Analizarea și Îmbunătățirea Performanței MiniSat în Rezolvarea Problemei SAT

Arpad Precup[1], Elber Da[2], Szopuch Denis[3] Radu Bogdan[4]

Universitatea de Vest din Timișoara,
Inginerie Software

1 Abstract

Acest proiect explorează procesul de instalare și rulare al MiniSat, un solver SAT de înaltă performanță, pe sisteme de operare variate, cu un focus specific pe Windows 10, Linux și pe o mașină virtuală cu Ubuntu. Problema SAT (satisfiabilitatea formulelor logice) este una dintre cele mai importante în informatică, având numeroase aplicații în domenii precum verificarea formală, inteligența artificială și teoria complexității. Proiectul documentează pașii de instalare și configurare a MiniSat, provocările întâmpinate în setările de debugging și compilare, precum și soluțiile aplicate pentru a obține o rulare funcțională a aplicației în medii de dezvoltare diferite.

Keywords: MiniSat · SAT Solver · Instalare · Debugging ·

2 Introducere

Problema satisfiabilității, cunoscută sub acronimul SAT, este o problemă esențială în teoria complexității computaționale, fiind prima problemă demonstrată ca fiind NP-completă. 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, optimizarea și bioinformatica.

MiniSat este un solver SAT de înaltă performanță, dezvoltat pentru a participa în competițiile SAT și pentru a rezolva probleme complexe de satisfiabilitate. MiniSat utilizează algoritmi avansați, precum DPLL (Davis-Putnam-Logemann-Loveland) și CDCL (Conflict-Driven Clause Learning), pentru a explora eficient spațiul de căutare al soluțiilor posibile. Datorită eficienței sale și accesibilității ca proiect open-source, MiniSat este adesea utilizat ca reper în benchmark-urile SAT și este o alegere populară pentru studii și cercetări în domeniul satisfiabilității.

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 platforme multiple.

3 Descrierea Problemei SAT

Problema satisfiabilității, cunoscută sub numele de SAT, constă în determinarea valorilor de adevăr pentru variabilele unei expresii logice astfel încât întreaga expresie să fie satisfăcută, adică evaluată ca adevărată. În limbajul logicii propoziționale, o expresie este satisfiabilă dacă există o atribuire a valorilor variabilelor sale care face întreaga expresie să fie adevărată. Această problemă are o importanță majoră în teoria complexității deoarece este prima problemă cunoscută ca fiind NP-completă, fiind astfel esențială în clasificarea și înțelegerea problemelor computaționale complexe.

Algoritmii utilizați pentru rezolvarea problemelor SAT includ tehnici diverse, cei mai reprezentativi fiind algoritmul DPLL și tehnica CDCL. Algoritmul DPLL (Davis-Putnam-Logemann-Loveland) utilizează tehnici de căutare recursivă, reducând treptat problema prin atribuiri succesive ale variabilelor și eliminarea condițiilor imposibile. CDCL (Conflict-Driven Clause Learning) este o optimizare a DPLL, care învață din conflictele întâlnite în timpul căutării soluției, reducând astfel numărul de căi explorate.

Aplicațiile problemelor SAT sunt vaste și acoperă domenii variate, de la verificarea formală a corectitudinii sistemelor software și hardware până la inteligența artificială și bioinformatică. MiniSat, fiind un solver SAT foarte eficient, este utilizat adesea pentru a testa diverse benchmark-uri SAT și pentru a valida probleme complexe în cercetare și dezvoltare.

4 Instalare MiniSat

4.1 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.

1. Descărcăm codul sursă MiniSat de pe GitHub-ul oficial al proiectului. 2. Instalăm MinGW pentru a putea compila codul C++. 3. Configurăm mediul de lucru în VSCode și adăugăm un fișier 'launch.json' pentru a permite rularea în modul debugging. 4. 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.

4.2 Instalare pe Linux

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 rularea cu succes a compilării a fost necesară utilizarea 'CXXFLAGS="-fpermissive"', fără de care procesul întâmpina erori de tipizare strictă. 4. Configurăm un fișier 'launch.json' pentru rularea în VSCode, care facilitează debugging-ul și rularea aplicării MiniSat într-un mod ușor de gestionat.

4.3 Instalare pe o Mașină Virtuală cu Ubuntu

După ce Ubuntu este instalat pe mașina virtuală, deschidem terminalul și actualizăm pachetele sistemului cu comanda: "sudo apt update && sudo apt upgrade" Apoi, instalăm dependențele necesare pentru compilare: "sudo apt install g++ make git" Clonăm codul sursă al MiniSat de pe GitHub: "git clone https://github.com/niklasso/minisat.git" Accesăm directorul minisat: "cd minisat" Și compilăm MiniSat folosind: "sudo make CXXFLAGS="-fpermissive"" După finalizare, verificăm instalarea rulând: "./minisat"

5 Provocări Întâmpinate

5.1 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.

5.2 Rularea în Mod Debugging

Pentru a permite debugging-ul, a fost necesară setarea opțiunii 'CXXFLAGS = -fpermissive -g -O0'. Aceste opțiuni au permis intervenții de depanare detaliată, iar MiniSat ar fi generat erori fără aceste flag-uri. Aceasta a făcut debugging-ul posibil pe Windows și Linux.

5.3 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, oferind suport pentru testare și analiză interactivă.

6 Benchmarks

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

6.1 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.

6.2 Rezultatele Testelor

Table 1. Coloring Results

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

6.3 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 **USAT**, **SAT** sau **INDETERMINATE**.

7 Codul Minisat

MiniSat este un SAT solver disponibil public sub formă de repository pe GitHub (TODO adaugă link spre repo). 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 (TODO adaugă biografie din http://minisat.se/MiniSat.html). 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.

7.1 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: 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.
   (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. Conține metode care crează noi variabile, adaugă cluaze, rezolvă problema. Conține de asemenea și diferiți membrii precum modelul, acesta reprezentând o soluție, dacă aceasta a fost găsită.
   (d) `Solver.h` - header-ul pentru clasa Solver.

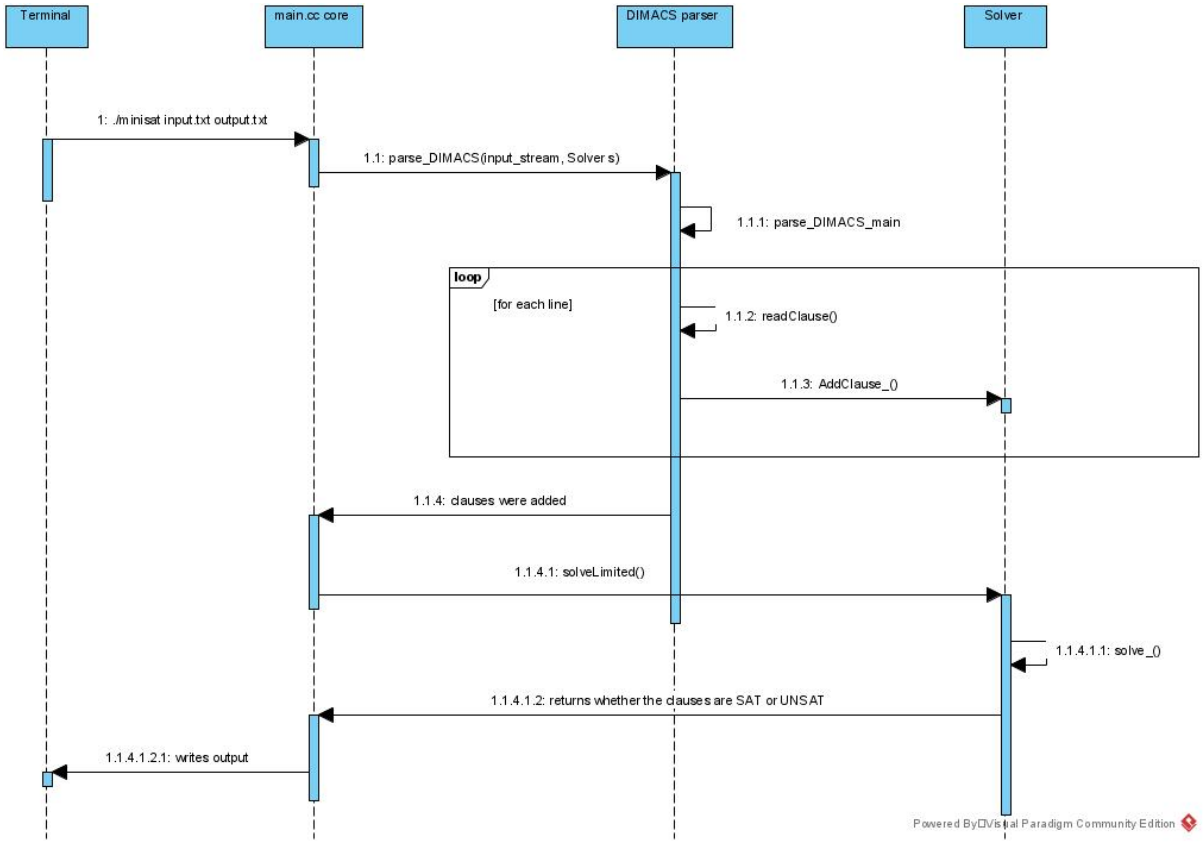
(e) `SolverTypes.h` - Fișier header care conține definiția mai multor tipuri de date folosite pentru Solver, printre care clasa **Clause** și struct-ul **Lit**. Este prezentat mai pe larg ce reprezintă aceste structuri de date în capitolul (TODO adauga link care sa duca la sectiunea structuri de date).

4. `minisat/simp/` - Un solver extins care are capabilități pentru aplicarea de simplificări, fapt care ajută la o performață mai bună.

7.2 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 (TODO adaugă link la flow-uri impoprtante).

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



7.3 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 o definiție de tip care este defapt un simplu 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: **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 valoarii 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 sau 2 * var + 1 (dacă semnul literalului este negativ).
  Așadar se ajunge ca int-ul stocat în **Lit** să aibă encodat pe cel mai puțin semnificant bit valoare semnului, iar pe restul de biți valoare **Var** de la care provine. Exemplu: Daca 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).

7.4 Metode remarcabile

În această secțiune sunt prezentate câteva dintre metodele importante folosite pentru a rezolva problema satisfiabilității în MiniSat.

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. În repository-ul de GitHub al echipei noastre se poate vedea diagrama de activitate asociată acestei metode (TODO insert link).

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.

```
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. Sursă TODO. De asemenea, diagrama de activitate poate fi găsită în proiectul echipei noastre de GitHub TODO link

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 (TODO source). Această metoda folosește two-literal watch scheme pentru a eficientiza propagarea unitară. Diagramă de activitate poate fi găsită pe GitHub-ul asociat proiectului nostru (TODO add link).

8 Propriul nostru repository si cum am impartit task-urile

9 Concluzii

În cadrul acestui proiect, am investigat și implementat MiniSat pentru a rezolva problema SAT, utilizând un benchmark din competiția SAT 2024. Proiectul a avut ca scop înțelegerea algoritmilor de rezolvare a problemei SAT, identificarea strategiilor de enumerare folosite de MiniSat și analiza performanței acestuia în diferite scenarii.

Am identificat principalele algoritmi de rezolvare a problemei SAT, precum DPLL și CDCL (Conflict-Driven Clause Learning), care sunt implementați eficient în MiniSat. 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 în ceea ce privește gestionarea memoriei și al performanței la scară mare, precum și în implementarea unor tehnici avansate de backtracking și reducerea conflictelor.

Un aspect important al proiectului a fost experimentarea cu benchmark-uri SAT din competiția SAT 2024. Rezultatele obținute au arătat performanțe bune ale MiniSat în rezolvarea unor instanțe de dimensiuni variabile, dar și provocări legate de timpul de execuție și eficiența în cazul unor formule extrem de mari sau complexe. Am realizat grafice și analize detaliate pentru a vizualiza performanțele în funcție de dimensiunea inputului și complexitatea formulelor SAT.

Provocările întâmpinate au inclus înțelegerea complexității codului sursă MiniSat și integrarea acestuia într-un mediu de testare eficient. De asemenea, documentația limitată 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 ceea ce privește dezvoltarea și îmbunătățirea MiniSat, am sugerat posibile optimizări ale algoritmilor de rezolvare, incluzând implementarea unor tehnici mai avansate de reducere a căutărilor inutile în cadrul algoritmului DPLL și îmbunătățirea strategiilor de conflicte din cadrul CDCL. De asemenea, am propus modalități prin care MiniSat poate fi îmbunătățit pentru a gestiona mai eficient formulele SAT foarte mari și complexe.

Aceste analize și sugestii pot contribui la dezvoltarea unui MiniSat mai performant, oferind un punct de plecare pentru cercetări viitoare. În ansamblu,

proiectul ne-a permis să aprofundăm atât cunoștințele teoretice legate de SAT, cât și abilitățile practice de programare și analiză de performanță a software-ului.

10 Bibliografie

1.GitHub Repository - MiniSat
Source code and documentation for MiniSat.
https://github.com/niklasso/minisathttps://github.com/niklasso/minisat

2.Marijn Heule, Oliver Kullmann, Victor Marek
"Solvers for SAT: Recent Advances and Applications."
Handbook of Satisfiability, 2021. ISBN: 978-1-58603-929-5.

3.Armin Biere, Marijn J.H. Heule, Hans van Maaren, Toby Walsh (Editors)
"Handbook of Satisfiability."
Frontiers in Artificial Intelligence and Applications, Vol. 185, IOS Press, 2009. ISBN: 978-1-58603-929-5.

4.Davis, Martin; Logemann, George; Loveland, Donald
"A Machine Program for Theorem-Proving."
Communications of the ACM, Volume 5, Issue 7, pp. 394-397, 1962.

5.Zhang, Lintao; Malik, Sharad
"The Quest for Efficient Boolean Satisfiability Solvers."
CAV 2002: Computer Aided Verification, pp.17-36. ISBN: 978-3-54045657-4.

6.SAT Competition
Annual SAT Competitions and benchmark problems.
https://www.satcompetition.org/https://www.satcompetition.org/