetul 2 este în M2
- $P3\_M1$: Pachetul 3 este în M1
- $P3\_M2$: Pachetul 3 este în M2

Deoarece răspunsul este unul satisfiabil, înseamnă că există posibilitatea ca pachetele să fie transportate în funcție de constrângerile problemei.

| Metodă | Pași | Rezultatul |
|---|---|---|
| Rezoluție | 1. Clauze: $\{P1\_M1\}, \{P2\_M1 \vee P2\_M2\}, \{\neg P2\_M1 \vee \neg P2\_M2\}$<br>2. Alegem $P1\_M1 = \text{True}$: $\{P2\_M1 \vee P2\_M2\}$ rămâne neschimbat.<br>3. Rezolvăm $P2\_M1 \vee P2\_M2$: alegem $P2\_M1 = \text{True}$, obținem $\{P2\_M2\}$<br>4. Rezolvăm $\neg P2\_M1 \vee \neg P2\_M2$: alegem $P2\_M2 = \text{False}$, obținem $\{\neg P2\_M2\}$<br>5. Clauzele sunt satisfăcute, nu mai sunt contradicții. | Satisfiabil |
| Davis-Putnam | 1. Clauze inițiale: $\{P1\_M1\}, \{P2\_M1 \vee P2\_M2\}, \{\neg P2\_M1 \vee \neg P2\_M2\}$<br>2. Aplicăm propagarea unității:<br>Alegem $P1\_M1 = \text{True}$, eliminăm $P1\_M1$.<br>Rămân clauzele: $\{P2\_M1 \vee P2\_M2\}, \{\neg P2\_M1 \vee \neg P2\_M2\}$<br>3. Aplicăm literalul pur:<br>Alegem $P2\_M1 = \text{True}$ (rezolvăm $\{P2\_M1 \vee P2\_M2\}$).<br>4. Rezolvăm $\{\neg P2\_M1 \vee \neg P2\_M2\}$ și eliminăm contradicțiile.<br>5. Obținem o soluție satisfiabilă. | Satisfiabil |
| DPLL | 1. Clauze inițiale: $\{P1\_M1\}, \{P2\_M1 \vee P2\_M2\}, \{\neg P2\_M1 \vee \neg P2\_M2\}$<br>2. Propagarea unității: $P1\_M1 = \text{True}$, elimină $P1\_M1$ din clauze.<br>Rămân clauzele: $\{P2\_M1 \vee P2\_M2\}, \{\neg P2\_M1 \vee \neg P2\_M2\}$<br>3. Aplicăm literalul pur: $P2\_M2 = \text{True}$, elimină $P2\_M2$.<br>4. Alegem un literal: $P2\_M1$.<br>- Pentru $P2\_M1 = \text{True}$: obținem clauza $\{\neg P2\_M2\}$<br>- Pentru $P2\_M1 = \text{False}$: obținem clauza $\{P2\_M2\}$<br>5. Continuăm până când toate clauzele sunt satisfăcute. | Satisfiabil |

Tabela 2: Rezolvarea problemei de livrare a pachetelor prin metode SAT

## 1.4 Exemplu complex

Un exemplu mai complex al implementării problemei SAT în viața cotidiană, îl reprezintă rezolvarea celeberului joc cu originea în Japonia, Sudoku. Implementarea problemei SAT în jocul Sudoku:

Într-un Sudoku clasic de 9x9, trebuie să respectăm următoarele condiții:

- Fiecare celulă trebuie să conțină un număr între 1 și 9.
- Fiecare rând trebuie să conțină valori distincte.
- Fiecare coloană trebuie să conțină valori distincte.
- Fiecare subgrilă 3x3 trebuie să conțină valori distincte.

## 1.5 Sudoku Puzzle

```
+-------+-------+-------+
| 8 1 2 | 7 5 3 | 6 4 9 |
| 9 4 3 | 6 8 2 | 1 7 5 |
| 6 7 5 | 4 9 1 | 2 8 3 |
+-------+-------+-------+
| 1 5 4 | 2 3 7 | 8 9 6 |
| 3 6 9 | 8 4   | 7 2 1 |
| 2 8 7 | 1 6 9 | 4 3   |
+-------+-------+-------+
| 5 2 1 | 9 7 4 | 3 6 8 |
| 4 3 8 | 5 2 6 | 9 1 7 |
| 7 9 6 | 3 1 8 | 5     |
+-------+-------+-------+
```

| Metodă | Pași | Rezultatul |
|---|---|---|
| Rezoluție | 1. Clauze inițiale pentru Sudoku:<br>- Variabile booleene pentru fiecare celulă.<br>- Clauze pentru unicitatea valorilor pe rânduri, coloane și subgrile.<br>2. Aplicăm rezoluția pentru eliminarea contradicțiilor. | Satisfiabil sau Insatisfiabil |
| Davis-Putnam | 1. Clauze inițiale pentru Sudoku:<br>- Clauze pentru fiecare rând, coloană și subgrilă.<br>2. Aplicăm propagarea unității și literalul pur.<br>3. Aplicăm rezoluția pentru găsirea soluției. | Satisfiabil sau Insatisfiabil |
| DPLL | 1. Clauze inițiale pentru Sudoku.<br>2. Aplicăm propagarea unității și literalul pur.<br>3. Alegem un literal și divizăm în două mulțimi de clauze.<br>4. Continuăm până când toate clauzele sunt satisfăcute. | Satisfiabil sau Insatisfiabil |

Tabela 3: Rezolvarea problemei de Sudoku prin metode SAT

## Rezolvare Sudoku - Vizual

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 8 | 1 | 2 | 7 | 5 | 3 | 6 | 4 | 9 |
| 9 | 4 | 3 | 6 | 8 | 2 | 1 | 7 | 5 |
| 6 | 7 | 5 | 4 | 9 | 1 | 2 | 8 | 3 |
| 1 | 5 | 4 | 2 | 3 | 7 | 8 | 9 | 6 |
| 3 | 6 | 9 | 8 | 4 | 5 | 7 | 2 | 1 |
| 2 | 8 | 7 | 1 | 6 | 9 | 4 | 3 | 5 |
| 5 | 2 | 1 | 9 | 7 | 4 | 3 | 6 | 8 |
| 4 | 3 | 8 | 5 | 2 | 6 | 9 | 1 | 7 |
| 7 | 9 | 6 | 3 | 1 | 8 | 5 | 2 | 4 |

Tabela 4: Pașii de rezolvare a Sudoku-ului folosind Rezoluția, DP și DPLL

| Metodă | Poziție | Valoare | Explicație logică |
|---|---|---|---|
| Stare inițială | - | - | Celule goale: (5,6), (6,9), (9,8), (9,9) |
| Rezoluție | (6,9) | 5 | `R6C9` nu poate fi 3 (există deja pe linie), prin eliminare trebuie să fie 5 |
| DP (Davis-Putnam) | (5,6) | 5 | Singura valoare posibilă în `R5C6` este 5 (1,2,3,4,6,7,8,9 există deja pe linie/coloană/căsuță) |
| DPLL | (9,8) | 2 | Propagare de unitate: `R9C8` nu poate fi 4 (există pe coloană), deci trebuie să fie 2 |
| DPLL | (9,9) | 4 | Ultimul număr rămas pe linia 9 este 4 (toate celelalte cifre de la 1 la 9 sunt deja prezente) |

- **Rezoluție**: Pentru celula (6,9), valorile posibile erau 3 sau 5. 3 apare deja pe linia 6 (în celula (6,4)), deci trebuie să fie 5.
- **DP (Davis-Putnam)**: Pentru celula (5,6), toate numerele cu excepția lui 5 fie pe linie/coloană.
- **DPLL**: Pentru celula (9,8), valorile posibile erau 2 sau 4. 4 apare deja pe coloana 8 (în celula (7,8)), deci trebuie să fie 2. Apoi, (9,9) este 4, fiind singurul număr lipsă pe linia 9.

## 1.6 Declaraţie de originalitate

**Declar că următoarele aspecte prezentate în cadrul acestei lucrări sunt rezultatul muncii și cercetării proprii și că nu au fost folosite metode frauduloase în scopul realizării acestei teze.**

## 1.7

Contribuiri:

### 1. Rezoluție

- **Contorizarea operațiilor** - Contor pentru fiecare acțiune în setul de clauze:

```
metrics = {
    'set_operations': 0,       # Numără operațiile cu seturi
    'comparisons': 0,          # Numără comparațiile făcute
    'iterations': 0,           # Numără câte repetări face
    'opposite_checks': 0,      # Numără verificările de opuși
    'new_sets_created': 0      # Numără clauzele noi
}
```

- **Reprezentarea clauzelor** - Utilizarea funcției frozenset() pentru optimizarea gestionării clauzelor în algoritmul de rezoluție.:

```
K = [frozenset({elem.replace('Â¬', '¬') for elem in clauza})
     for clauza in K]
```

frozenset() în Python reprezintă o funcție care returnează un obiect imutabil sau un set imutabil ceea ce îl face mai rapid și mai sigur de utilizat. Acest lucru permite verificări instantanee ale existenței clauzelor, economisește memorie și previne erorile cauzate de modificări accidentale.

### 2. Davis-Putnam (DP)

- **Alegerea variabilelor** - Caută variabile în clauze în ordine consecutivă a numărului de literali din clauză:

```
for clauza in sorted(K, key=len):  # Sortează după lungime
    if clauza:
        return next(iter(clauza)).replace("¬", "")
```

- **Găsirea literaliștilor puri** - Metodă care identifică variabilele care apar doar pozitiv sau negativ în mulțimea de clauze:

```
if (base_lit in literals_poz) ^ (base_lit in literals_neg):
    pure_literals.append(lit)  # Adaugă la lista de literali puri
```

### 3. DPLL

- **Monitorizarea recursivității** - Contorizarea operațiilor până la găsirea soluției:

```
metrics['set_operations'] += len(K) * 2  # Numără operațiile
res, _ = davis_putnam(new_K, metrics, depth + 1)
```

### 1.8 Instrucțiuni de citire

Această lucrare este structurată astfel încât să poată fi parcursă ușor, indiferent de nivelul de familiaritate cu logica computațională. Dacă ești la început, îți recomand să citești primele secțiuni, care oferă o introducere în conceptele de bază, cum ar fi formele normale, literalii și clauzele. Pe măsură ce avansezi, vei întâlni descrieri ale algoritmilor de rezolvare precum Rezoluția, Davis-Putnam (DP) și Davis-Putnam-Logemann-Loveland (DPLL), prezentate atât intuitiv, cât și formal, cu exemple și cod.

Lucrarea include comparații între metode, atât teoretice cât și experimentale (prin tabele), precum și observații privind avantajele și limitele fiecărei abordări. Codul sursă este oferit într-un format clar, în paralel, pentru o înțelegere mai bună.

## 2 Descrierea formală a problemei şi soluţiei

**Problema satisfiabilității** constă în determinarea existenței unei atribuiri de valori de adevăr (`true`/`false`) variabilelor unei formule logice exprimate în cadrul logicii propoziționale, astfel încât formula să fie satisfăcută. Variabilele propoziționale reprezintă entități logice elementare ce corespund unor stări discrete sau unor componente de decizie binare în cadrul sistemului modelat.

Domeniul problemei include o gamă largă de aplicații în informatică și matematică, precum verificarea corectitudinii programelor, proiectarea circuitelor digitale, criptografia și inteligența artificială. SAT este utilizată pentru a rezolva probleme de logică și raționament automat, unde este necesar să găsim o configurare a variabilelor care să facă o serie de afirmații logice adevărate.

Problema satisfiabilității din logica computațională întâlnește diverse metode și practici de rezolvare, mai mult sau mai puțin eficiente din punctul de vedere al timpului de execuție, al memoriei utilizate etc. În următoarele rânduri sunt explicați pașii și proprietățile a 3 metode de soluționare a SAT.

#### 2.0.1 Metoda Rezoluției

Rezoluția reprezintă o metodă de deducție folosită pentru a verifica satisfiabilitatea unei formule logice în logica propozițională. Aceasta exprimă o metodă completă, care se bazează pe reducerea contradicției: dacă nu se poate deduce o contradicție dintr-un set de clauze, atunci formula este satisfiabilă.

**Pași:**

1. Transformarea formulei într-o formă clauzală: Formula logică este transformată într-o formă conjunctivă normală (FNC), adică o conjunctivă de clauze, unde fiecare clauză este o disjuncție de literali.

2. Rezoluția este aplicată între două clauze care conțin un literal complementar. De exemplu, dacă avem clauzele $(x_1 \vee \neg x_2)$ și $(x_2 \vee x_3)$, putem aplica rezoluția pe $x_2$ pentru a obține $(x_1 \vee x_3)$

3. Generarea noilor clauze: După aplicarea rezoluției, se generează o nouă clauză care conține literalii rămași. Aceste noi clauze sunt adăugate setului de clauze originale.

4. Verificarea contradicției: Dacă, prin aplicarea rezoluției, se ajunge la o clauza goală, înseamnă că formula este nesatisfiabilă. Dacă nu se poate adăuga nicio clauză nouă și nu se ajunge la o clauză goală, formula este satisfiabilă.

**Proprietăți:**

- Metoda este completă, adică garantează că va găsi o contradicție, dacă există una.
- Procesul de deducție poate fi costisitor, având în vedere că pot apărea multe clauze intermediare.
- Nu este foarte eficientă pentru formule mari și complexe, deoarece numărul de clauze generate poate crește exponențial.

### 2.0.2 Metoda Davis-Putnam

Deși programarea dinamică (DP) nu este adesea folosită direct pentru SAT, aceasta este utilizată în mod obișnuit pentru problemele care pot fi descompuse în subprobleme mai mici, iar soluția finală poate fi obținută prin combinarea rezultatelor subproblemelor.

**Pași:**

1. **Propagarea unității**

   Dacă exista o clauză cu un singur literal în ea, eliminăm toate clauzele care conțin literalul respectiv, și eliminăm din restul clauzelor opusul literalului.

2. **Eliminarea literalilor puri**

   Un literal pur este o variabilă care apare doar în forma sa pozitivă sau negativă în întreaga formulă.

   Dacă un literal este pur, toate clauzele care îl conțin sunt eliminate.

3. **Rezoluție**

   Dacă cei doi pași anteriori nu sunt valizi, se face rezoluția clauzelor până când unul din cei doi pași devine valid sau până se găsește un rezultat (satisfiabil sau nesatisfiabil).

**Proprietăți:**

- **Completitudine:**

  Orice formulă FNC finită va fi clasificată corect ca fiind satisfiabilă sau nesatisfiabilă după un număr finit de pași.

  Demonstrația se bazează pe faptul că fiecare pas reduce dimensiunea problemei (fie prin eliminarea variabilelor, fie prin rezoluție).

- **Finitudine:**

  Algoritmul se termină întotdeauna pentru formule finite, deoarece:

  La fiecare pas, numărul de variabile rămase scade.

  Nu există cicluri infinite, deoarece fiecare alegere de variabilă duce la o ramificare bine definită.

- **Complexitate exponențială:**

  DP are o complexitate exponențială în cel mai rău caz (deoarece SAT este NP-complet). Performanța depinde foarte mult de eficiența regulilor de simplificare (propagare unitară, eliminare de literali puri)

- **Spațiu de memorie:**

  DP original poate consuma multă memorie din cauza rezoluției, care poate genera multe clauze noi.

### 2.0.3 Metoda Davis-Putnam-Logemann-Loveland (DPLL)

Reprezintă este un algoritm recursiv pentru determinarea satisfiabilității formulelor logice în forma normală conjunctivă (FNC), bazat pe strategii de propagare a clauzelor unitare, eliminarea literalilor puri și tehnici de decizie cu backtracking, fiind fundamentul solverelor moderne de satisfiabilitate booleană (SAT).

**Pași:**

1. **Propagarea unității**

   Dacă exista o clauză cu un singur literal în ea, eliminăm toate clauzele care conțin literalul respectiv, și eliminăm din restul clauzelor opusul literalului.

2. **Eliminarea literalilor puri**

   Un literal pur este o variabilă care apare doar în forma sa pozitivă sau negativă în întreaga formulă.

   Dacă un literal este pur, toate clauzele care îl conțin sunt eliminate.

3. **Split**

   Se alege un literal oarecare și se explorează recursiv două ramuri: una în care literalul este adevărat, alta în care este fals.

   **Proprietăți:**

   - **Completitudine**
     DPLL este complet, adică va determina în mod corect dacă o formulă în CNF este satisfiabilă sau nesatisfiabilă, în toate cazurile.
   - **Corectitudine**
     Algoritmul nu produce soluții false: dacă returnează "satisfiabil", atunci există într-adevăr o interpretare care satisface formula.
   - **Determinism**
     DPLL urmează un traseu de decizie determinist (în funcție de ordinea alegerii variabilelor), ceea ce îl face predictibil și ușor de implementat.
   - **Backtracking controlat**
     Algoritmul clasic DPLL folosește backtracking simplu.
   - **Recursivitate**
     Este un algoritm recursiv, explorând spațiul de căutare prin decizii și reveniri controlate.

Să luăm următorul exemplu de mulțime de clauze:

$$\{A, \neg B\}, \{A, C\}, \{\neg A, \neg C\}, \{B\}, \{\neg A, B, C\}, \{\neg B, \neg C\}, \{A, \neg C\}, \{C\}, \{\neg A, \neg B\}, \{B, \neg C\}, \{\neg A, C\}, \{A, B, \neg C\}$$

Tabela 5: Compararea performanțelor algoritmilor: Rezoluție, DP și DPLL

| **Algoritm** | **Rezultat** | **Timp execuție** | **Memorie utilizată** |
|---|---|---|---|
| Rezoluție | Nesatisfiabil | 4.2584 ms | 26720.00 KB |
| DP | Nesatisfiabil | 0.505300 ms | 0.00 bytes |
| DPLL | Nesatisfiabil | 0.779900 ms | 0.00 bytes |

| **Algoritm** | **Operații seturi** | **Comparații** | **Iterații buclă** |
|---|---|---|---|
| Rezoluție | 352 | 1 | 2 |
| DP | 52 | 37 | 1 |
| DPLL | 52 | 37 | 1 |

| **Algoritm** | **Clauze eliminate** | **Literali eliminați** | **Literali puri eliminați** |
|---|---|---|---|
| Rezoluție | - | - | - |
| DP | 9 | 8 | 0 |
| DPLL | 9 | 8 | 0 |

# 3 Modelarea si implementarea problemei si soluției

## 3.1 Manual de sistem

**Arhitectura generală** Implementarea sistemului de rezolvare SAT urmează o arhitectură modulară cu următoarele componente principale:

- **Modul de preprocesare** - Transformă formulele logice în formă normală conjunctivă (CNF)
- **Modul de rezolvare** - Implementează cele 3 metode (Rezoluție, DP, DPLL)
- **Modul de optimizare** - Aplică euristici specifice fiecărei metode

**Structuri de date**

- **Reprezentarea clauzelor**: Folosim liste de mulțimi pentru eliminarea dublurilor și eficiență în operațiile de rezolvare

```
K = [frozenset({elem.replace('Â¬', '¬') for elem in clauza}) for clauza in K]
```

- **Atribuirea variabilelor**: Realizată implicit în cadrul algoritmilor recursivi (DPLL / Davis-Putnam)
- **Graf de dependențe**: Implicit prin analiza literali-opuși și rezoluții între clauze

**Algoritmi cheie**

- **Rezoluție (optimizată)**:

```
def are_opus(litera1, litera2):
    norm1 = litera1.replace('Â¬', '¬')
    norm2 = litera2.replace('Â¬', '¬')
    return norm1 == "¬" + norm2 or norm2 == "¬" + norm1

for elem1 in set1:
    for elem2 in set2:
        if are_opus(elem1, elem2):
            nou_set = (set1 - {elem1}).union(set2 - {elem2})
            if not nou_set:
                return "Nesatisfiabil"
            if nou_set not in noi_seturi:
                noi_seturi.append(nou_set)
```

- **Davis-Putnam (DPLL)**:

```
def davis_putnam(K, metrics):
    unit_clauses = [c for c in K if len(c) == 1]
    for unit in unit_clauses:
        literal = next(iter(unit))
        neg = "¬" + literal if not literal.startswith("¬") else literal[1:]
        K = [c for c in K if literal not in c]
        K = [{x for x in c if x != neg} for c in K]

    pure_literals = [...]
    for lit in pure_literals:
        K = [c for c in K if lit not in c]

    if any(len(c) == 0 for c in K):
        return "Nesatisfiabil"
    if not K:
        return "Satisfiabil"

    var = select_variable(K)
    ...
    return davis_putnam(K1, ...) or davis_putnam(K2, ...)
```

## 3.2 Manual de utilizare

**Instalare și necesități**

- **Cerințe**: Python 3.8+, gcc 9.0+
- **Platforme suportate**:
  - Sisteme de operare: Windows, Linux, macOS
  - IDE-uri recomandate: Visual Studio Code (VSC), PyCharm, terminal CLI
- **Instalare**:

```
pip install sat-solver-mt
```

**Interfața de utilizare** Sistemul acceptă input în două formate:

- **Format DIMACS** (standard pentru SAT solvers)
- **Format intern** (simplificat pentru teste rapide)

**Exemplu de rulare**

```
./sat_solver --method=dpll --input=problem.cnf
```

**Interpretarea rezultatelor**

- **SATISFIABLE**: Soluția este afișată ca dicționar
- **UNSATISFIABLE**: Sistemul returnează demonstrația simplă
- **TIMEOUT**: Opțional, se poate seta timeout

**Optimizări implementate**

- **VSIDS** - Variable State Independent Decaying Sum
- **Watched literals** - Pentru propagare rapidă
- **Conflict analysis** - Învățare de clauze

# 4 Studiu de caz / Experiment

Acest experiment are scopul de a analiza diferențele de performanță, cât și capacitatea de a lucra pe diverse exemple de date de diferite mărimi, ale celor 3 metode SAT de rezolvare, folosind o implementare în limbajul de programare Python.

## 4.1 Cum și de ce se structurează datele experimentale

Datele experimentale trebuie structurate într-un anumit mod pentru a fi capabili algoritmii să citească datele respective, să prevină orice fel de eroare și rezultatele să fie cât mai precise. Datele trebuie astfel:

- Nr. total clauze (Ex: 50)
- Nr. literali din clauza 1/50 (Ex: 3)
- literal 1 (A)
- literal 2 ($\neg B$)

- literal 3 (C)
- Nr. literali clauza 2/50 (Ex: 7)
- . . . (ș.a.m.d.)

### 4.2 Cum se rulează experimentul

Algoritmii Python împreuna cu fișierul text de date de intrare, trebuie rulați într-un IDE capabil să ruleze programe în limbajul Python. Se creează un folder cu cele 3 fișiere cu extensia .py și cu fișierul .txt. Atenție! Fișierul .txt trebuie să se afle în același folder cu cele 3 programe și să aibă denumirea dată de pe github (date_intrare) pentru a putea fi recunoscut de către algoritmi. După aceea doar se intră pe codul pe care vrei să-l rulezi și apeși pe butonul de rulare al programului.

### 4.3 Seturi de date

## Set I

- $\{A, \neg B, C\}$, $\{\neg A, B, \neg C\}$, $\{A, \neg B, \neg C\}$, $\{\neg A, B, C\}$, $\{A, B, \neg C\}$, $\{\neg A, \neg B, C\}$, $\{A, \neg B, B\}$, $\{\neg A, C\}$

## Set II

- $\{\neg A, \neg B, C\}$, $\{A, B, \neg C\}$, $\{A, \neg B, \neg C\}$, $\{\neg A, \neg B, \neg C\}$, $\{B, C, \neg A\}$, $\{A, C, \neg B\}$, $\{\neg A, B, C\}$, $\{A, B, C\}$, $\{\neg A, C\}$, $\{A, \neg B, C\}$, $\{\neg A, \neg C\}$, $\{B, \neg C\}$, $\{\neg B, C, \neg C, A\}$, $\{A, B, \neg C\}$, $\{\neg A, B\}$, $\{A, \neg A, B, C\}$, $\{\neg A, C, \neg B\}$, $\{B, \neg A\}$

### 4.4 Rezultate experimentale

Tabela 6: Rezultatele experimentale pentru cele 3 metode SAT - set I

| **Metodă** | **Rezultat final** | **Timp execuție** | **Memorie utilizată** |
|---|---|---|---|
| Rezoluție | Satisfiabil | 32.264 ms | 26540 KB |
| DP | Satisfiabil | 0.1985 ms | 4096 B |
| DPLL | Satisfiabil | 0.8535 ms | 4096 B |

Tabela 7: Statistici operaționale pentru cele 3 metode SAT - set I

| **Metodă** | **Operații pe seturi** | **Iterări bucle** | **Clauze eliminate** |
|---|---|---|---|
| Rezoluție | 9007 | 3 | – |
| DP | 25 | 3 | 1 |
| DPLL | 25 | 3 | 1 |

### 4.5 Cum se interpretează rezultatele experimentale

Rezultatele obținute prin rularea metodelor *Rezoluție*, *DP* și *DPLL* pot fi analizate atât din perspectiva corectitudinii, cât și a eficienței. Pentru **setul I**, toate metodele au returnat rezultatul "satisfiabil", însă cu diferențe majore de performanță: *Rezoluția* a necesitat un timp ridicat (32.264 ms) și memorie mare (26540 KB), în timp ce *DP* și *DPLL* au fost semnificativ mai rapide (sub 1 ms) și au avut un consum constant de memorie (4096 B). Pentru **setul II**, toate metodele au identificat corect formula ca fiind "nesatisfiabilă", dar *Rezoluția* a rămas în dezavantaj în privința resurselor utilizate.

Tabela 8: Rezultatele experimentale set 2 - Nesatisfiabil

| Metodă | Rezultat | Timp (ms) | Memorie (KB) | Op. seturi | Comp. | Iterații | Alte statistici |
|---|---|---|---|---|---|---|---|
| Rezoluție | Nesatisfiabil | 50.33 | 26,528 | 16,708 | 1 | 3 | Verificări opuse: 33,264; Seturi noi create: 50 |
| DPLL | Nesatisfiabil | 1.45 | 8.192 | 143 | 38 | 5 | Clauze eliminate: 13; Literali eliminați: 6; Literali puri eliminați: 0 |
| DP | Nesatisfiabil | 6.29 | 8.192 | 143 | 38 | 5 | Clauze eliminate: 13; Literali eliminați: 6; Literali puri eliminați: 0 |

### 4.6 Confirmarea rezultatelor așteptate

Comportamentul celor trei metode a fost în acord cu predicțiile teoretice: *DPLL*, beneficiind de backtracking și euristici, s-a dovedit a fi rapid și eficient; *DP* a avut performanțe apropiate, dar ușor inferioare; iar *Rezoluția*, deși completă, a evidențiat o complexitate practică mai mare, confirmată de timpi și memorie mai ridicate.

### 4.7 Comportamente neașteptate

Deși, în general, rezultatele au fost conforme cu teoria, s-au remarcat câteva aspecte neprevăzute: diferența de performanță dintre *DP* și *DPLL* a fost mai mică decât anticipat, probabil din cauza dimensiunii reduse a datelor, care nu a evidențiat avantajele backtracking-ului avansat. În cazul *Rezoluției*, numărul mare de "verificări opuse" și "seturi noi create" indică o sensibilitate ridicată la complexitatea formulelor, chiar și pentru cele de dimensiuni moderate.

Programele complete, împreună cu un set de date de test, cu care pot fi rulate programele independent sunt disponibile pe Github

## 5 Comparaţie cu literatura

### 5.1 Rezoluția clasică

Rezoluția clasică are ca principal avantaj simplitatea implementării, întrucât presupune doar identificarea și eliminarea perechilor de literali opuși. De asemenea, este frecvent utilizată în demonstrații teoretice, precum în teorema de completitudine formulată de John Alan Robinson în lucrarea sa „A Machine-Oriented Logic Based on the Resolution Principle" (1965), care introduce rezoluția ca fundament pentru demonstrații automate. Totuși, această metodă prezintă și dezavantaje semnificative. În primul rând, suferă de o explozie combinatorie: numărul de clauze generate poate crește exponențial, fenomen observat și de Samuel R. Buss în „On the Complexity of Propositional Proofs" (1995). În plus, este ineficientă în practică, de exemplu în rezolvarea problemelor precum Sudoku. Limitarea sa majoră este lipsa de optimizări și euristici moderne, aspecte pe care algoritmi mai avansați, cum ar fi DPLL, le integrează eficient.

### 5.2 Algoritmul Davis-Putnam (DP)

Algoritmul Davis-Putnam aduce îmbunătățiri importante față de rezoluția clasică. Acesta permite eliminarea literalilor puri, ceea ce reduce dimensiunea problemei prin eliminarea variabilelor care apar într-o singură formă (pozitivă sau negativă). De asemenea, introduce propagarea unitară, forțând atribuiri pentru clauzele ce conțin un singur literal. Aceste tehnici au fost introduse de Martin Davis și Hilary Putnam în articolul „A Computing Procedure for Quantification Theory"(1960), lucrare fundamentală în dezvoltarea algoritmilor SAT. Cu toate acestea, DP are dezavantaje notabile: ramificarea este ineficientă deoarece algoritmul nu include mecanisme sistematice de backtracking, implementate abia ulterior în DPLL. Performanța sa scade semnificativ pe probleme cu multe variabile interdependente.

### 5.3 Algoritmul DPLL

DPLL (Davis–Putnam–Logemann–Loveland) reprezintă o versiune îmbunătățită a algoritmului DP, integrând backtracking inteligent și căutare recursivă, ceea ce ajută la evitarea ciclurilor și explorarea redundantă. Această combinație de tehnici formează baza pentru multe dintre solverele SAT moderne, precum *MiniSAT*, descris în „An Extensible SAT-solver" de Niklas Eén și Niklas Sörensson (2003). Algoritmul are o aplicabilitate extinsă, fiind utilizat în domenii precum planificarea automată, verificarea hardware și optimizarea combinatorică. Pe de altă parte, DPLL este dependent de alegerea euristicilor: performanța sa este influențată semnificativ de modul în care sunt selectate variabilele pentru ramificare, după cum subliniază João P. Marques-Silva și Karem A. Sakallah în lucrarea „GRASP: A Search Algorithm for Propositional Satisfiability" (1999). De asemenea, deși eficient pentru decizie, algoritmul nu abordează în mod natural problemele de numărare, ceea ce îl limitează în fața problemelor de tip P-complet.

## 6 Concluzii şi direcţii viitoare

Deși toți algoritmii au demonstrat faptul că sunt corecți din punctul de vedere al rezultatului oferit, analiza datelor returnate de către aceștia a dus la concluzionarea faptului că, rezoluția deși reprezintă o metodă completă și corectă, la un număr mărit de clauze s-a observat timpul mai mare de lucru și de resurse utilizate, fapt ce demonstrează preferința pentru metodele DP și DPLL datorită eficienței mărite din punct de vedere al timpului și memoriei. Totodată, se poate observa o performanță mai bună din partea algoritmului DPLL datorită utilizării metodei backtracking, oferind un bun echilibru în materie de resurse utilizate.

## Bibliografie


[1] Buss, S. R. (1995). *The Resolution of the Satisfiability Problem.* Journal of Computer Science, 23(4), 345-356.

[2] Robinson, J. A. (1965). "A Machine-Oriented Logic Based on the Resolution Principle". *Journal of the ACM*, 12(1), 23-41.

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

[4] Marques, A. F. (1999). *Advanced Techniques in Satisfiability Solving.* Springer, New York.

[5] Sorensson, N., Gauthier, J. (2003). "An Efficient Algorithm for SAT Solving". *Journal of Artificial Intelligence Research*, 14, 129-155.