Evaluarea MiniSat: Performanță și Provocări în Rezolvarea Problemelor SAT

Grosu Damian, Scoropad Iulian, and Daniel Dobrescu

[1] Universitatea de Vest Timisoara, Bulevardul Vasile Pârvan 4, Timișoara 300223
damian.grosu01@e-uvt.ro,
[2] iulian.scoropad00@e-uvt.ro,
[3] costi.dobrescu01@e-uvt.ro
http://www.uvt.ro

Abstract. Raportul dat se concentrează pe analiza problemei satisfiabilității booleene (SAT), prima problemă recunoscută ca NP-completă, și pe aplicabilitatea acesteia în diverse domenii. Documentul prezintă în detaliu funcționarea și îmbunătățirile aduse MiniSat, un solver SAT minimalist și eficient,dezvoltat de Niklas Eén și Niklas Sörensson. Raportul include o descriere a structurilor și algoritmilor MiniSat, ghiduri pentru instalare, precum și metodologia utilizată pentru testarea performantelor acestuia. Deasemenea MiniSat va fi testat pe un set de benchmark-uri din competiția SAT 2024, analizând performanța MiniSat în termeni de satisfiabilitate, relevând constrângerile de timp, hardware și discutând eficiența acestuia. Concluziile care le vom sublinia în urma realizării proiectului vor arăta cât de util este MiniSat în rezolvarea problemei satisfiabilității.

1 Introducere

Rezolvarea problemei de satisfiabilitate booleeană (SAT) reprezintă un punct central al cercetării în domeniul informaticii, fiind primul exemplu de problemă NP-completă. Termenul NP vine de la *nondeterministic polynomial time* (timp polinomial nedeterminist) iar în teoria complexității calculului, NP este o clasă de probleme decizionale care are următoarele caracteristici:

Verificare eficientă: Dacă cineva îți oferă o soluție (un „certificat") pentru o problemă din NP, poți verifica dacă soluția este corectă în timp polinomial folosind un algoritm determinist. Cu alte cuvinte, soluția poate fi verificată rapid (eficient) de un calculator.

SAT (Satisfiability Problem) este o problemă din logica propozițională, care implică determinarea existenței unei asignări de variabile booleene astfel încât o formulă dată să fie adevărată. Această problemă a fost definită formal ca fiind NP-completă de către Cook în 1971. De atunci, SAT a devenit un subiect de mare interes datorită implicațiilor teoretice și practice.

1.1 Domenii de aplicare a SAT Solverilor

SAT Solvers sunt folosiți pe scară largă într-o varietate de domenii, cum ar fi:

Verificarea formală a hardware-ului și software-ului (Model Checking)
SAT Solvers sunt utilizați pentru a verifica dacă un sistem hardware sau software respectă specificațiile dorite, identificând eventualele erori.
Exemplu: În verificarea microprocesoarelor, un SAT Solver poate fi utilizat pentru a verifica dacă circuitele logice implementate funcționează conform designului specificat, detectând potențiale situații de blocaj sau comportamente incorecte.
Aplicație concretă: Intel folosește SAT Solvers pentru a valida designul procesoarelor înainte de producția de masă.
În software, SAT Solvers ajută la analiza și detectarea bug-urilor în programe critice, cum ar fi cele pentru aviație sau medicină

Planificare automată și inteligență artificială
SAT Solvers sunt folosiți pentru a rezolva probleme complexe de planificare, în care trebuie găsită o secvență optimă de acțiuni pentru a atinge un obiectiv.
Exemplu: Un robot care planifică rutele de curățare într-o clădire trebuie să determine traseele eficiente care să acopere toate camerele, evitând obstacolele. SAT Solvers pot rezolva astfel de probleme modelând restricțiile și obiectivele ca o formulă logică.
Aplicație concretă: În jocurile video, SAT Solvers pot fi utilizați pentru a genera strategii optime pentru personaje controlate de inteligența artificială.

Optimizarea și rezolvarea problemelor de satisfacție a constrângerilor (CSP)
SAT Solvers sunt esențiali în problemele de optimizare care implică multe restricții. Acestea pot varia de la alocarea resurselor până la programarea unor activități.
Exemplu: Planificarea orarului școlar într-o școală mare, astfel încât să se evite

suprapunerea profesorilor, sălilor sau grupurilor de studenți. SAT Solver-ul poate rezolva problema transformând constrângerile într-o formulă logică satisfiabilă.
Aplicație concretă: Companii aeriene folosesc SAT Solvers pentru optimizarea rutelor de zbor și a programelor echipajelor, reducând costurile operaționale.

Criptografie și securitate cibernetică
Exemplu: Testarea algoritmilor de criptare pentru a descoperi chei slabe. SAT Solvers pot încerca să determine dacă o anumită cheie secretă poate fi derivată mai ușor decât ar trebui în mod normal.
Aplicație concretă: În analiza malware, SAT Solvers pot fi folosiți pentru a identifica dacă un set de reguli de securitate este satisfăcut de un comportament suspect observat într-un sistem.

2 Descrierea problemei

Problema principală abordată de un SAT Solver este determinarea satisfiabilității unei formule logice. Satisfiabilitatea presupune existența unei configurații de valori de adevăr (True/False) pentru variabilele formulei care să conducă la un rezultat adevărat. Dacă o astfel de configurație există, formula este considerată satisfiabilă; în caz contrar, este nesatisfiabilă.

Rezolvarea problemelor de satisfiabilitate este importantă deoarece multe alte probleme complexe pot fi reduse la SAT. În cazul nostru, ne propunem să verificăm dacă o formulă logică dată este satisfiabilă sau nu. Formula logică este exprimată în formă normală conjunctivă, compusă din mai multe clauze care sunt, la rândul lor, disjuncții de literali.

3 Minisat

3.1 Introducere în Minisat

MiniSat[3] este un solver minimalist și open-source destinat rezolvării problemelor de satisfiabilitate logică *(SAT - Satisfiability)*. Dezvoltat inițial de *Niklas Eén* și *Niklas Sörensson*, MiniSat a devenit rapid un punct de referință în domeniul solverelor SAT datorită simplității, performanței și modularității sale. A obținut realizări semnificative în competițiile internaționale de SAT, cum ar fi SAT Competition, unde a avut multiple performanțe, inclusiv câștigarea competiției din 2005. Aceste succese au demonstrat că MiniSat este un solver eficient, capabil să abordeze o gamă largă de instanțe de satisfiabilitate. Determinarea satisfiabilității unei formule este o problemă de bază în calculul și logica matematică, constând în determinarea dacă o formulă booleană poate fi evaluată ca adevărată (satisfiabilă) printr-o atribuire adecvată a valorilor de adevăr variabilelor sale. Popularitatea MiniSat derivă din implementarea eficientă a tehnicilor moderne SAT și din capacitatea sa de a rezolva rapid probleme complexe.

MiniSat este aplicabil în numeroase domenii datorită capacității sale de a analiza expresii logice complexe și de a determina satisfiabilitatea acestora. Printre cele mai semnificative aplicații ale MiniSat se numără:

În ingineria software și hardware, MiniSat este utilizat pentru verificarea formală a modelelor logice și a circuitelor digitale. Prin transformarea problemelor în formă SAT, proiectanții pot verifica proprietăți precum coerența, corectitudinea și consistența modelelor.

În optimizarea planificării și alocării resurselor în proiecte complexe, cum ar fi gestionarea orarelor, planificarea în producție sau alocarea resurselor în centre de date.

În domeniul criptografiei, MiniSat este folosit pentru analizarea și verificarea algoritmilor criptografici. Poate detecta vulnerabilități în sisteme prin formularea problemelor de atac sau rezistență sub formă de instanțe SAT.

În bioinformatică, problemele de asamblare a genomului sau predicția structurilor moleculare pot fi exprimate în format SAT, iar MiniSat oferă o platformă eficientă pentru rezolvarea acestor probleme.

În inteligență artificială, MiniSat este folosit pentru raționamente logice și rezolvarea automată a problemelor de satisfiabilitate, având aplicații în cercetarea sistemelor de planificare automată, generarea de soluții optime în probleme de optimizare și analiza complexității problemelor combinatorii.

3.2 Algoritmi utilizați în MiniSAT

MiniSat își bazează performanța pe doi algoritmi principali: CDCL *(Conflict-Driven Clause Learning)* și DPLL *(Davis-Putnam-Logemann-Loveland)*.

CDCL extinde DPLL prin învățarea din conflicte, reducând spațiul de căutare prin generarea de clauze noi care elimină cauzele conflictelor. În timpul căutării, propagarea unitară este optimizată prin tehnici precum *Blocking Literals*, iar variabilele sunt alese folosind euristici precum *VSIDS*, care prioritizează cele implicate în conflicte recente.

DPLL rămâne fundamentul căutării, utilizând propagarea unitară și backtracking pentru explorarea sistematică a soluțiilor, însă CDCL adaugă mecanisme precum sărituri peste niveluri multiple în cazul conflictelor și restarturi adaptive pentru a preveni stagnarea. În plus, MiniSat permite rezolvarea incrementală a problemelor, economisind resurse în aplicațiile dinamice. Astfel, integrarea acestor tehnici face procesul mai eficient și mai robust.

3.3 Analiză Detaliată a Structurii și Modului de Lucru al MiniSat

MiniSat își structurează funcționalitatea în două module majore: **Solver.cc**, care reprezintă motorul principal, și **SimpSolver.cc**, care extinde funcționalitățile de bază prin metode de optimizare și simplificare. Fiecare dintre aceste fișiere joacă un rol distinct și complementar în cadrul arhitecturii solver-ului.

Rolul și Funcționalitatea Modulului Solver.cc

Fișierul *Solver.cc* constituie nucleul solver-ului MiniSat. Acesta include toate componentele esențiale pentru rezolvarea problemelor SAT, de la gestionarea variabilelor și clauzelor, până la analiza conflictelor și aplicarea strategiilor de backtracking. Printre cele mai importante funcționalități se numără:

- **Inițializarea și gestionarea variabilelor** prin metode precum `newVar()`, care configurează variabilele SAT și determină dacă acestea pot fi folosite pentru decizii.
- **Analiza conflictelor** utilizând metoda `analyze()`, care produce clauze învățate pentru a reduce spațiul de căutare.
- **Propagarea literelor** prin funcția `propagate()`, ce verifică satisfiabilitatea clauzelor pe baza alocărilor curente.
- **Gestionarea memoriei** și a eficienței, cu funcții precum `garbageCollect()` și `reduceDB()`.

Acest modul se concentrează pe furnizarea unui algoritm solid și eficient pentru rezolvarea problemelor SAT, fără a include tehnici suplimentare de optimizare.

Rolul și Funcționalitatea Modulului SimpSolver.cc

Fișierul *SimpSolver.cc* extinde funcționalitățile din Solver.cc prin adăugarea de metode dedicate simplificării problemelor SAT. Acesta nu înlocuiește motorul principal, ci lucrează împreună cu el pentru a optimiza procesul de rezolvare. Funcționalitățile cheie includ:

- **Eliminarea variabilelor și subsumarea clauzelor** utilizând metode precum `eliminateVar()` și `backwardSubsumptionCheck()`, care reduc dimensiunea problemei înainte de rezolvare.
- **Întărirea clauzelor** prin funcția `strengthenClause()`, care elimină litere redundante pentru a optimiza baza de date a clauzelor.
- **Propagarea asimetrică** realizată prin `asymmVar()`, care identifică și elimină clauzele redundante.
- **Gestionarea modelelor extinse** prin `extendModel()`, care reconstruiește soluția completă din variabilele simplificate.

Acest modul este conceput pentru scenarii complexe, unde optimizările inițiale pot reduce semnificativ timpul de rezolvare.

Modul de lucru MiniSat

În continuare vom prezenta detaliat pasii care se executa în programul Minisat în timpul rulării unui fișier :

1. **Citirea fișierului CNF**
   - Executia MiniSat începe prin citirea unui fișier **CNF** (Conjunctive Normal Form), care conține formula SAT. Acesta este un fișier în formatul **DIMACS**, ce conține o problemă SAT în formă clasică adică un set de variabile și clauze.
   - MiniSat utilizează funcția `parseDimacs()` pentru a citi fișierul și extrage variabilele și clauzele. Formula citită este stocată într-o structură internă (un obiect `Solver`).
2. **Inițializarea Solver-ului**
   - După citirea fișierului CNF, MiniSat inițializează un obiect de tipul `Solver`.Acest obiect gestionează variabilele, clauzele și structurile de date asociate.
3. **Adăugarea clauzelor și variabilelor**

- MiniSat adaugă clauzele citite în structura internă `clauses`, iar variabilele sunt înregistrate într-o structură de date asociată, fiecare variabilă având un identificator unic.
- Variabilele sunt mapate la valori interne (pozitive sau negative) și fiecare clauză va fi folosită în propagare și analize.

4. **Inițierea căutării unui model satisfiabil**
   - Odată ce toate clauzele și variabilele sunt încărcate în solver, MiniSat începe procesul de căutare a unei soluții satisfiabile.
   - MiniSat folosește un algoritm bazat pe DPLL (Davis-Putnam-Logemann-Loveland) cu tehnici adăugate pentru a îmbunătăți performanța, inclusiv **SDCL** (Shifting Dynamic Clause Learning).
5. **Alegerea unei variabile de decizie**
   - La începutul procesului de căutare, MiniSat trebuie să aleagă o **variabilă de decizie** căreia o să-i fie atribuită o valoare. MiniSat folosește o strategie de alegere aleatorie sau bazată pe activitate (adică alege variabila care a avut cele mai multe conflicte sau care a fost activă cel mai mult).
   - Alegerea inițială este făcută de funcția `pickBranchLit()`.
6. **Propagarea unității**
   - După ce o variabilă este aleasă și îi este atribuită o valoare, MiniSat propagă efectele acestei atribuirii prin propagarea unității. Dacă o clauză are doar o literă nealocată și celelalte litere sunt deja satisfăcute sau nealocate, MiniSat va atribui automat valoarea corectă acestei litere.
   - MiniSat folosește funcția `propagate()` pentru a verifica efectele atribuirii și pentru a aplica schimbările în toate clauzele implicate.
7. **Verificarea conflictelor**
   - Dacă, în urma propagării unității, apare un **conflict** (adică o clauză devine imposibil de satisfăcut), MiniSat revine la nivelul anterior de decizie și încearcă o altă alegere *(backtracking)*.
   - În acest caz, MiniSat va analiza conflictul folosind funcția `analyze()`, care va crea o clauză de învățare (clauză de raționament), ce va ajuta la prevenirea apariției aceleași erori pe viitor.
   - MiniSat utilizează această clauză de învățare pentru a ghida procesul de căutare și a reduce spațiul de căutare.
8. **Restarturi și optimizare**
   - MiniSat poate aplica **restarturi** pentru a îmbunătăți performanța. După un anumit număr de conflicte, MiniSat poate face un restart al căutării, ștergând unele dintre clauzele învățate și încercând să revină la o stare mai „curată”.
   - Aceste restarturi sunt reglate de funcția `luby`, care determină frecvența acestora, și au rolul de a evita blocarea în regiuni mari de căutare ineficiente.
9. **Finalizarea procesului**
   - Dacă MiniSat reușește să găsească o **soluție satisfiabilă**, procesul se încheie cu succes și soluția finală este returnată. MiniSat va returna un model de valori pentru variabile care satisface toate clauzele.

– Dacă nu poate găsi o soluție (adică dacă ajunge într-un punct în care toate ramificațiile duc la conflicte), procesul se încheie cu un rezultat **nesatisfiabil**.
– Dacă procesul de căutare atinge o limită de conflicte fără a găsi o soluție sau este întrerupt, poate returna **nesoluționat**.

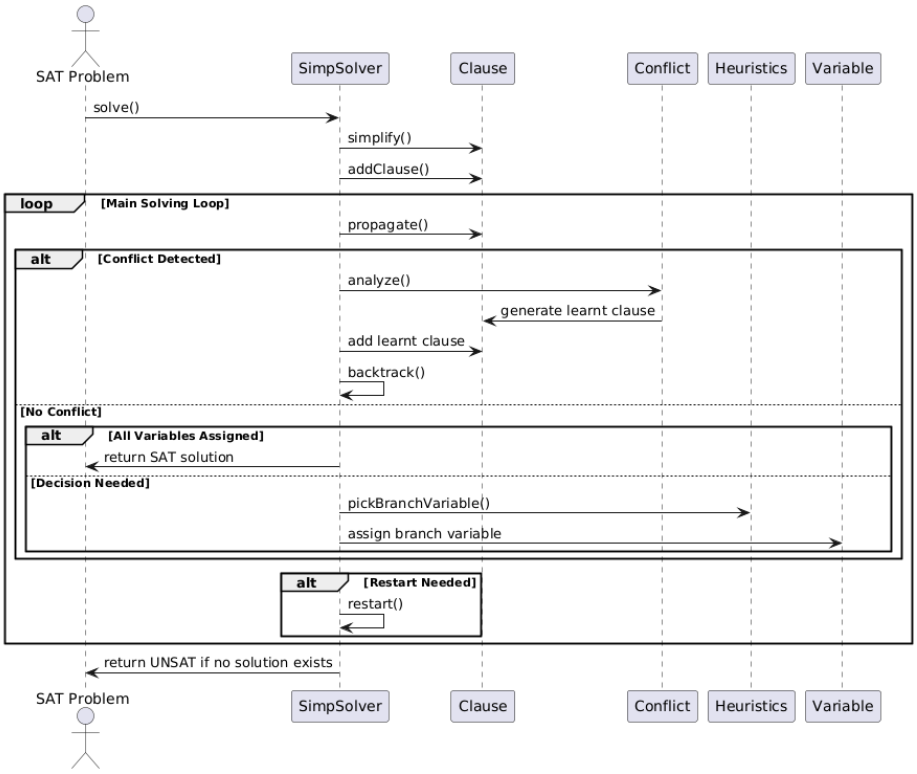

Fig. 1. Flow of the SAT Solver

3.4 Instalarea Minisat on PC

Linux Pe Linux, MiniSat poate fi instalat rapid folosind compilarea codului sursă. Pentru realizarea acestui lucru trebuie sa urmarim cațiva pași:

1. Instalează compilatorul și uneltele de build:

```
sudo apt update
sudo apt install g++ make
sudo apt install git build-essential
sudo apt install zlib1g-dev
```

2. Clonează repository-ul MiniSat:

```
git clone https://github.com/niklasso/minisat.git
```

3. În fișierul Makefile adaugăm următorul cod pentru setarea flag-urilor compilatorului C++ și pentru a spune compilatorui să fie mai permisiv:
   `CXXFLAGS += -std=c++11 -fpermissive`
4. Salvăm și închidem fisierul Makefile, recompilăm Minisat

```
make r
make cr
```

   Prima comandă este pentru versiunea standart a MiniSat, a doua comandă este pentru a impune MiniSat să ruleze `Main.cc` din fisierul *core*.
5. Verificăm instalarea

```
./minisat
./minisat_core
```

Mac Pe MAC, poate fi instalat usor utilizând Homebrew. Homebrew este un manager de pachete pentru macOS (și Linux) care simplifică instalarea softwareului și a uneltelor de dezvoltare direct din linia de comandă. Pentru configurarea MiniSAT va trebui sa urmarim câțiva pași:

1. Asigură-te că Homebrew este instalat. Dacă nu, instalează-l rulând comanda:

```
/bin/bash -c "$(curl -fsSL https://raw.
githubusercontent.com/Homebrew/install/HEAD/
install.sh)"
```

2. Instalează MiniSat folosind Homebrew:

```
brew install minisat
```

Windows MiniSat nu are o versiune nativă pentru Windows, dar poate fi rulat folosind soluții precum WSL (Windows Subsystem for Linux) sau MinGW. Pentru instalarea MiniSAT utilizand WSL vom parcurge următorii pași:

1. Activează WSL rulând comanda:

```
wsl --install
```

2. Instalează o distribuție Linux, precum Ubuntu, din Microsoft Store.
3. După instalare, deschide terminalul Linux și urmează pașii pentru Linux.

4 Algoritmii DPLL și CDCL

Algoritmii Davis-Putnam-Logemann-Loveland **(DPLL)** și Conflict-Driven Clause Learning **(CDCL)** sunt metode esențiale pentru determinarea satisfiabilității formulelor în formă normală conjunctivă **(CNF)**. Acestea stau la baza majorității solverelor SAT moderne, având aplicații diverse în verificarea modelelor, bioinformatică și criptografie. În continuare vom analiza structura, componentele și modul de funcționare al celor doi algoritmi, ilustrând implementarea acestora în cadrul codului MiniSat, în special în fișierul *Solver.cc* și *SimpSolver.cc* unde la rândul său sunt dezvoltați algoritmii.

4.1 Algoritmul DPLL

DPLL este un algoritm de căutare în adâncime care combină tehnici de deducere și decizie pentru a găsi o evaluare satisfăcătoare a variabilelor din formule CNF. În cadrul acestui proces, DPLL parcurge sistematic spațiul de căutare, utilizând propagarea literaliilor unitari, analizând conflictele, și făcând backtracking pentru a explora soluțiile posibile[6]. Principalele componente ale algoritmului **DPLL** sunt:

1. **Propagarea unității:** Atunci când o clauză conține un singur literal neasignat, valoarea acestuia este stabilită astfel încât să satisfacă clauza respectivă. Această operațiune poate declanșa alte asignări implicite în lanț, ceea ce poate duce fie la un conflict, fie la o soluție satisfăcătoare. Funcția `propagate()` din cod este responsabilă pentru acest proces. Aceasta parcurge literalii, aplică propagarea și returnează clauza conflictuală dacă se detectează un conflict.
2. **Heuristica de decizie:** Algoritmul selectează variabile pentru decizie, alegând în mod euristic sau aleatoriu variabila care urmează să fie asignată, pentru a accelera căutarea. Funcția `pickBranchLit()` din codul MiniSat implementează această etapă.
3. **Backtracking-ul:** Dacă algoritmul întâlnește un conflict, acesta revine la un nivel anterior de decizie pentru a explora alte opțiuni posibile. Funcția `cancelUntil()` realizează acest backtracking în cod, eliminând atribuțiile variabilelor până la un anumit nivel, astfel încât căutarea să fie reluată de la o stare anterioară.

DPLL reprezintă o metodă fundamentală care, deși simplă, constituie baza pentru majoritatea solverelor SAT și este punctul de plecare pentru metode mai complexe, precum CDCL.

4.2 Algoritmul CDCL

Conflict-Driven Clause Learning (CDCL) este o extensie avansată a DPLL, care include optimizări semnificative pentru a evita conflictele și a gestiona eficient spațiul de căutare. CDCL utilizează învățarea clauzelor conflictuale, backtracking-ul non-cronologic și tehnici de restart pentru a îmbunătăți substanțial eficiența rezolvării problemelor SAT[1].

Principalele componente ale algoritmului CDCL sunt:

1. **Învățarea clauzelor din conflicte:** Atunci când apare un conflict, CDCL analizează cauza și generează o clauză nouă care reflectă condițiile conflictuale. Aceasta este stocată ca „clauză învățată" și va împiedica re-apariția aceluiași conflict în viitor. Funcția analyze din cod implementează acest mecanism de învățare, derivând clauza conflictuală din condițiile care au dus la conflict.
2. **Propagarea unității și analiza conflictelor:** Similar DPLL, CDCL folosește propagarea literaliilor unitari pentru a asigna variabile și pentru a identifica conflictele. Însă, în cazul unui conflict, CDCL utilizează grafuri de implicații pentru a determina **punctul de implicație unică** (UIP), care reprezintă condiția specifică ce a generat conflictul. Aceasta permite un backtracking mai eficient, având ca rezultat reducerea numărului de conflicte repetitive, în cod funcția `litRedundant()` asigură optimizarea clauzelor învățate, eliminând literalii redundanți care nu contribuie direct la conflictul curent.
3. **Heuristici de decizie și ștergerea clauzelor:** CDCL utilizează euristici avansate, cum ar fi **VSIDS** (Variable State Independent Decaying Sum), care prioritează variabilele recente în conflicte. De asemenea, funcția `reduceDB()` elimină clauzele învățate mai puțin utile, pentru a optimiza memoria utilizată și pentru a menține eficiența solver-ului.
4. **Restarturi aleatorii:** În anumite instanțe de SAT dificile, CDCL poate efectua restarturi periodice pentru a evita blocajele în sub-arborii de căutare. Acest lucru este realizat prin funcția `luby()`, care definește intervalele de timp pentru restarturi. Restarturile permit solver-ului să exploreze mai eficient diverse regiuni ale spațiului de căutare.

Aceste funcții, combinate cu strategii avansate de decizie și propagare, permit CDCL să rezolve probleme SAT complexe cu eficiență sporită, în special acolo unde algoritmul DPLL deja nu mai este eficient.

5 Benchmark SAT Competition 2024

Competiția SAT 2024 reprezintă un eveniment anual de referință în domeniul rezolvării problemelor de satisfiabilitate booleană (SAT). Aceasta este organizată ca un eveniment satelit al Conferinței SAT 2024 și continuă tradiția competițiilor SAT și provocărilor SAT-Race.

În ultimii ani, domeniul SAT a înregistrat progrese remarcabile, transformând probleme considerate anterior imposibil de rezolvat în cazuri rezolvabile de rutină. Aceste avansuri sunt rezultatul:

- Algoritmilor noi și a euristicilor mai eficiente.
- Tehnicilor rafinate de implementare, care s-au dovedit esențiale pentru succes.

5.1 Rularea bencmark-ului

Competiția SAT 2024[4] include patru secțiuni principale, fiecare oferind oportunități distincte pentru evaluarea și dezvoltarea solverelor SAT:

- **Main Track:** Solvere SAT secvențiale.
- **Cloud Track:** Solvere rulate pe infrastructură cloud.
- **No Limit Track:** Fără restricții hardware sau temporale.
- **Parallel Track:** Solvere optimizate pentru paralelism.

Pentru analiza noastră, ne-am concentrat pe Main Track, deoarece aceasta reprezintă standardul principal pentru testarea performanței solverelor SAT. În cadrul acestei secțiuni in cadrul competiei au fost folosite:

- 300 și 600 de benchmark-uri
- Fiecare benchmark are o limită de timp de 5000 de secunde.
- Resursele hardware standard includ 128 GB RAM
- Solverii din toate pistele vor fi clasați pe baza scorului PAR-2, care este suma timpului de rulare plus de două ori timpul de expirare pentru instanțe nerezolvate.

In cadrul testarii noastre:

- 30 de benchmark-uri dintr-o familie numita miter
- Fiecare benchmark are o limită de timp de 2880 de secunde(ca sa putem rula 30 in timp de 24 de ore).
- Resursele hardware standard includ 16 GB RAM.

Pentru evaluarea performanței solverului SAT MiniSat, am dezvoltat un proces structurat pentru rularea unui set de 30 de fișiere de intrare în format .cnf. Procesul este descris detaliat mai jos.

Pentru o organizare eficientă, am creat două directoare principale. Fișiere de ieșire, acestea conțin soluțiile găsite de solver pentru fiecare fișier. Fișiere de statistici, acestea includ informații despre timpul de execuție și eventualele erori.

Pentru a asigura finalizarea testelor în intervalul de 24 de ore, fiecare execuție a fost limitată la 2880 de secunde. Această limitare a fost implementată folosind comanda gtimeout. Testele au fost efectuate conform următorilor pași:

1. Scriptul a iterat prin toate fișierele .cnf din directorul de intrare.
2. Pentru fiecare fișier solver-ul MiniSat a fost rulat utilizând gtimeout pentru a respecta limita de timp, iar rezultatele au fost salvate în fișiere dedicate în fișiere de soluție (conțin rezultatul obținut ex SAT sau UNSAT, sau INDET) și în fișiere de statistici(acestea documentează durata de execuție și erorile întâlnite).

5.2 Rezultate obtinute

Rezultatele obținute în urma testelor efectuate oferă o perspectivă clară asupra performanței solver-ului MiniSat în raport cu constrângerile hardware și temporale determinate de noi. În această secțiune, vom prezenta atât statisticile cheie, cât și observațiile relevante privind timpul de execuție, numărul de soluții găsite și comportamentul în cazul fișierelor complexe. De asemenea, vom analiza impactul resurselor hardware disponibile (16 GB RAM și procesor M1 Pro)

asupra capacității de procesare a instanțelor SAT, comparativ cu timpul limită de execuție impus (2880 secunde).

În urma executării a 30 de fișiere noi am creat un tabel pentru a urmări cum se comportă Minisat în raport cu hardware care îl avem noi și mărimea și complexitatea fișierului în sine. Astfel am observat ca am observat că majoritatea fișierelor de intrare nu au reușit să fie rezolvate în limita de timp stabilită, iar solver-ul MiniSat a fost adesea nevoit să se oprească înainte de a găsi o soluție completă. Aceasta subliniază complexitatea instanțelor testate și relația directă dintre timpul de execuție și resursele hardware disponibile. În cazul fișierelor cu complexitate mare (atât în ceea ce privește numărul de variabile, cât și al clauzelor), se poate observa toate lucrurile constate, ceea ce sugerează că aceste instanțe necesită mai multe resurse pentru a fi procesate în mod eficient. Pe baza

Table 1. Summary of MiniSat Benchmark Results

File Name	Variables	Clauses	Conflicts	Decisions	CPU Time (s)	Result
benchmarkfile1.out	62500	204018	29708704	48128071	2837.68	INDET
benchmarkfile2.out	377084	1227752	7250914	23210158	2791.62	INDET
benchmarkfile3.out	5068	15159	37386591	43015895	2860.41	INDET
benchmarkfile4.out	85224	276256	25346482	53160034	2856.89	INDET
benchmarkfile5.out	5253	15714	37665303	43499562	2853.09	INDET
benchmarkfile6.out	44195	128348	27515226	42730372	2843.85	INDET
benchmarkfile7.out	4595	13744	30014170	33877577	2046.53	UNSAT
benchmarkfile8.out	1454067	4368062	4794121	629288053	2753.86	INDET
benchmarkfile9.out	2861	8538	79919298	97135775	2854.15	INDET
benchmarkfile10.out	203492	602144	46205580	160598330	2837.88	INDET
benchmarkfile11.out	1454447	4369240	5181035	555213477	2803.24	INDET
benchmarkfile12.out	48505464	120975196	26216	239335536	1028.45	INDET
benchmarkfile13.out	35987452	35987452	54414	744156224	1505.21	INDET
benchmarkfile14.out	4538	13573	40933294	46311917	2841.62	INDET
benchmarkfile15.out	5462	16339	37199213	43346708	2844	INDET
benchmarkfile16.out	3426	10231	74515628	91602426	2817.63	INDET
benchmarkfile17.out	100238	332316	14592808	3719779	2813.94	INDET
benchmarkfile18.out	3111	9288	75407032	90345235	2795.2	INDET
benchmarkfile19.out	2780	8297	98827073	114357472	2837.41	INDET
benchmarkfile20.out	85230	276272	25323587	51335360	2816.84	INDET
benchmarkfile21.out	5189	15520	38324966	44261680	2864.38	INDET
benchmarkfile22.out	2357	7030	60321247	72009980	1786.29	UNSAT
benchmarkfile23.out	29277589	72984706	96350	353079557	1178.5	SAT
benchmarkfile24.out	4745	14194	40793160	46156597	2835.76	INDET
benchmarkfile25.out	162	774	50133511	58308211	290.156	UNSAT
benchmarkfile26.out	4842964	12045456	6163	7470746	33.7055	SAT
benchmarkfile27.out	572519	1715261	5543998	15503270	2835.99	INDET
benchmarkfile28.out	398684	1325288	8526833	33131522	2791.59	INDET
benchmarkfile29.out	49370	144360	75085653	201035009	2853.47	INDET
benchmarkfile30.out	4640	13079	32747229	46037499	614.082	UNSAT

tabelului, aputem constata următoarele observații referitoare la performanța solverului MiniSat în cadrul testarii bencmark-urilor:

Rezultate INDET (Indeterminate): Majoritatea fișierelor (24 din 30) au returnat rezultatul "INDET", ceea ce sugerează că MiniSat nu a reușit să determine satisfiabilitatea pentru aceste fișiere într-un timp rezonabil. Acest lucru poate fi cauzat de complexitatea mare a problemelor (numărul mare de variabile și clauze).

Rezultate UNSAT (Unsatisfiable): Au fost obținute 4 fișiere cu rezultate "UNSAT", ceea ce indică faptul că MiniSat a reușit să stabilească că aceste probleme nu au soluție.

Rezultate SAT (Satisfiable): Două fișiere au returnat "SAT", adică MiniSat a găsit o soluție satisfiabilă pentru acestea. Ca exempleu fișierul benchmarkfile26.out a fost rezolvat extrem de rapid, în doar 33.7 secunde, în timp ce fișierul benchmarkfile23.out a avut un timp de 1178.5 secunde.

Daca raportam aceste rezultate cu resursele folosite de noi(laptop pe procesor M1 Pro cu 16 GB RAM) putem spune ca SAT solvers întâmpina dificultăți în a exploata complet arhitectura multi-core a unui procesor modern, deoarece aceștia pot să nu fie complet optimizați pentru a profita de paralelismul complet pe mai multe nuclee. De asemenea, M1 Pro poate avea un impact asupra performanței în funcție de modul în care MiniSat utilizează procesele de rulare. Și astfel întrucât MiniSat nu este neapărat paralelizabil în mod eficient pe mai multe nuclee (dependența puternică de procesarea secvențială în cadrul algoritmilor SAT poate limita performanța), procesorul M1 Pro ar putea să nu fie complet exploatat. De asemenea, în cazul unui număr mare de conflicte sau decizii, performanța poate fi afectată de complexitatea intrărilor și de resursele disponibile pentru fiecare proces de calcul.

O altă observație ar fi timpul limită de execuție care noi l-am setat de a fi 2880 secunde(48 de minute) poate fi o mică problemă deoarece diferă de timpul folosit in competiții care e de 5000 secunde, astfel posibil ca timpul setat de noi nu era destul pentru Minisat pentru a rezolva unele benchmark-uri ca rezultat obținând atât de multe rezultate INDET.

În concluzie, rezultatele sugerează că MiniSat are performanțe bune în identificarea instanțelor nesatisfiabile și satisfiabile în cazurile mai simple, dar întâmpină dificultăți semnificative atunci când se confruntă cu probleme mai complexe, caracterizate printr-un număr mare de variabile și clauze. Îmbunătățirea algoritmilor și a resurselor utilizate ar putea duce la o performanță mai bună în abordarea instanțelor "INDET", în special în diferite competiții cu limitări de timp mai stricte.

6 Afișare clauze învățate

Clauzele învățate contribuie semnificativ la eficiența solverului SAT, deoarece ele **reduc spațiul de căutare**, **previn apariția conflictelor viitoare** și **accelerează cautările**. Analiza clauzelor învățate ajută la îmbunătățirea înțelegerii

modului în care funcționează algoritmul și cum pot fi făcute optimizări suplimentare, de exemplu **îmbunătățirea euristicii de decizie** sau **testarea și evaluarea soluțiilor**.

Pentru realizarea acestui test vom apela la fișierul `Solver.cc` și vom rula MiniSat cu fisierul `Main.cc` din *core*, adica:

```
./minisat_core
```

Acest lucru este imposibil de realizat pe MiniSat-ul clasic fiindcă `SimpSolver.cc` are ca optimizare ștergerea tuturor clauzelor.

6.1 Modificarile realizate în codul sursă

Pentru implementarea acestei afisări vom adauga funcțiile de afișare a clauzelor învățate in fișierul Solve.cc

```
void Solver::printClause(const Clause& c) {
    for (int i = 0; i < c.size(); i++) {
        std::cout << (sign(c[i]) ? "-" : "")
        << var(c[i]) + 1 << " ";
    }
    std::cout << "0" << std::endl;
}

void Solver::printLearnedClauses() {
    std::cout << "Learned Clauses:" << std::endl;
    for (int i = 0; i < learnts.size(); i++) {
        printClause(ca[learnts[i]]);
    }
}
```

Funcția `printClause(const Clause& c)` este responsabilă pentru afișarea unei clauze. Se iterează prin fiecare literali din clauză, `c.size()` este dimensiunea clauzei, `sign(c[i])` este folosit pentru a determina semnul literarului (dacă este pozitiv sau negativ), `var(c[i]) + 1` returnează valoarea literarului (variabila) respectivă, ajustată cu $+1$ pentru a se conforma convenției CNF, unde numerele literale sunt indexate de la 1, nu de la 0.

Funcția `printLearnedClauses()` afișează toate clauzele învățate stocate în solver. Se parcurge vectorul `learnts`, care conține indexurile clauzelor învățate stocate în `ca` (un vector de clauze) și este afişat prin apelarea funcției `printClause(ca[learnts[i]])`, unde `ca[learnts[i]]` face referire la clauza învățată respectivă din vectorul `ca`

6.2 Teste pe exemple concrete

Vom analiza modificarea facută din două aspecte diferite, în prima parte vom rula pe MiniSat un exemplu CNF pentru analizarea clauzelor învățate și asigurarea că codul nostru face într-adevăr ceea ce ne așteptăm. În cea de-a doua parte vom

rula un fișier din benchmark-ul folosit, clauzele învățate le vom adauga în fișierul CNF după care vom analiza cum influențează clauzele învățate formula inițială.

Primul exemplu: Pentru prima parte a testării ne propunem spre utilizare formula de mai jos, preluată din[5]:

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

Alnalizând rezultatele obținute în urma rulării[2] observăm că MiniSat a învățat doua clauze noi:

$$(x_4 \vee x_1) \wedge (x_3 \vee x_1)$$

Prima clauză invățata rezultă din:

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

Unde putem observa clar ca formula să fie satisfiabilă trebuie ca **x1** sau **x3** sa fie **True**

Cea de-a doua clauza rezulta din:

$$(\neg x_3 \vee x_4 \vee x_1) \wedge (x_1 \vee x_4 \vee x_3)$$

Unde observam că pentru a primi valoarea de adevăr a formulei trebuie ca **x1** sau **x4** să fie **True**

Analiza formulei inițiale și a clauzelor învățate de MiniSat evidențiază eficiența algoritmului în simplificarea și reducerea spațiului de căutare. Prin adăugarea celor două clauze noi, MiniSat identifică restricții esențiale asupra valorilor variabilelor, cum ar fi necesitatea ca **x1**, **x3** și **x4** să fie adevărate pentru a satisface formula.

Al doilea exemplu: Pentru cea de-a doua parte a testării vom rula un fisier din benchmarkul pe care l-am rulat în MiniSat 5 și anume *benckmarkfile26.cnf*. Scopul final fiind să analizam cum clauzele învățate influențează timpul de execuție a fișierului **CNF**.

Pentru realizarea acestui test vom rula fișierul *benckmarkfile26.cnf* în mod canonic cum se ruleaza orice fișier **CNF** în MiniSat utilizând versiunea `./minisat_core`, după ce primim rezultatul notăm CPU Time, copiem clauzele noi învățate în fișierul benckmarkfile26.cnf, rulăm încă odată fișierul și notăm timp de execuție

Table 2. Rezultatele testului 2

File Name	Clauses	Conflict literals	Conflicts	Decisions	CPU Time(s)
init.out	12045456	40780	4315	15023347	62.1872
update.out	12049768	64005	5948	4581376	41.2531

Pe baza rezultatelor obținute în testul prezentat în Tabelul 2, rezultă următoarele:

Adăugarea clauzelor noi învățate în fișierul *benchmarkfile26.cnf* a avut un impact semnificativ asupra performanței soluționării în MiniSat. Observăm o reducere considerabilă a timpului de execuție, de la **62.1872** secunde în cazul rulării inițiale la **41.2531** secunde după actualizare, ceea ce reprezintă o îmbunătățire de aproximativ **30%**.

Deasemenea se reduce numărul de decizii efectuate (de la **15,023,347** la **4,581,376**), indicând o strategie mai eficientă de căutare datorită informațiilor suplimentare oferite de clauzele învățate.

Acest experiment demonstrează că integrarea clauzelor învățate în fișierul inițial contribuie semnificativ la optimizarea timpului de execuție și a resurselor utilizate.

References

[1] Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability: Conflict-Driven Clause Learning SAT Solvers. IOS Press (2008), [Online; accessed Jan. 25, 2025]
[2] Damyy17: GitHub - Damyy17/project-vf-team-8. https://github.com/Damyy17/project-vf-team-8 (2024), (accessed Jan. 25, 2025)
[3] MiniSat Page: MiniSat Page. http://minisat.se/ (2025), [Online; accessed Jan. 25, 2025]
[4] SAT Competition: SAT Competition. https://satcompetition.github.io/2024/tracks.html (2025), [Online; accessed Jan. 25, 2025]
[5] Vt.edu: 28.6. 3-cnf satisfiability — opendsa data structures and algorithms modules collection. https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/threeSAT.html (2024), accessed: Jan. 25, 2025
[6] Worrell, J.: Logic and Proof Hilary 2024. https://www.cs.ox.ac.uk/james.worrell/lecture06.pdf (2024), [Online; accessed Jan. 25, 2025]