Reproducerea rezultatelor de la competiția SAT2025 pentru două benchmark-uri non-triviale

Coordonator:
Conf. Dr. Mădălina Erașcu

Autori:
Denis Gall, Daniel Ichim, Mark Corojor, and Alexandru Teleman

Universitatea de Vest din Timișoara
Programul de studii Inginerie Software

Rezumat Lucrarea prezintă o analiză comparativă a performanței solverului SAT MiniSat, bazat pe algoritmul Conflict-Driven Clause Learning (CDCL), utilizând benchmark-uri din familia *schedule*. Obiectivul principal constă în evaluarea influenței parametrului de decădere a variabilelor (`var_decay`) din euristica de decizie VSIDS (Variable State Independent Decaying Sum) asupra eficienței procesului de rezolvare. În acest scop, sunt analizate două instanțe non-triviale din competiția SAT 2025, iar configurația de referință (`var_decay`=0.95) este comparată cu valori alternative din intervalul $[0.90, 0.99]$. Evaluarea se bazează pe timpul total de execuție, numărul de decizii și rata conflictelor, urmărind identificarea unei setări care reduce timpul de rezolvare și îmbunătățește eficiența mecanismelor CDCL, în special backjumping-ul. Rezultatele experimentale sugerează că o calibrare fină a „memoriei" euristice, controlată de `var_decay`, poate conduce la îmbunătățiri semnificative ale performanței MiniSat pentru această clasă de probleme aplicative.

Keywords: SAT · MiniSat · CDCL · VSIDS · `var_decay` · *schedule* · Benchmark

1 Introducere

Problema satisfiabilității booleene (SAT) ocupă un rol central în informatică teoretică și aplicată, fiind prima problemă demonstrată ca NP-completă. Solverele SAT moderne sunt utilizate extensiv în domenii precum verificarea hardware, criptografia și planificarea automată, unde eficiența algoritmică este esențială. Lucrarea analizează performanța solverului MiniSat, o implementare consacrată a algoritmului Conflict-Driven Clause Learning (CDCL), cu accent pe influența euristicii de decizie VSIDS. În mod specific, este investigat impactul parametrului `var_decay` asupra timpului de rezolvare pentru instanțe din familia *schedule* utilizate în cadrul competiției SAT 2025.

2 Descrierea problemei și arhitectura MiniSat

MiniSat este un solver SAT destinat rezolvării formulelor booleene în Formă Normală Conjunctivă (CNF). Deși își are originea în algoritmul clasic **DPLL** (Davis–Putnam–Logemann–Loveland), MiniSat implementează extensia modernă **Conflict-Driven Clause Learning (CDCL)**, care constituie baza performanței solverelor SAT contemporane. Analiza arhitecturii și a codului sursă (`Solver.cc`) evidențiază mai multe mecanisme fundamentale pentru eficiența procesului de rezolvare, prezentate în continuare.

2.1 De la DPLL la CDCL

Spre deosebire de backtracking-ul cronologic utilizat în DPLL, MiniSat aplică **backtracking non-cronologic** (*backjumping*). În momentul apariției unui conflict, mecanismul de analiză a conflictului identifică cauza acestuia și generează o *clauză învățată* (*learnt clause*) prin rezoluție. Această clauză este adăugată bazei de date și permite solverului să revină direct la nivelul de decizie relevant, evitând explorarea repetată a acelorași ramuri ale spațiului de căutare.

2.2 Euristica de decizie VSIDS

Selecția variabilelor este realizată prin euristica **VSIDS** (Variable State Independent Decaying Sum), care atribuie fiecărei variabile un scor de activitate. Variabilele implicate frecvent în conflicte primesc un scor mai mare, fiind astfel prioritizate în procesul de decizie. Pentru a controla influența conflictelor mai vechi, scorurile de activitate sunt supuse unui proces de degradare multiplicativă, utilizând parametrul `var_decay`. Acest mecanism permite ajustarea echilibrului dintre exploatarea informației recente și menținerea unui istoric al conflictelor, având un impact direct asupra comportamentului euristicii și, implicit, asupra performanței solverului.

2.3 Propagarea și structuri de date

Propagarea constrângerilor este optimizată prin utilizarea tehnicii **Two-Watched Literals**, implementată în funcția `propagate()`. Această metodă reduce semnificativ costul propagării prin monitorizarea a doar doi literali per clauză, evitând evaluarea completă a clauzelor la fiecare asignare. În consecință, accesul la memorie este redus, iar eficiența globală a solverului este îmbunătățită.

3 Conexiunea cu conceptele teoretice din curs

Lucrarea se bazează pe conceptele fundamentale de SAT solving prezentate în cadrul disciplinei Verificare Formală, cu accent pe arhitectura solverelor CDCL și pe modelarea problemelor prin constrângeri booleene.

3.1 Arhitectura CDCL și graful de implicație

În cadrul cursului de Verificare Formală [2] este prezentat algoritmul **Conflict-Driven Clause Learning (CDCL)**, utilizat de solvere SAT moderne, inclusiv MiniSat. În CDCL, solverul alternează decizii și propagare unitară (BCP), iar atunci când apare un conflict declanșează analiza acestuia pentru a extrage informație utilă pentru pașii următori.Analiza conflictelor este formalizată prin **graful de implicație**, un graf orientat aciclic în care nodurile reprezintă literali asignați la diferite niveluri de decizie, iar muchiile indică dependențele introduse prin propagare unitară din clauze [2]. Când o clauză devine nesatisfiabilă, conflictul este reprezentat printr-un nod special, iar pe baza grafului se aplică reguli de **rezoluție** [3] pentru a deriva o *clauză învățată*. Aceasta este adăugată formulei și permite *backjumping* non-cronologic către un nivel relevant, reducând reexplorarea configurațiilor care duc la același conflict.

3.2 Modelarea problemelor de planificare

Benchmark-urile din familia *schedule* utilizate în experimente reprezintă instanțe concrete ale problemelor de planificare și alocare de resurse discutate în cadrul cursului [4]. Astfel de probleme pot fi modelate prin constrângeri CNF de tipul *at least one* ($\bigvee x_{i,j}$) și *at most one* ($\neg x_{i,j} \vee \neg x_{i,k}$). Numărul ridicat al acestor constrângeri conduce la formule cu un volum semnificativ de clauze, justificând necesitatea utilizării unei euristici de decizie eficiente, precum VSIDS, pentru explorarea eficientă a spațiului de căutare.

4 Instalare și configurare

Experimentele au fost rulate în **Windows Subsystem for Linux (WSL)** folosind distribuția **Ubuntu**. Solverul **MiniSat** a fost compilat din surse, conform instrucțiunilor din fișierul `README`, utilizând `gcc/g++` și biblioteca `zlib`. Compilarea a fost realizată în modul *Release*, pentru a obține o versiune optimizată pentru evaluarea performanței.
Executabilul rezultat, `minisat_release`, a fost utilizat pentru rularea instanțelor SAT, iar parametrii au fost controlați prin opțiuni de linie de comandă. Un exemplu de rulare este:

```
minisat benchmarks/problema1.cnf -var-decay=0.98 -no-luby
```

5 Rezultate experimentale

5.1 Problema 1: Instanță din familia *schedule*

Prima instanță analizată aparține familiei *schedule* (planificare/alocare de resurse). Formula CNF conține **145 943** variabile și **625 454** clauze, corespunzând unui spațiu de căutare de dimensiuni ridicate.

Configurația implicită vs. cele mai bune rezultate observate Configurația implicită a solverului, corespunzătoare valorii `var_decay=0.95`, a fost utilizată drept linie de bază pentru comparație. Pentru această setare au fost înregistrate **133 820** conflicte și un timp CPU de **46.42** secunde. În urma testării valorilor `var_decay` în intervalul $[0.90, 0.99]$, cea mai bună performanță obținută prin ajustarea exclusivă a acestui parametru a fost observată la `var_decay=0.98`, unde numărul de conflicte a scăzut la **6 115** (o reducere de aproximativ **95%**), iar timpul de execuție la **2.75** secunde (o reducere de aproximativ **94%**). Per total, cea mai bună configurație observată pentru această instanță a fost combinația `var_decay=0.98` și opțiunea `-no-luby`, care a redus timpul de execuție la doar **0.68** secunde. Acest rezultat indică faptul că, pentru această instanță, dezactivarea restart-urilor de tip Luby permite o explorare mai eficientă a spațiului de căutare atunci când este utilizată o rată ridicată de conservare a scorurilor de activitate.

Tabela 1. Performanța MiniSat pentru Problema 1 (familia *schedule*)

`var_decay`	**Conflicte**	**Timp CPU (s)**	**Observație**
0.90	178 209	62.92	Degradare rapidă a activităților
0.95 (referință)	133 820	46.42	Configurație implicită
0.98	**6115**	**2.75**	Cea mai bună setare `var_decay`
0.98 + `-no-luby`	1 164	**0.68**	Cea mai bună configurație observată
0.99	156 331	56.84	„Memorie” prea persistentă

5.2 Problema 2: Instanță din familia *schedule*

A doua instanță analizată (**12 845** variabile și **51 681** clauze) s-a dovedit mai sensibilă la setările care păstrează prea mult istoricul euristic, ceea ce a influențat vizibil timpul de execuție și numărul de conflicte.

Configurația implicită vs. cele mai bune rezultate observate Pentru `var_decay=0.95` (configurația implicită) au fost obținute **41.38** secunde și **510 895** conflicte. În testele în care a fost ajustat doar parametrul `var_decay` (păstrând restul setărilor implicite), cea mai bună valoare a fost `var_decay=0.97`, cu **10.31** secunde și **218 064** conflicte. Comparativ cu Problema 1, `0.97` este mai eficient decât `0.98`, sugerând că pentru această instanță este utilă o adaptare mai rapidă a scorurilor de activitate. În plus, `var_decay=0.99` a condus la o degradare accentuată a performanței (timp de execuție > 200 secunde). Per total, cea mai bună configurație observată pentru această instanță a fost combinația `-var-decay=0.94 -no-luby`, care a redus timpul de execuție la **5.49** secunde, depășind rezultatele obținute prin ajustarea exclusivă a `var_decay`.

Tabela 2. Performanța MiniSat pentru Problema 2 (familia *schedule*)

`var_decay`	**Conflicte**	**Timp CPU (s)**	**Observație**
0.90	959 047	58.63	Degradare rapidă a activităților
0.95 (referință)	510 895	41.38	Configurație implicită
0.97	**218064**	**10.31**	Cea mai bună setare `var_decay`
0.98	467 620	24.30	Adaptare mai lentă
0.99	3 501 448	237.483	Degradare accentuată a performanței
0.94 + `-no-luby`	**91644**	**5.49**	Cea mai bună configurație observată

6 Limitări și provocări

În desfășurarea experimentelor au fost observate mai multe aspecte care influențează interpretarea și reproductibilitatea rezultatelor:

- **Sensibilitatea hiper-parametrilor:** variații minore ale parametrului `var_decay` (de exemplu, 0.97 vs. 0.98) pot produce diferențe semnificative în timpul de execuție și în numărul de conflicte, ceea ce indică o dependență puternică de reglajul euristicii.
- **Constrângeri de resurse:** anumite setări neadecvate (de exemplu, `var_decay`=0.99 pentru Problema 2) pot conduce la creșteri substanțiale ale timpului de rulare (până la depășirea pragurilor de timp) și la un consum ridicat de resurse, limitând numărul de configurații ce pot fi evaluate practic.
- **Dependența de instanță:** rezultatele sugerează că nu există o valoare universal optimă pentru `var_decay`; setarea eficientă depinde de structura instanței și de distribuția conflictelor, necesitând ajustare specifică pentru fiecare benchmark analizat.

7 Link proiect

Codul sursă modificat, scripturile de rulare și raportul complet sunt disponibile pe GitHub:

`https://github.com/galldenis/vfproject`

Bibliografie

1. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004).
2. Erașcu, M.: Curs 5-6: SAT Solving. Verificare Formală, Universitatea de Vest din Timișoara (2025).
3. Erașcu, M.: Curs 2: Logica propozițională. Verificare Formală, Universitatea de Vest din Timișoara (2025).

4. Erașcu, M.: Curs 4: Modelarea problemelor folosind logica propozițională. Verificare Formală, Universitatea de Vest din Timișoara (2025).
5. SAT Competition 2025 Benchmarks, http://satcompetition.org/
6. Marques-Silva, J.P., Sakallah, K.A.: GRASP: A Search Algorithm for Propositional Satisfiability. IEEE Trans. Comput. 48(5), 506–521 (1999)