Analizarea și Îmbunătățirea Performanței MiniSat în Rezolvarea Problemei SAT -Versiune draft-

Arpad-Adrian Precup, Elber Da, Szopuch Denis, Radu Bogdan

Universitatea de Vest din Timișoara,
Inginerie Software

Abstract. SAT (problema satisfiabilității propoziționale) este una dintre cele mai importante și provocatoare probleme în informatică, având numeroase aplicații în domenii precum verificarea formală, inteligența artificială și teoria complexității. Aceasta se referă la a determina dacă există o interpretare care să satisfacă o formulă booleană dată. Acest proiect explorează procesul de instalare și rulare al MiniSat, un SAT solver complet și de înaltă performanță, pe diferite sisteme de operare: Windows 10, o mașină virtuală ce rulează Pop!_OS (distribuție de Linux) și o mașină virtuală ce rulează Ubuntu. Proiectul documentează pașii de instalare și configurare a MiniSat, provocările întâmpinate de-a lungul utilizării acestuia, de la configurare până la debugging, precum și soluțiile folosite pentru a obține o rulare funcțională a aplicației în medii de dezvoltare diferite.

Keywords: MiniSat · SAT Solver · Benchmark · Code analysis ·

1 Introducere

Problema satisfiabilității, cunoscută sub acronimul SAT, este o problemă esențială în teoria complexității computaționale, fiind prima problemă demonstrată ca fiind NP-completă, prin intermediul teoremei Cook-Levin[1]. Aceasta constă în determinarea unei atriburi de adevăr pentru variabilele unei expresii logice astfel încât întreaga expresie să fie satisfăcută. Soluționarea problemelor SAT a devenit fundamentală într-o varietate de aplicații, inclusiv verificarea formală a software-ului și hardware-ului, inteligența artificială, probleme de optimizarea și bioinformatică.

MiniSat este un SAT solver disponibil public sub formă de repository pe GitHub[2]. A fost creat cu scopul de a ajuta la introducerea de noi membrii în comunitatea SAT prin furnizarea unui SAT solver de dimensiuni reduse (dar eficient) și documentat. Cu toate că are dimensiuni reduse, MiniSat este un SAT solver complet și, cel puțin pentru data apariției sale în 2003, suportă toate funcționalitățile de ultimă oră are solverelor, precum: învățare bazată pe conflicte, backtracking bazat pe conflicte și two-literal watch scheme[3]. MiniSat are la bază algoritmul CDCL (Conflict-Driven Clause Learning), care este eficientizat prin diferite strategii, spre exemplu simplificarea clauzelor înainte de a începe procesul de rezolvare CDCL și aplicarea diferitelor euristici, precum luby restarts[4]. Dezvoltatorii MiniSat pun la dispoziție și diferite documente de documentație, care să ajute noii utilizatori și dezvoltatori să înțeleagă felul în care MiniSat funcționează[5].

Acest proiect urmărește să documenteze procesul de instalare și configurare a MiniSat pe diverse platforme, punând accent pe provocările întâmpinate pe Windows și Linux, inclusiv rularea în mod debugging. De asemenea, include soluții pentru configurarea mediului de dezvoltare Visual Studio Code (VSCode), astfel încât să permită rularea și testarea programului într-un mod flexibil și reproductibil. Această lucrare va contribui la îmbunătățirea accesului la documentație pentru dezvoltatori și cercetători interesați de utilizarea MiniSat pe diferite platforme.

2 Instalare MiniSat

2.1 Instalare pe Windows 10

Pentru a instala MiniSat pe Windows 10, urmăm câțiva pași ce implică descărcarea codului sursă și configurarea unui mediu de dezvoltare compatibil. MiniSat este un proiect scris în C++, astfel că vom folosi Visual Studio Code sau un IDE similar pentru a permite compilarea și rularea codului C++. Este necesară instalarea MinGW, o suită de unelte care aduce funcționalitățile GNU Compiler Collection (GCC) pe Windows. Pașii necesari:

1. Descărcăm codul sursă MiniSat de pe GitHub-ul oficial al proiectului[2].
2. Instalăm MinGW pentru a putea compila codul C++.
3. În timpul instalării am întâmpinat erori legate de tipizare strictă; astfel, a fost necesar să folosim opțiunea 'CXXFLAGS="-fpermissive"' pentru a ignora aceste erori și a permite compilarea.

2.2 Instalare pe mașină virtuală ce rulează Pop!_OS

Instalarea MiniSat pe un sistem Linux este relativ simplă, datorită accesibilității librăriilor necesare și a suportului nativ pentru GNU Compiler Collection (GCC). Procesul se realizează prin terminal și implică actualizarea sistemului, descărcarea codului sursă, compilarea și rularea acestuia.

1. Actualizăm lista de pachete și instalăm dependențele necesare.
2. Descărcăm și compilăm codul MiniSat din repo-ul oficial.
3. Pentru a compila cu succes a fost necesară utilizarea 'CXXFLAGS="-fpermissive"', fără de care procesul întâmpina erori de tipizare strictă.
4. Pentru a putea face debugging, mediul de lucru trebuie configurat. Astfel, pentru Visual Studio Code trebuie creat un fișier 'launch.json', care permite rularea în modul debugging. Structura acestui fișier este exemplificată în secțiunea de README a repository-ului echipei noastre[6].

2.3 Instalare pe o Mașină Virtuală ce rulează Ubuntu

1. Deschidem terminalul și actualizăm pachetele sistemului cu comanda: "sudo apt update && sudo apt upgrade".
2. Instalăm dependențele necesare pentru compilare: "sudo apt install g++ make git"
3. Clonăm codul sursă al MiniSat de pe GitHub: "git clone https://github.com/niklasso/minisat.git"
4. Accesăm directorul minisat: "cd minisat" și compilăm MiniSat folosind: "sudo make CXXFLAGS="-fpermissive""
5. Verificăm instalarea rulând: "./minisat"

3 Provocări Întâmpinate

3.1 Instalarea pe Windows

Instalarea MiniSat pe Windows a întâmpinat mai multe provocări, în special din cauza diferențelor dintre compilatorul Windows și cerințele stricte de tipizare din codul sursă al MiniSat. Am utilizat flag-ul '-fpermissive' pentru a permite compilatorului să ignore aceste diferențe și să accepte codul.

3.2 Rularea în Mod Debugging

Pentru a permite debugging-ul în cea mai utilă formă, a fost necesară setarea opțiunii 'CXXFLAGS = -fpermissive -g -O0'. Flag-ul '-g' este folosit pentru a specifica compiler-ului să includă informații pentru debugging în codul compilat. Astfel, putem folosi metodele de debugging (spre exemplu 'step over', 'step into') ca și cum am executa codul sursă și nu fișierele binare rezultate prin compilare. De asemenea, flag-ul '-O0' specifică compiler-ului nivelul de optimizare pe care acesta sa îl realizeze (adică să nu se realizez nici o optimizare a codului la compile time, tocmai pentru a putea parcurge codul în modul de depanare). Am adăugat toate aceste informații în fișierul de README al proiectului nostru pe GitHub[6]

3.3 Crearea fișierului launch.json în VSCode

Pentru rularea MiniSat în Visual Studio Code, a fost necesară configurarea unui fișier 'launch.json' specific, astfel încât MiniSat să fie accesibil prin extensia de C++ debugging din VSCode, această extensie oferind suport pentru testare și analiză interactivă. Fișierul de input pentru MiniSat este specificat ca parametru în acest fișier. Un aspect important este ca fișierul ales ca input să fie disponibil pentru operații de citire. Acest lucru se poate realiza cu ajutorul rulării comenzii `chmod 777 <fisier_input>` înainte de a rula programul în mod debugging.

4 Benchmarks

Benchmark-urile utilizate pentru teste au fost preluate din competiția internațională SAT Competition 2024[7]. Familia selectată este `Coloring`, care constă în probleme de colorare a grafurilor, reprezentate în format DIMACS CNF.

4.1 Mediul de Execuție

Hardware

- **Procesor:** 11th Gen Intel® Core™ i3-1115G4 @ 3.00 GHz.
- **Memorie RAM:** 8 GB (7.80 GB utilizabili).
- **Stocare:** SSD Samsung 970 EVO Plus, 1TB, NVMe, M.2.
- **Grafică:** Procesor grafic integrat Intel® UHD Graphics.
- **Arhitectură:** Sistem pe 64 de biți.

Software

- **Sistem de operare:** Windows 11 Home (versiunea 24H2) cu Ubuntu 24.04.1 LTS (prin WSL).
- **Solver SAT:** MiniSAT, versiunea preluată din repository-ul oficial.

4.2 Rezultatele Testelor

Coloring	Time (s, h)	Result
18c0eb461bda29214bd43b84199a3b61-cliquecolouring_n31_k5_c4.sanitized	34624.8 s (9 h 37 m)	INDETERMINATE
67c533489a498495525efee429340958-cliquecolouring_n15_k9_c8.sanitized	74838.0 s (20 h 48 m)	INDETERMINATE
973d699ec01b88da869233a79aaa1912-cliquecolouring_n13_k9_c8.sanitized	27204.1 s (7 h 33 m)	INDETERMINATE
768956cc8d1f2d18ae1929f6bb26557a-cliquecolouring_n13_k8_c7.sanitized	30714.0 s (8 h 32 m)	INDETERMINATE
ac347c21ca0759079c0be9a758e4e924-cliquecolouring_n41_k5_c4.sanitized	16077.5 s (4 h 28 m)	INDETERMINATE
b172b4c218f1e44e205575d2b51e82c4-Schur_161_5_d38	30172.4 s (8 h 22 m)	INDETERMINATE
bbfe2b27182d2ee7fefdb557f458ac9c-cliquecolouring_n21_k6_c5.sanitized	25784.5 s (7 h 10 m)	INDETERMINATE
cdd131110acc861a5a01fae6c4936c91-6g_6color_366_050_04	Unknown	KILLED

Table 1. Rezultatul rulării benchmark-urilor din familia `Coloring`

În repository-ul echipei noastre de GitHub[6], în folderul `benchmarks`, se pot vedea mai multe detalii despre rularea fiecăruia dintre teste, cum ar fi numărul de conflicte, numărul de restart-uri, numărul de decizii luate sau memoria folosită.

4.3 Observații

- În cadrul executării aplicației, s-au utilizat **1 GB RAM** dintr-un total de **8 GB**, ceea ce reprezintă aproximativ **12.5%** din memoria disponibilă.
- Utilizarea procesorului (**CPU**) a fost de **36%** pe parcursul execuției aplicației.
- Având în vedere că sistemul nu prezintă un nivel ridicat de performanță, la testul final realizat cu MiniSat, fișierul a avut o dimensiune semnificativă, astfel încât procesul a fost **întrerupt (KILLED)**, iar rezultatele nu au fost nici **UNSAT**, **SAT** sau **INDETERMINATE**.

5 Codul Minisat

5.1 Structura fizică

Proiectul MiniSat are următoare structură:

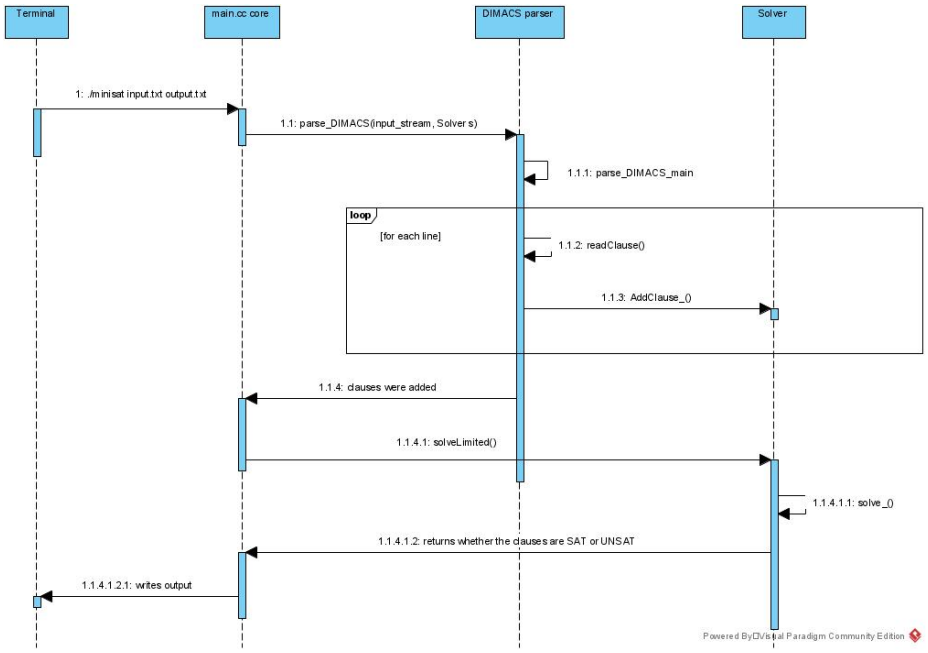
1. `minisat/mtl/` - reprezintă Mini Template Library-ul asociat MiniSat. Conține diferite structuri de date generice care fac posibilă utilizarea MiniSat fără nevoia altor dependințe. În acest directory se găsesc, spre exemplu, implementări pentru structuri de date precum: Maps, cozi și vectori.

2. `minisat/utils/` - conține helpere pentru cod. Spre exemplu, aici se găsesc metodele folosite pentru a interacționa cu sistemul I/O al mașinii pe care MiniSat este rulat și metode generice de parsat text, de exemplu: trecerea peste spațiile albe, trecerea peste o linie.
3. `minisat/core/` - Versiunea core a MiniSat. Cu ajutorul metodelor și structurilor de date prezente în fișierele anterior menționate, aici are loc procesul de SAT solving. Aceasta este versiunea asupra căreia ne-am concentrat atenția (precizăm acest lucru deoarece există și o versiune cu mai multe îmbunătățiri, versiunea `simp`).
   (a) `Dimacs.h` - Un fișier de tip header, care conține trei metode: **parse_DIMACS**, **parse_DIMACS_main** și **readClause**. Dintre acestea, doar metoda **parse_DIMACS** este apelată din exteriorul acestui fișier, aceasta fiind apelată din Main.cc pentru a insera problema (care este reprezentată într-un fișier text al cărui conținut trebuie să respecte formatul DIMACS) în Solver.
   (b) `Main.cc` - Punctul de intrare în program. Acesta acceptă diferiți parametrii care pot fi utilizați pentru a particulariza execuția codului. Spre exemplu, prin folosirea parametrului **"verb"** pot fi selectate trei nivele de verbozitate, care modifică numărul de log-uri afișate pe parcursul rulării programului. Alți parametrii care pot fi setați: limită de timp pentru rulare (folosind parametrul **"cpu_lim"**), limită memorie folosită (folosind parametrul **"mem_lim"**).
   (c) `Solver.cc` - Conține codul pentru Solver. Include metode care crează noi variabile, adaugă clauze și rezolvă problema. Conține de asemenea și diferiți membrii precum modelul, acesta reprezentând o soluție a problemei, dacă aceasta a fost găsită.
   (d) `Solver.h` - header-ul pentru clasa Solver.
   (e) `SolverTypes.h` - Fișier header care conține definiția mai multor tipuri de date folosite de către Solver, printre care clasa **Clause** și struct-ul **Lit**. Aceste structuri de date sunt prezentate detaliat în secțiunea `Structuri de date`.
4. `minisat/simp/` - Un solver extins care are capabilități pentru aplicarea de simplificări, fapt care duce la o performață mai bună. Această versiune este cea compilată implicit când se execută comanda `make` în terminal. Dacă se dorește compilarea versiunii core, este nevoie de rularea comenzii `make cr`.

5.2 Structura arhitecturală

Componenta principală arhitecturală o reprezintă clasa **Solver**. Odată inserată problema în solver (folosind metoda **parse_DIMACS**), acesta este capabil de a o rezolva prin intermediul metodei **solve_**. Informații amănunțite despre cum funcționează metoda **solve_** sunt disponibile în secțiunea dedicată `metodelor remarcabile`.

La nivel de diagramă de secvență, fluxul de date, de la rulare MiniSat din terminal până la afișarea output-ului, este următorul:



5.3 Structuri de date

Datorită naturii problemei abordate de MiniSat, au fost declarate mai multe tipuri speciale de structuri de date peste care vom trece în această secțiune.

– **Var** - este un alias pentru un număr întreg (integer). Variabilele trebuie alese de la 0 la N pentru a putea fi folosite ca indecși în vectori. Totodată, se declară și o instanță specială de tipul **Var** care reprezintă o variabilă neasignată, care are valoare -1. Aceste variabile reprezintă fiecare literal parsat din fișierul de input, cu o mică ajustare: variabilele de tip **Var**, fiind folosite ca indecși de vectori nu pot avea valori negative. De asemenea, trebuie să înceapă de la valoarea 0. Având în vedere că în formatul DIMACS literalii încep de la valoare absolută 1, valoare **Var** asociată se calculează prin decrementarea valorii absolute a literalului. Astfel, spre exemplu, dacă literalul parsat este ”-1”, valoare **Var** asociată este 0.

– **Lit** - este un struct care, conținând o singură valoare de tip integer, este capabil să encodeze un literal cu tot cu semnul asociat, folosindu-se de semnul literalului parsat și de valoarea **Var** asociată acestuia. Acest lucru este făcut posibil prin modul ingenios prin care valoarea din struct-ul **Lit** este calculată. Aceasta este calculată ca fiind dublul valorii **Var** asociate la care se adaugă 1 daca semnul literalului este negativ. Astfel, fiecărei posibile variabile **Var** îi sunt asociate două valori: `2 * var` și `2 * var + 1` (dacă semnul literalului este negativ).

Așadar se ajunge ca numărul întreg stocat în **Lit** să aibă encodat pe cel mai puțin semnificant bit valoare semnului, iar pe restul de biți valoarea **Var** de la care provine.
Exemplu: Dacă literalul parsat este -7, atunci **Var**-ul asociat este 6. Luând în considerare și semnul literalului, variabila stocată în **Lit** va fi 13 (6 * 2 + 1). în reprezentare binară 13 este 1101, ultimul bit (1) reprezentând faptul că literalul este negat, iar restul reprezentând valoare **Var** asociată 0b110 = 6

- **Clause** - este o clasă care reprezintă o clauză. Conține un vector de **Lit** și diferite câmpuri care stochează date despre clauză cum ar fi dacă a fost învățată sau dimensiunea acesteia.

- **VMap** - o structură de date folosită în multe câmpuri din clasa Solver pentru a stoca date de tip `cheie -> valoare`, unde cheia este de tip **Var**. Exemple de astfel de câmpuri în solver sunt: **assigns** (stochează asignările curente pentru fiecare variabilă) și **polarity** (stochează polaritatea preferată pentru fiecare variabilă).

5.4 Metode remarcabile

În această secțiune sunt prezentate câteva dintre metodele importante folosite pentru a rezolva problema satisfiabilității în MiniSat. Pentru metodele prezentate în această secțiune am realizat diagrame de activitate asociate, care pot fi vizualizate în directorul `diagrams` din repository-ul echipei noastre[6].

Solve - Metoda care implementează strategia de rezolvarea a problemei la nivelul cel mai superior. Această metoda folosește metoda **Search** și o restartează până când se ajunge la o concluzie sau "bugetul" (numărul de conflicte și propagări permise) este depășit.

Search - cea mai complexă procedură. În această procedură, variabilele sunt alese și le sunt atribuite valori (aceste asignări sunt stocate într-un vector de **Lit** numit **trail**. De asemenea, într-un vector de integers numit **trail_lim** se reține indecșii de la care încep nivele diferite de decizie în **trail**), până în momentul în care propagarea determină un conflict, punct în care se creează o clauză conflict care este adăugată Solver-ului ca și clauză învățată. Apoi se face backtracking, anulând asignările făcute la nivele mai înalte decât nivelul de la care a pornit conflictul (folosind informațiile din **trail** și **trail_lim**). Odată procesul de backtracking terminat, se reîncepe cu procesul de propagare.

Propagate - Metoda care se ocupă de propagarea unității. 80% din timpul de rulare MiniSat este datorat propagării, astfel această metodă poate fi considerată principală țintă când se dorește îmbunătățirea performanței[8]. Această metoda folosește two-literal watch scheme pentru a eficientiza propagarea unitară.

```
loop
    propagate()    – propagate unit clauses
    if not conflict then
        if all variables assigned then
            return SATISFIABLE
        else
            decide()    – pick a new variable and assign it
    else
        analyze()    – analyze conflict and add a conflict clause
        if top-level conflict found then
            return UNSATISFIABLE
        else
            backtrack()    – undo assignments until conflict clause is unit
```

Fig. 1. Pseudocod pentru metoda Search[8]

6 Repository GitHub

Fork-ul nostru de MiniSat poate fi accesat la https://github.com/aprecup14/minisat [6]. Acesta conține pe lângă codul de MiniSat (căruia i-am actualizat părți din documentația scrisă sub formă de comentarii) următoarele foldere:

- `Benchmarks` - fișier care conține rezultatele rulării benchmark-urilor. În acest fișier se pot observa două fișiere care corespund membrilor echipei care s-au ocupat de rularea testelor. În aceste fișiere se pot observa afișările care se produc în timpul rulării MiniSat. De asemenea, există și un fișier `Specs.txt` care precizează specificațiile mașinii care a rulat benchmark-urile.
- `Diagrams` - Acest fișier conține diagramele UML pe care le-am creat pentru a reprezenta diferite fluxuri de date în MiniSat. Pentru realizarea diagramelor am folosit `Visual Paradigm`.
- `Latex overleaf` - Fișier în care încărcăm codul sursă pentru documentul latex.
- `README.md` - Fișier pe care l-am actualizat cu diferite informații legate de cum trebuie rulat MiniSat. Spre exemplu, instrucțiuni pentru a putea rula MiniSat în mod debugging în `Visual Studio Code`.

7 Responsabilități

Fiecare membru al echipei a avut responsabilități:

- `Arpad-Adrian Precup` -
  - Crearea mediului propice pentru a realiza comunicarea și lucrul în echipă. Crearea repository-ului de GitHub, crearea proiectului în Overleaf, adăugarea

celorlalți membrii ai echipei ca și contribuitori la aceste proiecte. Adăugarea de foldere pentru încărcarea benchmark-urilor, diagramelor și a codului sursă latex în GitHub.
- Rularea MiniSat ca developer (folosind codul sursă). Rularea MiniSat în mod debugging pentru a observa fluxul de date.
- Înțelegerea modului de funcționare a MiniSat. Identificare utilizării algoritmului CDCL cu diferite ajustări/îmbunătățiri.
- Crearea documentației tehnice și a diferitelor diagrame UML.
- Redactat secțiune latex care prezintă cum funcționează codul MiniSat.
- Revizuit document latex și ajutat membrii echipei la nevoie.

– `Denis Szopuch`
- Redactat abstract și introducere care să explice contextul proiectului, tehnologiile utilizate și importanța acestuia.
- Scrierea unui rezumat clar și concis al proiectului, subliniind scopul și obiectivele.
- Instalare MiniSat.
- Crearea unui ghid pas cu pas pentru instalarea MiniSat, inclusiv cerințele și pașii necesari pentru a-l rula pe diferite platforme.
- Redactarea secțiunii de concluzii, sintetizând rezultatele obținute și impactul implementării.

– `Elber Da`
- Instalare MiniSat.
- Rularea benchmark-ului de coloring
- Observarea și documentarea comportamentului MiniSat în timpul rulării benchmark-urilor.
- Redactarea secțiunii legată de benchmarks.
- Încărcarea rezultatelor benchmark-urilor pe GitHub.

– `Cristian Radu`
- Instalare MiniSat.
- Rularea benchmark-ului de coloring
- Observarea și documentarea comportamentului MiniSat în timpul rulării benchmark-urilor.
- Redactarea secțiunii legată de benchmarks.
- Încărcarea rezultatelor benchmark-urilor pe GitHub.

Pe lângă contribuțiile individuale, am păstrat legătura constant și am avut discuții despre stadiul sarcinilor fiecăruia prin intermediul unui grup de WhatsApp. Am avut de asemenea ședinte de lucru comune pe Google Meet prin care am conectat părțile lucrate individual de fiecare membru.

8 Concluzii

În cadrul acestui proiect, am investigat și utilizat MiniSat pentru a rezolva probleme SAT, utilizând o familie de probleme de benchmark din competiția SAT 2024[7] pentru a analiza performanța acestuia. De asemenea, versiunea draft a proiectului a avut ca scop familiarizarea cu MiniSat și înțelegerea algoritmilor de rezolvare a problemei SAT folosiți de MiniSat.

Am identificat principalul algoritm de rezolvare a problemei SAT folosit de MiniSat, CDCL (Conflict-Driven Clause Learning), care este implementat eficient în MiniSat, cu tehnici care îmbunătățesc performanța cum ar fi two-literal watch scheme și diferite euristici de resetare precum luby restarts. Aceste tehnici sunt esențiale pentru îmbunătățirea performanței MiniSat în probleme de satisfiabilitate, iar analiza detaliată a codului sursă ne-a permis să înțelegem cum aceste metode sunt integrate și optimizate. De asemenea, am identificat zonele unde MiniSat poate fi îmbunătățit, în special zona de propagare care ocupă 80% din timpul de rulare al MiniSat[8].

Un aspect important al proiectului a fost experimentarea cu benchmark-uri SAT din competiția SAT 2024[7]. Rezultatele obținute au arătat performanțe bune ale MiniSat în rezolvarea unor instanțe de dimensiuni reduse, dar și provocări legate de timpul de execuție și eficiența în cazul unor formule extrem de mari sau complexe.

Provocările întâmpinate au inclus înțelegerea complexității codului sursă MiniSat și integrarea acestuia într-un mediu de testare eficient. De asemenea, documentația limitată sau învechită a fost o barieră în calea înțelegerii complete a tuturor funcționalităților MiniSat, ceea ce a făcut necesar un studiu suplimentar al literaturii de specialitate pentru a clarifica anumite concepte avansate.

În ansamblu, proiectul ne-a permis să aprofundăm cunoștințele teoretice legate de SAT solving și să ne dezvoltăm abilitățile practice de programare, analiză de performanță a software-ului și capacitățile de colaborare în echipă.

References

1. UBC Department of Computer Science, `https://www.cs.ubc.ca/~condon/cpsc506/handouts/Cook-Levin.pdf`
2. Repository-ul oficial MiniSat, `https://github.com/niklasso/minisat`
3. MiniSat Home Page, `http://minisat.se/MiniSat.html`
4. Luby restart, `https://choco-solver.org/docs/solving/restarts/#luby`
5. MiniSat documentation page, `http://minisat.se/Papers.html`
6. Fork-ul nostru de MiniSat, `https://github.com/aprecup14/minisat`
7. SAT competition home page, `https://satcompetition.github.io/2024/`
8. An Extensible SAT-solve paper, `http://minisat.se/downloads/MiniSat.pdf`