DOI: https://doi.org/10.53486/sstc2026.v1.100

CZU: 330.43

MODELS OF COMPUTATION AND THEIR LIMITS: TURING MACHINES, NONDETERMINISM AND THE P VS NP HYPOTHESIS

MODELE DE CALCUL ȘI LIMITELE LOR: MAȘINI TURING, NONDETERMINISM ȘI IPOTEZA P VS NP

BORȘ Alexandru student, specialitatea: INFA
Academia de Studii Economice din Moldova, Republica Moldova,
Chișinău, str. Bănulescu-Bodoni 61, www.ase.md
e-mail autor: bors.alexandru.ptgt@ase.md

GURANDA Vadim student, specialitatea: INFA
Academia de Studii Economice din Moldova, Republica Moldova,
Chișinău, str. Bănulescu-Bodoni 61, www.ase.md
e-mail autor: guranda.vadim@ase.md

Abstract:
The article explores fundamental concepts of computational complexity theory, with a focus on the relationship between the complexity classes P and NP and the Boolean satisfiability problem (SAT) as a central NP-complete problem. Theoretical foundations are introduced through models of computation, particularly deterministic and nondeterministic Turing machines, which formalize the limits of efficient computation. A significant part of the work is dedicated to SAT, including its role in complexity theory and its practical relevance. Classical and modern solving techniques are presented, with emphasis on the DPLL algorithm and its extension, CDCL, which form the basis of modern SAT solvers. To illustrate the practical implications of these concepts, the graph coloring problem is used as a case study. It is approached from two perspectives: a reduction to SAT and a direct backtracking-based algorithm. The two methods are implemented and experimentally compared in terms of execution time on randomly generated graph instances with varying sizes and densities. The experimental results show that the direct approach generally outperforms the SAT-based formulation in the analyzed instances, while the SAT approach provides a more general theoretical framework. The study highlights the trade-off between generality and efficiency in algorithm design, as well as the practical challenges associated with NP-complete problems.

Key words: *Computational complexity, P vs NP, SAT, NP-complete, Turing machine, DPLL, CDCL, Graph coloring, Backtracking, Algorithm comparison.*

JEL CLASSIFICATION: C00, C60, C63

INTRODUCERE. “Să considerăm problema organizării cazării pentru 400 de studenți universitari, în condițiile în care sunt disponibile doar 100 de locuri. În plus, decanul furnizează o listă de perechi de studenți incompatibili, impunând ca nicio pereche din listă să nu apară simultan în selecția finală. Această situație constituie un exemplu clasic de problemă NP: verificarea unei selecții candidate este simplă, însă generarea unei selecții valide de la zero devine rapid impracticabilă pentru dimensiuni mari ale problemei.” („P vs NP”, f.a.)

Problema P vs NP reprezintă una dintre cele mai importante întrebări nerezolvate din știința calculatoarelor, având un grad de dificultate extrem de ridicat și o relevanță potențială care ar putea transforma fundamental lumea actuală. Aceasta face parte dintre cele șapte Probleme Millennium propuse de Clay Mathematics Institute, iar rezolvarea sa este recompensată cu un premiu de un milion de dolari. („The Millennium Prize Problems”, f.a.)

Importanța acestei probleme derivă din impactul său potențial asupra numeroaselor domenii, inclusiv criptografia, optimizarea, inteligența artificială, logistica și știința datelor. O soluție la

problema P vs NP ar influența fundamental modul în care sunt abordate problemele computaționale dificile în practică.

Din punct de vedere teoretic, problema explorează relația dintre clasele de complexitate P și NP: dacă orice problemă a cărei soluție poate fi verificată rapid poate fi și rezolvată rapid, atunci P = NP; în caz contrar, P ≠ NP.

Această lucrare își propune:

- analiza fundamentelor teoretice ale problemei SAT și ale claselor de complexitate P și NP;
- prezentarea modelelor de calcul utilizate în teoria complexității, în special mașinile Turing deterministe și nedeterministe;
- studierea algoritmilor de rezolvare a problemei SAT, cu accent pe DPLL și CDCL;
- implementarea și compararea a două abordări pentru problema colorării grafurilor: reducerea la SAT și un algoritm direct de tip backtracking;
- evaluarea experimentală a performanței celor două metode și analiza rezultatelor obținute.

În continuare, pentru a susține aceste obiective, lucrarea prezintă conceptele fundamentale care stau la baza teoriei complexității computaționale.

Algoritmi și complexitatea lor. Un algoritm este un set finit de reguli bine definite și neambigue, care oferă o succesiune de operații pentru rezolvarea unui anumit tip de problemă. (Knuth, 1982). În esență, un algoritm reprezintă un set de operații necesare pentru rezolvarea unei probleme. Pentru ca o succesiune de pași să fie considerată algoritm, aceasta trebuie să respecte anumite proprietăți fundamentale:

- Finitudinea: trebuie să se termine după un număr finit de pași.
- Determinarea (claritatea): fiecare pas trebuie să fie definit precis și fără ambiguități.
- Intrarea: poate avea zero sau mai multe valori de intrare.
- Ieșirea: trebuie să producă una sau mai multe valori de ieșire.
- Eficiența: operațiile trebuie să fie suficient de simple încât să poată fi executate de un om, cu creion și hârtie, într-un timp finit (Knuth, 1982).

Complexitatea algoritmilor. În analiza teoretică a algoritmilor, este obișnuit să se estimeze complexitatea în sens asimptotic, adică să se evalueze funcția de complexitate pentru dimensiuni foarte mari ale datelor de intrare. În informatică, complexitatea unui algoritm măsoară eficiența acestuia prin resursele consumate – în special timpul și spațiul – în raport cu volumul datelor de intrare. Pentru această analiză se utilizează notații precum Big O, Big-omega și Big-theta, dintre care notația BigO *(O(f(n))* (Cormen, 2009) este cea mai frecvent folosită, indicând cazul cel mai nefavorabil și concentrându-se pe termenii de ordin superior. Componentele complexității:

- Complexitatea în timp: numărul de operații elementare executate de algoritm.
- Complexitatea în spațiu: volumul de memorie necesar pentru stocarea datelor și variabilelor.

Dintre acestea, complexitatea temporală este, în general, considerată mai importantă, deoarece influențează direct performanța algoritmului în practică (Erickson, 2019).

Mașina Turing. Mașina Turing este un model matematic fundamental al calculului, utilizat pentru a descrie formal noțiunea de algoritm și proces de calcul. Conceptul a fost introdus de Alan Turing și stă la baza teoriei moderne a complexității (Sipser, 2013). O mașină Turing este alcătuită dintr-o bandă teoretic infinită, împărțită în celule care conțin simboluri, și un cap de citire/scriere care se deplasează pe bandă, modificând simbolurile conform unui set de reguli. La fiecare pas, mașina își schimbă starea în funcție de starea curentă și simbolul citit, realizând astfel o succesiune de operații elementare. Acest model este important deoarece permite definirea riguroasă a noțiunilor de timp și spațiu de calcul, care sunt esențiale în clasificarea problemelor în clase de complexitate.În mod clasic, se disting două tipuri principale de mașini Turing:

- Mașina Turing deterministă (MTD); Pentru fiecare pereche (stare curentă, simbol citit) există o singură tranziție posibilă. Astfel, execuția este complet determinată, similară rulării unui algoritm clasic pe un calculator.
- Mașina Turing nedeterministă (MTN); Pentru aceeași pereche (stare, simbol) pot exista mai multe tranziții posibile. Modelul „alege" teoretic o cale de execuție, iar problema este considerată rezolvată dacă există cel puțin o cale care duce la o stare de acceptare.

Mașinile nedeterministe nu reprezintă dispozitive fizice reale, ci instrumente teoretice utilizate pentru a defini clase de complexitate precum NP, unde soluțiile pot fi verificate eficient, chiar dacă nu sunt neapărat ușor de găsit (Cormen, 2009; Sipser, 2013). Pe baza acestor modele de calcul sunt definite clasele de complexitate prezentate în secțiunea următoare.

Clasele de complexitate. Clasele de complexitate reprezintă categorii de probleme computaționale, grupate în funcție de resursele necesare pentru rezolvarea lor, în special timpul de execuție și spațiul de memorie. Aceste clase sunt definite în funcție de tipul problemei (de regulă probleme de decizie), modelul de calcul utilizat (de exemplu, mașini Turing deterministe sau nedeterministe) și limitele impuse asupra resurselor (cum ar fi timp polinomial sau spațiu logaritmic). În acest context, o clasă de complexitate poate fi formalizată ca mulțimea problemelor care pot fi rezolvate de un anumit model de calcul în limite bine definite de resurse. Spre deosebire de analiza complexității unui algoritm individual (exprimată prin notații precum $O(n)$), clasele de complexitate descriu dificultatea intrinsecă a problemelor.

Clase fundamentale precum P și NP joacă un rol central în teoria complexității, alături de alte clase precum PSPACE, L sau EXPTIME, fiecare caracterizând diferite niveluri de eficiență computațională (Cormen, 2009; Sipser, 2013).

Unele dintre cele mai importante clase de complexitate sunt (Figura 1):

- **NL (Nondeterministic Logarithmic Space)** Clasa problemelor de decizie care pot fi rezolvate de o mașină Turing nedeterministă folosind spațiu logaritmic O(log n).
- **P (Polynomial Time)** Clasa problemelor de decizie care pot fi rezolvate în timp polinomial de o mașină Turing deterministă; este considerată clasa problemelor eficient rezolvabile.
- **NP (Nondeterministic Polynomial Time)** Clasa problemelor de decizie pentru care o soluție candidat poate fi verificată în timp polinomial de o mașină deterministă.
- **PSPACE (Polynomial Space)** Clasa problemelor care pot fi rezolvate folosind spațiu polinomial, indiferent de timpul de execuție.
- **EXPTIME (Exponential Time)** Clasa problemelor care pot fi rezolvate în timp exponențial de o mașină Turing deterministă.
- **EXPSPACE (Exponential Space)** Clasa problemelor care necesită spațiu exponențial pentru a fi rezolvate.

După cum se observă în Figura 1, aceste clase sunt organizate ierarhic, fiecare nivel reprezentând probleme din ce în ce mai dificil de rezolvat din punct de vedere computațional.

Problema satisfiabilității (SAT). Problema satisfiabilității booleene (SAT) este una dintre cele mai importante probleme din teoria complexității și constă în determinarea existenței unei atribuiri de valori de adevăr pentru variabilele unei formule booleene astfel încât aceasta să fie evaluată ca adevărată (Sipser, 2013).

O formulă booleană este o expresie construită din variabile booleene (care pot lua valori din mulțimea {0,1} sau {adevărat, fals}), operatori logici precum AND, OR și NOT, precum și paranteze. Pentru o anumită atribuire a variabilelor, formula poate fi evaluată fie la valoarea adevărat, fie la fals.

O formulă este satisfiabilă dacă există cel puțin o atribuire a variabilelor pentru care aceasta devine adevărată; în caz contrar, formula este nesatisfiabilă. De asemenea, o formulă poate fi:

- tautologie, dacă este adevărată pentru orice atribuire;
- contradicție, dacă este falsă pentru orice atribuire.

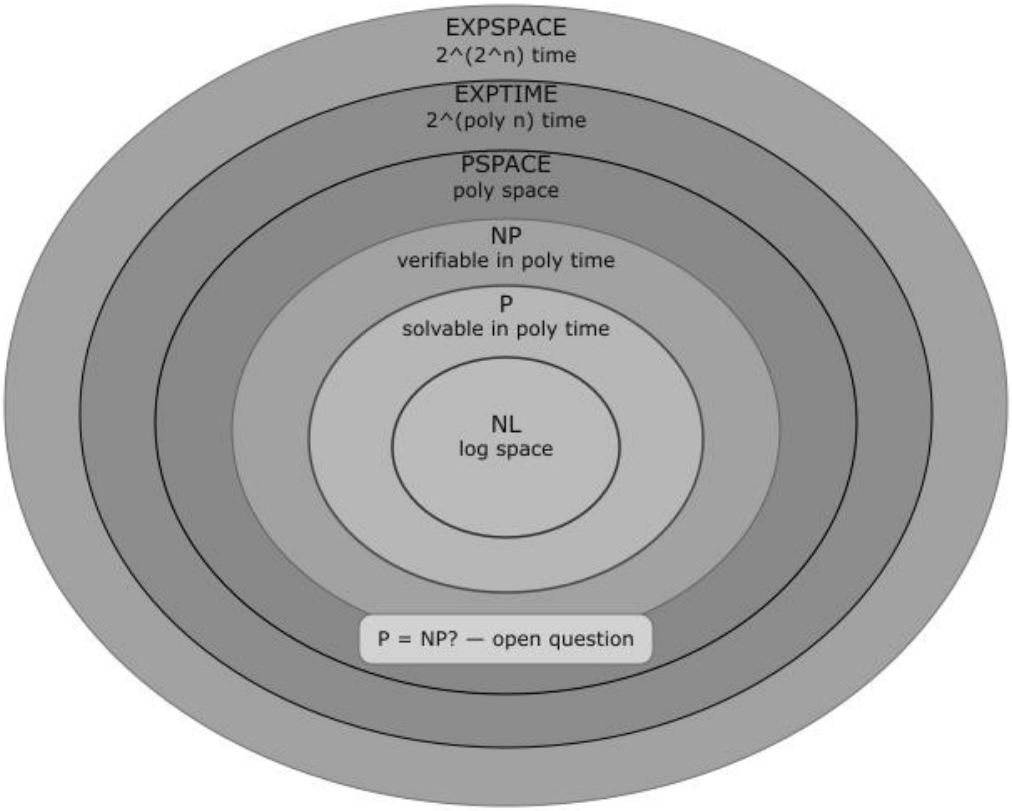


Figura 1. Clasele de complexitate importatnte
Sursa: Realizat de autori după (Sipser, 2013)

Formulele booleene pot fi transformate, folosind legi logice (de exemplu, legile lui De Morgan), în forme echivalente. În mod particular, orice formulă booleană poate fi adusă într-una dintre formele canonice:

- tautologie, dacă este adevărată pentru orice atribuire;
- forma normală conjunctivă (CNF);
- forma normală disjunctivă (DNF).

Forma normală conjunctivă (CNF) este deosebit de importantă în contextul problemei SAT, deoarece majoritatea algoritmilor și reducțiilor utilizează această reprezentare (Cormen et al., 2009).

O formulă în CNF este alcătuită din:

- **literal**: o variabilă (x) sau negația acesteia ($\neg x$);
- **clauză**: o disjuncție de literali (ex.: $x \vee \neg y \vee z$);
- **conjuncție**: o conjuncție de clauze (ex.: $(x \vee y) \wedge (\neg x \vee z) \wedge (y \vee \neg z)$).

De exemplu, formula $a \wedge \neg b$ este satisfiabilă (pentru $a = adevarat,\ b = fals$), în timp ce formula $a \wedge \neg a$ este nesatisfiabilă.

Un rezultat fundamental în teoria complexității este faptul că SAT a fost prima problemă demonstrată ca fiind NP-completă, prin teorema Cook–Levin (Cormen, 2009; Sipser, 2013). Aceasta înseamnă că orice problemă din clasa NP poate fi redusă în timp polinomial la o instanță a problemei SAT.

În consecință, SAT este considerată una dintre cele mai dificile probleme din NP și joacă un rol central în studiul complexității algoritmilor. În prezent, nu se cunoaște niciun algoritm determinist în timp polinomial care să rezolve toate instanțele problemei SAT, iar existența unui astfel de algoritm este echivalentă cu rezolvarea celebrei probleme P versus NP.

Clasele P, NP și NP-completitudine. Pentru a înțelege rolul problemei satisfiabilității (SAT), este necesară analiza relațiilor dintre principalele clase de complexitate. Aceste clase permit clasificarea problemelor în funcție de dificultatea lor computațională. Clasa P conține problemele care pot fi rezolvate în timp polinomial de un algoritm determinist. Acestea sunt considerate probleme „eficiente", deoarece timpul de execuție crește rezonabil odată cu dimensiunea datelor de intrare (Cormen, 2009). Clasa NP conține problemele pentru care o soluție, odată oferită, poate fi verificată în timp polinomial de un algoritm determinist. Echivalent, aceste probleme pot fi rezolvate în timp polinomial de o mașină Turing nedeterministă (Sipser, 2013). Relația dintre aceste clase este dată de incluziunea $\mathbf{P} \subseteq \mathbf{NP}$. Aceasta se explică prin faptul că, dacă o problemă poate fi rezolvată rapid, atunci soluția obținută poate fi și verificată rapid. O problemă este $\mathbf{NP-hard}$ dacă orice problemă din NP poate fi redusă la aceasta în timp polinomial. Acest lucru înseamnă că problema este cel puțin la fel de dificilă ca toate problemele din NP, însă nu este obligatoriu să aparțină clasei NP. O problemă este $\mathbf{NP-completă}$ dacă îndeplinește simultan două condiții: aparține clasei NP este NP-hard.

Problemele NP-complete sunt considerate cele mai dificile probleme din NP. Dacă s-ar găsi un algoritm în timp polinomial pentru o singură problemă NP-completă, atunci toate problemele din NP ar putea fi rezolvate eficient, ceea ce ar implica egalitatea $\mathbf{P} = \mathbf{NP}$ (Sipser, 2013). Clasa NP-hard este mai largă decât NP și poate include și probleme care nu sunt verificabile în timp polinomial. Intersecția dintre NP și NP-hard este exact mulțimea problemelor NP-complete.

Relațiile dintre aceste clase pot fi rezumate astfel:

- $P \subseteq NP$
- $NP - complet \subseteq NP$
- $NP - complet \subseteq NP - hard$
- $NP \cap NP - hard = NP - complet$

Aceste relații sunt ilustrate în Figura 2, care prezintă o diagramă de tip Euler pentru clasele P, NP, NP-complete și NP-hard. Figura evidențiază atât cazul general considerat în mod uzual ($P \neq NP$), în care clasa P este strict inclusă în NP, cât și scenariul ipotetic în care $P = NP$. În acest din urmă caz, clasele P, NP și NP-complete coincid, ceea ce ar însemna că toate problemele pentru care o soluție poate fi verificată eficient pot fi și rezolvate eficient.

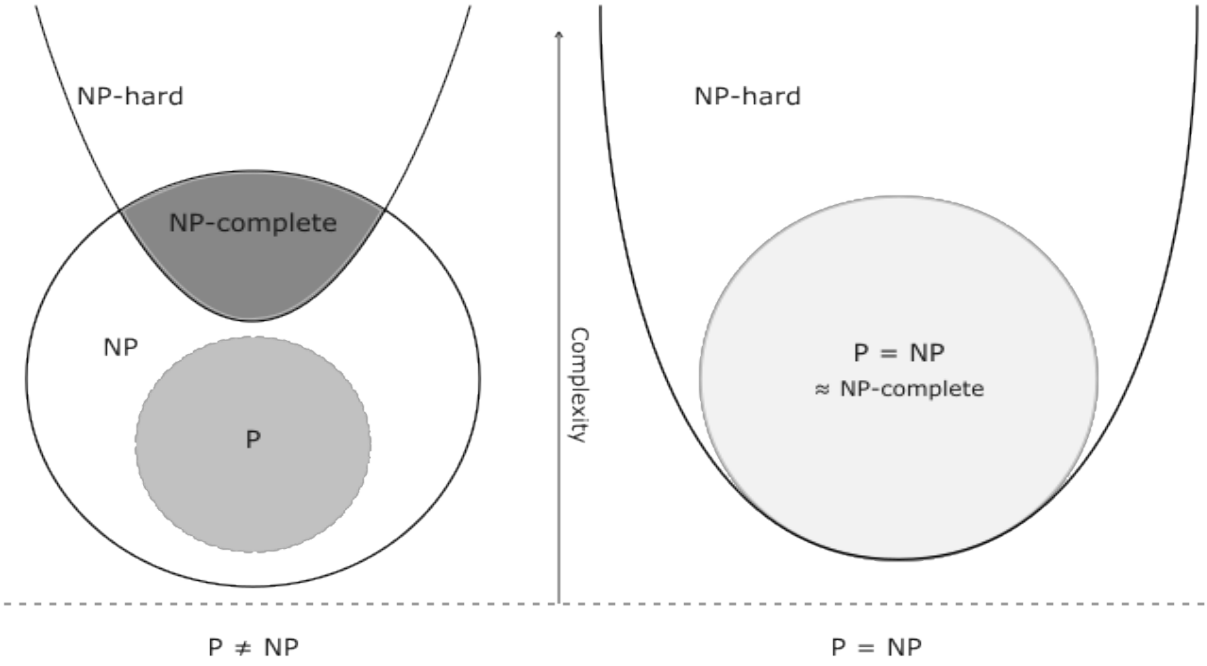


Figura 2. Diagrama Euler pentru clasesle P, NP, NP-complete și NP-hard în cazul P≠NP și P=NP
Sursa: Realizat de autori după (Sipser, 2013)

Rezolvatoarea SAT. Pentru rezolvarea practică a problemei satisfiabilității booleene au fost dezvoltați algoritmi specializați, cunoscuți sub denumirea de SAT solvers. Aceștia au rolul de a determina dacă o formulă booleană este satisfiabilă, adică dacă există o atribuire a variabilelor care o face adevărată (Biere et al., 2021). Din perspectivă teoretică, problema SAT este reprezentativă pentru clasa NP și este strâns legată de modelul mașinilor Turing nedeterministe – o soluție poate fi „ghicită" și verificată în timp polinomial. În practică însă, astfel de mecanisme nu există, motiv pentru care rezolvarea trebuie realizată prin explorarea sistematică a spațiului de căutare. SAT solvers moderne încearcă să simuleze această idee de „ghicire" prin tehnici eficiente de căutare. Algoritmi fundamentali precum DPLL (Davis–Putnam–Logemann–Loveland) explorează recursiv posibile atribuiri ale variabilelor, utilizând strategii de ramificare și mecanisme de backtracking pentru a reveni asupra deciziilor atunci când se ajunge la contradicții (Davis et al., 1962). Pe baza acestuia, au fost dezvoltate metode moderne precum CDCL (Conflict-Driven Clause Learning), care extind DPLL prin învățarea de clauze și analiza conflictelor, reducând semnificativ spațiul de căutare și îmbunătățind performanța în practică (Biere et al., 2021).În general, majoritatea solverelor moderne sunt optimizate pentru formule în forma normală conjunctivă (CNF), deoarece aceasta permite aplicarea eficientă a tehnicilor de propagare și simplificare.

DPLL (Davis–Putnam–Logemann–Loveland). Esența algoritmului DPLL constă în explorarea sistematică a tuturor posibilelor atribuiri ale variabilelor printr-un proces de backtracking.(Davis et al., 1962) La fiecare pas, se selectează o variabilă și i se atribuie o valoare de adevăr, după care formula este simplificată în funcție de această alegere. Dacă în procesul de evaluare apare o contradicție, algoritmul revine (backtracking) și încearcă valoarea opusă pentru aceeași variabilă. Acest mecanism este aplicat recursiv și este cunoscut sub numele de **regula de împărțire (splitting rule)**. Pentru a reduce spațiul de căutare, DPLL utilizează două optimizări importante (Biere et al., 2021):

- **Unit propagation**; Dacă o clauză conține un singur literal, atunci acel literal trebuie să fie adevărat pentru ca formula să fie satisfiabilă, iar valoarea sa este impusă automat.

- **Pure literal elimination;** Dacă o variabilă apare în formulă doar într-o singură formă (doar pozitivă sau doar negativă), aceasta poate fi atribuită astfel încât să satisfacă toate clauzele în care apare, permițând eliminarea lor din formulă.

CDCL (Conflict-Driven Clause Learning). CDCL poate fi privit ca o extensie modernă a algoritmului DPLL. În timp ce DPLL explorează spațiul de căutare prin backtracking simplu, CDCL adaugă mecanisme care permit învățarea din greșeli și evitarea repetării aceloraşi situații. La fel ca în DPLL, solverul construiește treptat o atribuire parțială a variabilelor și încearcă să o extindă până când obține o soluție completă sau până când descoperă că formula nu poate fi satisfiabilă. Diferența esențială apare atunci când se ajunge la o contradicție (conflict). În loc să revină pur și simplu la pasul anterior, așa cum face DPLL, CDCL analizează cauza conflictului și extrage informații utile din acesta. (Marques-Silva & Sakallah, 1999)

Analiza unei problem reale (Graph Coloring). Pentru a ilustra aplicarea practică a conceptelor teoretice prezentate anterior, a fost aleasă problema colorării grafurilor (Graph Coloring), o problemă clasică din teoria grafurilor și informatică, cunoscută pentru complexitatea sa și pentru faptul că poate fi formulată ca o problemă de satisfiabilitate booleană (SAT). În cadrul acestei lucrări, problema a fost abordată prin două metode distincte:

- utilizarea unui algoritm direct de tip backtracking, adaptat specific pentru problema colorării grafurilor;
- reducerea problemei la o formulă în forma normală conjunctivă (CNF) și rezolvarea acesteia folosind un solver SAT bazat pe algoritmul DPLL.

Scopul acestei abordări este de a compara performanța și eficiența celor două metode, evidențiind avantajele și dezavantajele fiecăreia în funcție de dimensiunea și structura instanțelor analizate. În continuare, va fi prezentată transformarea problemei de colorare a grafurilor în SAT, urmată de o descriere succintă a algoritmului direct utilizat și, de analiza rezultatelor experimentale obținute.

Problema colorării grafurilor. Problema colorării grafurilor (Graph Coloring) constă în atribuirea unor culori nodurilor unui graf astfel încât două noduri adiacente să nu aibă aceeași culoare (Jensen & Toft, 1995). Formal, fie un graf $G = (V, E)$, unde V este mulțimea nodurilor și E mulțimea muchiilor. Pentru un număr dat de culori k, se cere determinarea unei funcții:

$$f: V \to \{1,2,\dots,k\}$$

astfel încât pentru orice muchie $(u, v) \in E$, să fie îndeplinită condiția:

$$f(u) \neq f(v)$$

Problema prezintă o relevanță practică ridicată, având aplicații în alocarea resurselor, planificare, compilatoare și rețele. Din punct de vedere teoretic, forma sa decizională („există o colorare validă cu k culori?") aparține clasei problemelor NP-complete (Karp, 2010).

Transformarea problemei Graph Coloring în SAT. Pentru a putea utiliza un solver SAT, problema de colorare a grafurilor este transformată într-o formulă booleană în forma normală conjunctivă (CNF), astfel încât satisfiabilitatea acesteia să fie echivalentă cu existența unei colorări valide. Pentru fiecare nod $v \in V$ și fiecare culoare $c \in \{1,2,\dots,k\}$, se introduce o variabilă booleană: $x_{v,c}$ care indică dacă nodul v primește culoarea c.

Pentru a modela corect problema, sunt introduse următoarele tipuri de constrângeri:

- Fiecare nod are cel puțin o culoare: pentru fiecare nod v: $(x_{v,1} \vee x_{v,2} \vee \cdots \vee x_{v,k})$
- Fiecare nod are cel mult o culoare: pentru fiecare nod v și orice două culori diferite $c_1 \neq c_2$:

  $(\neg x_{v,c1} \vee \neg x_{v,c2})$
- Nodurile adiacente au culori diferite: pentru fiecare muchie $(u, v) \in E$ și fiecare culoare c:

  $(\neg x_{u,c} \vee \neg x_{v,c})$

  Toate constrângerile sunt combinate într-o formulă CNF:
  $\Phi = \wedge(toate\ clauzele\ definite\ mai\ sus)$

Formula CNF construită este generată programatic pentru graful dat și este analizată cu ajutorul unui rezolvator SAT bazat pe algoritmul DPLL

Algoritm direct pentru Graph Coloring. Pe lângă abordarea bazată pe reducerea la SAT, problema colorării grafurilor a fost rezolvată și printr-un algoritm direct, conceput special pentru această problemă. Algoritmul utilizează o strategie de tip backtracking, îmbunătățită prin tehnici de selecție a variabilelor și verificare anticipată a constrângerilor. Algoritmul construiește incremental o soluție, atribuind culori nodurilor grafului. La fiecare pas, se selectează un nod necolorat și se încearcă atribuirea unei culori valide. Dacă la un moment dat nu mai există nicio atribuire posibilă, algoritmul revine asupra deciziilor anterioare (backtracking) și încearcă alte variante. Pentru fiecare atribuire, se verifică dacă aceasta respectă constrângerea de bază a problemei: două noduri adiacente nu pot avea aceeași culoare. Această verificare se realizează prin parcurgerea vecinilor nodului curent și compararea culorilor deja atribuite. În locul unei selecții arbitrare a nodurilor, algoritmul utilizează o euristică de tip „minimum remaining values" (MRV). Astfel, se alege nodul necolorat care are cel mai mic număr de culori valide disponibile. Această strategie reduce semnificativ spațiul de căutare, deoarece nodurile cele mai restrictive sunt tratate mai devreme, crescând probabilitatea de detectare rapidă a conflictelor.

Pentru a evita explorarea unor ramuri imposibile, algoritmul include o tehnică de tip forward checking. După fiecare atribuire, se verifică dacă există noduri necolorate pentru care nu mai este disponibilă nicio culoare validă. Dacă o astfel de situație este detectată, ramura curentă este abandonată imediat.

Algoritmul poate fi descris succint astfel:

- Selectează un nod necolorat (folosind MRV)
- Pentru fiecare culoare posibilă:
  - Verifică dacă atribuirea este validă
  - Atribuie culoarea
  - Aplică forward checking
  - Apelează recursiv algoritmul
  - Dacă nu există soluție, revine (backtracking)
- Dacă toate nodurile sunt colorate programul returnează soluție atribuirea de culori găsită.

Rezultate experimentale și analiză. Pentru a evalua performanța celor două abordări prezentate – reducerea problemei la SAT și algoritmul direct de colorare – au fost realizate experimente pe o serie de grafuri generate cu parametri variați. În cadrul testelor, s-au modificat dimensiunea grafului (n), densitatea (probabilitatea în procente a existenței unei muchii între 2

noduri aleatorii) acestuia (p) și numărul de culori disponibile (k). Rezultatele obținute sunt sintetizate în Tabelul 1, unde sunt prezentați timpii de execuție (în secunde) pentru fiecare instanță, atât pentru metoda bazată pe SAT (utilizând algoritmul DPLL), cât și pentru algoritmul direct. Analizând datele experimentale, se pot observa mai multe tendințe relevante.

Tabelul 1. Timpi de execuție (în secunde) pentru rezolvarea problemei Graph Coloring folosind DPLL și algoritmul direct

Caz	DPLL	Direct	Caz	DPLL	Direct
n10_p10_k2	0.00007	0.00006	n30_p10_k2	0.00055	0.00041
n10_p10_k3	0.00011	0.00006	n30_p10_k3	0.0012	0.00078
n10_p10_k4	0.0002	0.00008	n30_p10_k4	0.00301	0.0008
n10_p10_k5	0.00038	0.00008	n30_p10_k5	0.00496	0.00091
n10_p30_k2	0.00008	0.00007	n30_p30_k2	0.0005	0.00019
n10_p30_k3	0.00036	0.00017	n30_p30_k3	0.01421	0.00185
n10_p30_k4	0.00064	0.00014	n30_p30_k4	0.54536	0.11611
n10_p30_k5	0.00048	0.00012	n30_p30_k5	0.02581	0.00126
n10_p50_k2	0.00014	0.0001	n30_p50_k2	0.00063	0.00022
n10_p50_k3	0.00063	0.00022	n30_p50_k3	0.01057	0.00125
n10_p50_k4	0.00124	0.00027	n30_p50_k4	0.09133	0.00482
n10_p50_k5	0.00085	0.00016	n30_p50_k5	2.06872	0.05784
n20_p10_k2	0.00038	0.00037	n35_p10_k2	0.00051	0.0003
n20_p10_k3	0.00051	0.00026	n35_p10_k3	0.0026	0.00104
n20_p10_k4	0.00112	0.00051	n35_p10_k4	0.00398	0.00165
n20_p10_k5	0.00179	0.00064	n35_p10_k5	0.00635	0.00158
n20_p30_k2	0.00047	0.00015	n35_p30_k2	0.00084	0.00027
n20_p30_k3	0.0045	0.00098	n35_p30_k3	0.01499	0.00171
n20_p30_k4	0.05491	0.0077	n35_p30_k4	0.35569	0.02099
n20_p30_k5	0.0136	0.00072	n35_p30_k5	0.55993	0.01249
n20_p50_k2	0.00051	0.00019	n35_p50_k2	0.00116	0.00029
n20_p50_k3	0.00461	0.00066	n35_p50_k3	0.01344	0.00136
n20_p50_k4	0.04017	0.00404	n35_p50_k4	0.19438	0.00656
n20_p50_k5	2.33759	0.22631	n35_p50_k5	4.56129	0.06892

Sursa: realizat de autor pe baza implementării proprii

Pentru instanțele de dimensiuni mici (de exemplu, $n = 10$), diferențele dintre cele două metode sunt reduse, timpii de execuție fiind comparabili. Acest lucru sugerează că, pentru probleme simple, costul suplimentar al transformării în SAT și al rezolvării prin DPLL nu este semnificativ. În schimb, pe măsură ce dimensiunea grafului și densitatea acestuia cresc, apar diferențe semnificative de performanță. Metoda bazată pe SAT prezintă o creștere accentuată a timpului de execuție, în special pentru valori mai mari ale parametrilor n, p și k. De exemplu, pentru cazul $n = 35$, $p = 50$, $k = 5$, timpul de execuție ajunge la peste 4 secunde, în timp ce algoritmul direct rezolvă aceeași instanță în mai puțin de 0,1 secunde. Această diferență poate fi explicată prin natura celor două abordări. Reducerea la SAT implică transformarea problemei într-o formulă CNF de dimensiuni considerabile, ceea ce conduce la un număr mare de variabile și clauze. În consecință,

algoritmul DPLL este nevoit să exploreze un spațiu de căutare mult mai extins. În contrast, algoritmul direct utilizează structura problemei de colorare a grafurilor, aplicând constrângeri locale și tehnici precum selecția variabilelor (MRV) și forward checking. Aceste optimizări permit eliminarea rapidă a soluțiilor invalide și reduc semnificativ numărul de combinații explorate.

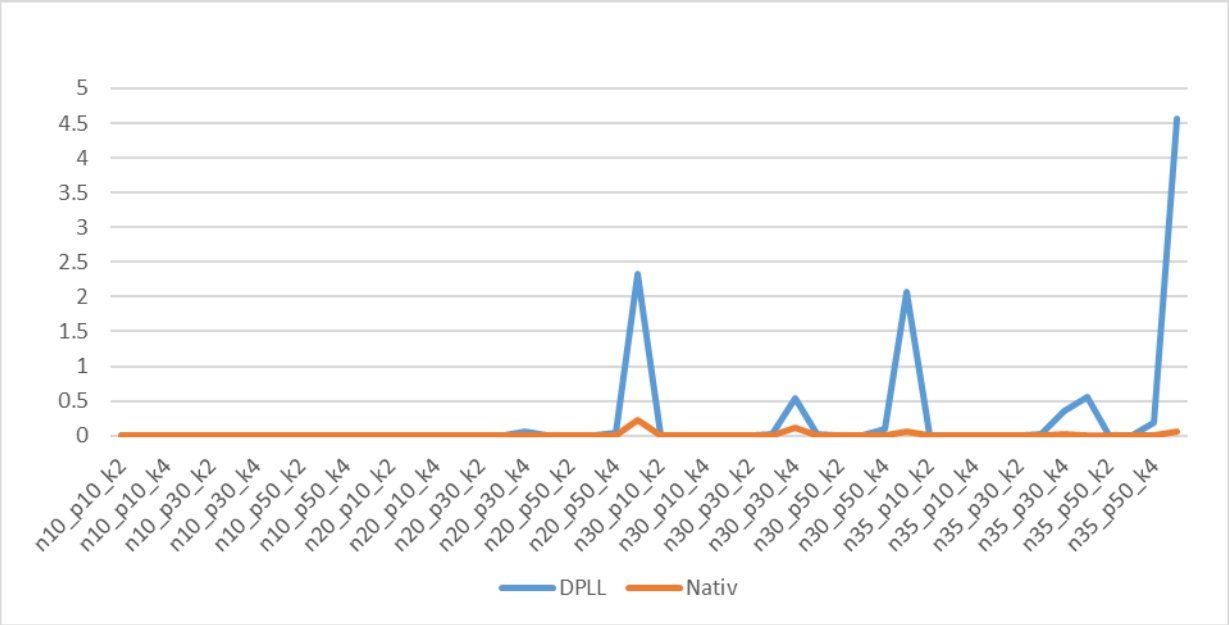


Figura 3. Compararea timpilor de execuție (în secunde) pentru rezolvarea problemei Graph Coloring utilizând metoda SAT (DPLL) și algoritmul direct
Sursa: realizat de autor pe baza rezultatelor experimentale din Tabelul 1

Un alt aspect important observat este influența densității grafului. Pentru grafuri mai dense, numărul de constrângeri crește, ceea ce afectează mai puternic performanța metodei SAT (Figura 3), în timp ce algoritmul direct reușește să gestioneze mai eficient aceste restricții prin verificări directe între nodurile adiacente.

CONCLUZII

În cadrul acestei lucrări au fost analizate conceptele fundamentale ale teoriei complexității computaționale, cu accent pe problema satisfiabilității booleene (SAT) și relația acesteia cu clasele de complexitate P și NP. Au fost analizate principalele abordări de rezolvare a SAT-urilor, cu accent pe algoritmii DPLL și CDCL, care constituie baza solverelor moderne de satisfiabilitate (Biere et al., 2021). Rezultatele experimentale obținute indică faptul că abordarea bazată pe SAT, deși oferă un cadru general și solid din punct de vedere teoretic, prezintă o performanță inferioară în instanțele analizate, acest lucru evidențiind în mod practic dificultatea problemelor din clasa NP. Cu toate acestea, algoritmul DPLL își păstrează importanța teoretică fundamentală, reprezentând baza conceptuală a rezolvatoarelor moderne de satisfiabilitate, a căror performanță este considerabil îmbunătățită prin extensii precum Conflict-Driven Clause Learning (CDCL) (Biere et al., 2021) și prin utilizarea euristicilor avansate de explorare a spațiului de căutare.

Dintr-o perspectivă teoretică generală, rezultatele evidențiază complexitatea intrinsecă a problemelor din clasa NP, în special a celor NP-complete, iar problema satisfiabilității booleene (SAT), ca reprezentant canonic al acestei clase, ocupă un rol central în studiul limitelor computației eficiente.

REFERINȚE:

1. Biere, A., Heule, M., & Maaren, H. van (Ed.). (2021). *Handbook of satisfiability* (Second edition). IOS Press.
2. Cormen, T. H. (Ed.). (2009). *Introduction to algorithms* (3. ed). MIT Press.

3. Davis, M., Logemann, G., & Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, *5*(7), 394-397. https://doi.org/10.1145/368273.368557
4. Erickson, J. (2019). *Algorithms* (1st paperback edition). Selbstverlag.
5. Jensen, T. R., & Toft, B. (1995). *Graph coloring problems*. J. Wiley & sons.
6. Karp, R. M. (2010). Reducibility Among Combinatorial Problems. În M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, & L. A. Wolsey (Ed.), *50 Years of Integer Programming 1958-2008* (pp. 219-241). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-68279-0_8
7. Knuth, D. E. (1982). *The art of computer programming. 1: Fundamental algorithms* (2. ed., 7. print). Addison-Wesley.
8. Marques-Silva, J. P., & Sakallah, K. A. (1999). GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, *48*(5), 506-521. https://doi.org/10.1109/12.769433
9. P vs NP. (f.a.). *Clay Mathematics Institute*. Preluat în 13 aprilie 2026, din https://www.claymath.org/millennium/p-vs-np/
10. Sipser, M. (2013). *Introduction to the theory of computation* (Third edition, international edition). Cengage Learning.
11. The Millennium Prize Problems. (f.a.). *Clay Mathematics Institute*. Preluat în 13 aprilie 2026, din https://www.claymath.org/millennium-problems/

Toacă Zinovia COORDONATOR ŞTIINŢIFIC
Academia de Studii Economice din Moldova, Republica Moldova,
Chișinău, str. Bănulescu-Bodoni 61, www.ase.md
Telefon: + 373 79144860
e-mail: toaca@ase.md