

Benchmark SAT Competition

Berindeie Adrian, Buzuriu Roxana, Constantinescu Mihai, și Felea Irina

Universitatea de Vest din Timișoara
<https://www.uvt.ro>

Abstract. 150-250 cuvinte

Keywords: SAT, satisfiabilitate, CDCL, DPLL, Solvers

1 Introducere

- metodele utilizate la competițiile SAT, metode actuale, ce s-a adăugat la ele
- Utilizarea SAT și ce este SAT
- definirea unui benchmark
- de ce vrem optimizare? s-a încercat deja? rezultate?
- împărțirea pe familii, ce reprezintă fiecare <https://benchmark-database.de/>
- rationamentul pentru care facem optimizare MiniSat și ideile propuse
- motivarea alegerii familie/familii
- related works? capitol separat?

Idei propuse pentru rulare și optimizare:

- Optimizare:
 - Optimizarea ordonării variabilelor: MiniSat folosește o metodă euristică pentru a decide ordinea variabilelor în timpul căutării. Astfel, o îmbunătățire ar fi să prioritizăm variabilele mai importante, adică cele care apar mai frecvent în conflictele recente.
 - Ștergerea clauzelor inutile. Solverul poate acumula la un moment dat mai multe clauze de conflict, care la un moment dat pot deveni redundante. O ștergere periodică a acestora poate optimiza procesul. Aceasta implică testarea unor condiții pentru a decide păstrarea sau ștergerea clauzelor și o gestionare a memoriei.
 - Optimizarea memoriei alocate și reducerea dimensiunii alocate pentru clauze neutilizate.

Problemele de satisfiabilitate booleană (SAT) sunt o componentă esențială a logicii computaționale, având aplicații extinse în domenii precum verificarea formală a sistemelor software și hardware, planificarea automată și optimizarea. Solverele de tip SAT, cum ar fi MiniSat, sunt utilizate pentru a determina dacă o formulă booleană poate fi satisfăcută. Se poate confirma că este satisfăcută atunci când există o soluție, adică dacă s-a găsit o atribuire de valori de adevăr pentru variabile care să facă întreaga formulă adevărată [1].

MiniSat, este un solver SAT care se bazează pe algoritmul CDCL (Conflict-Driven Clause Learning), fiind bazat pe învățarea de conflicte și reordonarea dinamică a variabilelor. Acesta s-a remarcat prin eficiență și simplitate, astfel putem spune că a devenit un reper în dezvoltarea de soluții avansate pentru SAT. Algoritmul CDCL folosit în MiniSat a adus îmbunătățiri semnificative față de DPLL (Davis-Putnam-Logemann-Loveland), introducând mecanisme precum învățarea din conflicte, generarea clauzelor de conflict, deciziile euristice pentru selectarea variabilelor și reluarea adaptivă a căutării prin backtracking cronologic. Aceste îmbunătățiri au condus la o creștere semnificativă a eficienței în rezolvarea instanțelor complexe de SAT. De-a lungul timpului, solvele CDCL au fost optimizate prin integrarea unor tehnici suplimentare, cum ar fi reordonarea dinamică a clauzelor și metodele de propagare folosind doi literalii urmăriti [2].

Competițiile SAT oferă o platformă de testare riguroasă pentru evaluarea și compararea noilor soluții în rezolvarea problemelor SAT. Aceste evenimente adună dezvoltatori pentru a prezenta și testa cele mai recente soluții, contribuind astfel la progresul constant al acestui domeniu și evidențiind rolul esențial al implementărilor optimizate și a algoritmilor eficienți [3]. MiniSat și variațiile sale au fost supuse unor îmbunătățiri constante în cadrul acestor competiții, care includ strategii avansate de propagare și gestionare a clauzelor învățate. Aceste tehnici s-au dovedit esențiale pentru reducerea timpului de rulare și creșterea performanței generale a solverului. În acest context, proiectul nostru își propune să implementeze și să evalueze modificări specifice în MiniSat, axate pe optimizarea algoritmilor, cu scopul de a îmbunătăți eficiența în soluționarea problemelor.

Modificările propuse vor fi testate pe un set de benchmark-uri din una sau mai multe familii diferite, utilizate în competiția SAT 2024, permițând o analiză detaliată a impactului asupra performanței și o comparație cu versiunea inițială a MiniSat. Pentru un studiu cât mai elaboros, vom folosi 3 sisteme de operare diferite, Windows, Linux și MacOS. Prin această cercetare vom evidenția atât dificultățile întâmpinate, cât și posibilele direcții de optimizare viitoare pentru solvele de tip SAT.

2 Descrierea Problemei

-Detalii tehnice despre cum abordez problema găsită și variantele de optimizare, soluționarea problemei explicată tehnic.

-Încapsularea contribuției echipei

3 MiniSat

- intelegere cod
- descriere implemenatare
- detectare implementare DPLL
- detectare implementare CDCL
- detectare implementare euristici

- implementare DLIS
- diagrame
- documentarea codului

3.1 Instalare MiniSat

Instalare MiniSat pe Linux Pentru a instala MiniSat pe sistemul de operare Linux, este nevoie de clonarea proiectului de repository-ul de GitHub, folosind comanda prezentată în Secvența de cod 1.1. După clonarea proiectului, din directoriul MiniSat-ului, se rulează comanda prezentată în Secvența de cod 1.2 pentru a rula codul într-un mod mai permisiv. Această comandă setează un flag specific pentru compilatorul de C++, transformând erorile mai puțin critice în avertismente și astfel ajutând la compilarea codului. Comanda prezentată în Secvența de cod 1.3 este o comandă necesară pentru cazul în care sistemul nu are deja comanda "make", comandă necesară pentru a crea un executabil din codul C++. După rularea comenzilor se navighează în locația executabilului prin comanda prezentată în Secvența de cod 1.4, iar din acest directoriu putem folosi programul MiniSat.

Secvența de cod 1.1: Clonare proiect

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

Secvența de cod 1.2: Compilare permisivă

```
cd minisat/
make CXXFLAGS="-fpermissive"
```

Secvența de cod 1.3: Instalarea comenzii make

```
sudo apt install make
```

Secvența de cod 1.4: Navigarea către directoriul executabilului

```
cd build/release/bin
```

Instalare MiniSat pe Windows Instalarea MiniSat pe sistemul de operare Windows este precedată de instalarea unor programe și configurarea căilor de acces. Pentru început, este necesară instalarea subsistemului Unix pentru Windows, Cygwin¹, rularea fișierului `setup.exe`, în urma căruia Cygwin va crea directoarele în directoriul `./cygwin/`. Apoi este necesară descărcarea software-ului CMake² și extragerea fișierelor într-un director cu denumirea `C:\CMake`. După realizarea instalărilor necesare, se configurează căile de acces în PATH. Pentru a configura căile de acces este necesară deschiderea setărilor sistemului, iar în "**Environment Variables**", la variabila `Path` sunt adăugate căi de acces către

¹ Cygwin <https://www.cygwin.com> [accesat în 02.11.2024]

² CMake <https://cmake.org/download/> [accesat în 03.11.2024]

MinGW\bin și CMake\bin. Pentru a instala Minisat, acesta trebuie descărcat de pe pagina oficială a programului³. Se deschide subsistemul Cygwin și se navighează în directoriul Minisat. Din acest directoriu se rulează comanda prezentată în Secvența de cod 1.5, comandă care specifică generatorul de fișiere Makefiles pentru Minimalist GNU for Windows (MinGW) și creează un fișier Makefile compatibil cu mingw32-make. În Secvența de cod 1.6 este prezentată comanda care compilează fișierele generate anterior și creează executabilul minisat.exe.

Secvența de cod 1.5: Generare fișiere Makefile

```
cmake -G "MinGW Makefiles" ...
```

Secvența de cod 1.6: Generare fișiere Makefile

```
cmake --build
```

3.2 Utilizare MiniSat

```
minisat input.txt sau minisat input.cnf
```

3.3 Detalii de implementare

- Unde a fost rulat benchmark-ul: Rularea benchmarkului pe 2/3 sisteme de operare diferite (Windows, MacOS și Linux)
- Ce tip de date s-au folosit, împărțirea pe familii
- Explicam benchmarkul/dataset, metrice pentru performanța, eficiența deja existentă

Structuri de date folosite În această secțiune vom prezenta principalele structuri de date utilizate în cadrul aplicației MiniSat. Definirea acestor structuri de date a fost realizată în cadrul fișierului `minisat/mtl/SolverTypes.h`.

- `struct Lit` - Reprezentarea unui literal logic utilizând o variabilă de tip `int`, astfel făcând posibilă și stocarea semnului literalului.
- `class lbool` - Reprezentarea unei valori logice. Această clasă este creată pentru a optimiza comparația între o variabilă și o constantă, precum și pentru a se asigura că `gcc` realizează suficientă propagare a constantelor. Aceste aspecte sunt precizate pe linia 89. Această clasă mai conține în plus suprascrierea unor operatori pentru a ușura lucrul cu acest tip de variabilă în contextul rezolvării unei probleme de satisfacibilitate în logica computațională, cum ar fi: `==`, `!=`, `^`, `&&`, `||`. Aceste variabile au fost create cu scopul de a reprezenta 3 valori, chiar dacă acestea sunt definite de tipul `uint8_t: true, false, undefined`. Astfel, pe liniile 123–131, putem observa definirea următoarelor constante:
 - `l_True (lbool((uint8_t)0))`

³ The MiniSat Page <http://minisat.se/MiniSat.html> [accesat în 05.11.2024]

- `l_False (lbool((uint8_t)1))`
 - `l_Undef (lbool((uint8_t)2))`
- `class Clause` - Reprezentarea unei clauze logice. Această clasă reține informațiile de bază precum lista de literalii și lungimea acesteia dar și alte informații mai avansate precum dacă această clauză a fost învățată deja (proprietatea `learned`). Totuși este precizat că nu se poate folosi în mod direct constructorul acestei clase, ci se va folosi clasa prieten `ClauseAllocator` care se ocupă de alocarea corectă și eficientă a memoriei pentru această structură de date.

Pașii parcurși de algoritm Analiza pașilor principali executați de programul MiniSat începe cu fișierul `minisat/core/Main.cc`. La începutul funcției `main()` putem observa cum programul analizează argumentele primite din linia de comandă și extrage calea către fișierul de intrare (dacă există) și opțiunile selectate cum ar fi: verbozitatea, limita de timp de execuție pe procesor permisă (în secunde), limita de memorie utilizată și alegerea de a valida sau nu antetul DIMACS în timpul parcurgerii (liniile 63-68). După acest pas programul inițializează solver-ul S, extrage timpul inițial, limitează resursele bazat pe argumentele primite și citește datele de intrare de la tastatură sau dintr-un fișier de intrare după caz. Pe linia 95 programul începe să parcurgă datele de intrare și să adauge clauze în interiorul lui S în cadrul funcției `parse_DIMACS`. Se încearcă simplificarea clauzelor. Dacă acesta eșuează, se declară că formula este nesatisfiabilă iar execuția este oprită.

Rezolvarea propriu-zisă a problemei începe pe linia 124 prin apelarea metodei `S.solveLimited(dummy)`. Implementarea acestei metode se poate găsi în fișierul `minisat/core/Solver.cc`. Această metodă începe prin resetarea atributelor `model` și `conflict` urmată de verificarea dacă solver-ul este într-o stare validă. Căutarea unei soluții se realizează în interiorul buclei dintre liniile 865-870 care se execută atât timp cât valoarea variabilei `status` este `l_Undef`. Principala acțiune de căutare a unei soluții se realizează în cadrul metodei `search()` care va fi detaliată în subsecțiunea 3.3. După găsirea unui status acesta este procesat și afișat utilizatorului alături de restul statisticilor adunate în timpul execuției.

TODO: diagrama de secvențe și activitate
 TODO: detaliere pt funcția `luby()`

Detalierea funcțiilor principale

- `simplify()` - În implementarea curentă, această funcție doar elimină clauzele care sunt deja satisfăcute, reducând dimensiunea problemei, ajutând astfel la accelerarea procesului de căutare a unei soluții. Aceasta parcurge clauzele care sunt satisfăcute cu valorile deja atribuite unor literalii, până în acel moment al execuției.

- **pickBranchLit()** - Metoda selectează următoarea decizie, atât literalul cât și polaritatea acestuia. Se alege o valoare aleatoare întreaga între 0 și 1. Dacă acesta este mai mică decât variabila **random_var_freq** și heap-ul în care se rețin variabilele neasignate nu este gol, atunci se alege o variabilă la întâmplare din heap.
Dacă această condiție nu este adevărată cât timp variabila **next** este egală cu **var_Undef** sau valoarea pentru **next** este diferită de **l_Undef** sau **decision[next]** încă este fals se verifică heap-ul. Dacă acesta este gol, lui **next** i se asignează valoarea **var_Undef** și se iese din buclă. Dacă nu, lui **next** i se dă valoarea minimă din heap.
Pentru a se alege polaritatea, prima dată se verifică dacă **next** este **var_Undef**. În acest caz se returnează **lit_Undef**. Dacă nu, dacă există o polaritate aleasă de utilizator, acesta este aignat. În caz contrar, dacă variabila **rnd_pol** este asignată se alege o polaritate aleatoare. Dacă nici una dintre condițiile de mai sus nu sunt împlinite se asignează o polaritate predefinită **polarity[next]**.
- **propagate()** - Metoda este responsabilă pentru propagarea unitate, atribuirea de valori bazată pe clauzele învățate pentru a reduce domeniul de căutare a unui model. În variabila **Lit p** se reține literalul curent de la care se pleacă cu propagare. Acesta este extras din coada **trail** care este parcursă cu ajutorul variabilei **qhead**. Pentru fiecare literal, funcția **propagate()** parcurge fiecare clauză. Dacă aceasta este deja satisfăcută, atunci se trece peste ea. Dacă nu este, se caută un literal care nu are o valoare atribuită pentru a încerca să satisfacă clauza. Dacă nu se găsește nici un literal care să satisfacă clauza se aplică din nou propagarea unitate (linia 554). În cadrul acestei funcții se mai execută și verificarea unor noi conflicte, iar dacă unul este găsit se va asigna în variabila **confl** referința către acest conflict (liniile 545-552).
- **analyze(CRef confl, vec<Lit>& out_learnt, int& out_btlevel)** - Analizează conflictul transmis ca parametru pentru a produce clauza care a produs acest conflict. Clauza găsită este adăugată în mulțimea de clauze învățate pentru a optimiza procesul de căutare a unui model, aplicându-se propagarea unitate peste această mulțime pentru a asigura valori unor literal bazate pe implicațiile rezultate din aceste clauze, ajutând la evitarea conflictelor întalinite deja în procesul căutării.
Metoda începe prin inițializarea variabilelor **pathC** și **p** (literalul curent analizat). În cadrul buclei **do-while** se analizează clauza de conflict curentă, iar dacă aceasta este deja învățată, i se crește activitatea. Pentru fiecare literal al clauzei dacă nu mai fost văzut până în acel moment și se găsește pe un nivel de decizie mai mare decât 0, atunci este marcat ca văzut. Dacă nivelul pe care se găsește acest literal este mai mare sau egal cu nivelul de decizie, se incrementează variabila **pathC**, iar dacă nu literalul este adăugat în variabila **out_learned**. După acest pas se caută următoarea clauză care va trebui analizată pe baza următorului literal din coada **trail**. Pentru acesta se va găsi clauza care l-a produs și va fi asignată către **conf**, iar apoi se va decrementa variabila **pathC**. Această buclă se oprește atunci când ajunge

variabila `pathC` 0. După terminarea buclei, se realizează simplificarea clauzei, urmată de găsirea nivelului corect de backtracking.

- `search(int nof_conflicts)` - Metoda începe prin declararea și inițializarea variabilelor pentru nivelul de backtrack, variabilei `conflictC` și clauzelor învățate, apoi se începe un ciclu infinit în care se încearcă găsirea unui model. Ciclul începe prin căutarea unei clauze de conflict cu ajutorul metodei `propagate()`. În cazul găsirii unui conflict se verifică dacă nivelul de decizie este 0, iar dacă această condiție este adevărată se returnează `l_False`, deducându-se că formula este satisfiabilă. Dacă acesta este diferit de 0, se resetează lista de clauze învățate și folosind metoda `analyze()` pentru a o repopula cu noile clauze învățate care iau în considerare conflictul găsit. După analiză se revine la ultimul nivelul găsit aplicând metoda `cancelUntil()`. Dacă nu este găsit nici un conflict verificăm dacă ne aflăm în limitele setate, adică limita de conflicte, dacă acesta există și limitele de resurse (timp, memorie). Dacă una oricare dintre limită este depășită se va return `l_Undef`, indicând că nu s-a putut studia complet formula în limitările setate. Dacă se trece de această verificare se încearcă simplificarea clauzelor. Dacă ne aflăm pe nivelul 0 și nu se poate realiza nici o simplificare atunci se va returna `l_False`. Următoarea acțiune care se realizează este reducerea clauzelor învățate dacă numărul acestora depășește limita setată. După realizarea verificărilor menționate mai sus, se realizează o nouă decizie. Inițial se încearcă asignarea unei valori indicate de utilizator asupra unui literal. Dacă aceste presupuneri nu există se încearcă asignarea folosind metoda `pickBranchLit()`. Dacă și după această încercare de asignare nu a putut fi aleasă nici o valoare înseamnă că am găsit un model, deci vom returna `l_True`. Dacă s-a ales o valoare pentru un literal, incrementăm nivelul de decizie și adăugăm noul literal în coadă.

Identificarea algoritmilor de enumerare, DPLL sau CDCL

4 Rezultatele experimentale

Pentru familia *scheduling*, au fost rulate fișierele specifice acestei familii.

Pentru rularea testelor, a fost setat un timp CPU limită de *1728 de secunde*. Testele au fost rulate pe diferite configurații:

1. Laptop **Macbook Air**:
 - Procesor **Apple M1**
 - Memorie RAM **8GB**
 - Sistem de operare **MacOS Sonoma 14.4.1**
2. Laptop **Lenovo V310-15IKB**:
 - Procesor **Intel Core i7-7500U**
 - Memorie RAM **8GB**
 - Sistem de operare **Windows 10 Home**

- grafic ce arata relatia dintre dimensiune si timpul de executie.

Table 1: Rezultatele testelor experimentale.

Scheduling								
			Configuratie 1			Configuratie 2		
Test	Variabile	Clauze	Timp CPU	Memorie	Răspuns	Timp CPU	Memorie	Răspuns
1	14756	141683	1109.72 s	200.94 MB	UNSAT	1826.18 s	226.49 MB	INDET
2	38781	362841	1659.87 s	381.07 MB	INDET	1873.77 s	1082.81 MB	INDET
3	14400	63874	17.3344 s	33.80 MB	UNSAT	38.1849 s	33.24 MB	UNSAT
4	2973	15516	1648.76 s	146.58 MB	INDET	1880.09 s	68.54 MB	INDET
5	221	1084	0.009421 s	4.85 MB	UNSAT	0.006469 s	5.97 MB	UNSAT
6	1101	5036	0.422917 s	9.49 MB	UNSAT	0.678463 s	6.23 MB	UNSAT
7	2010	11953	1465.29 s	141.27 MB	INDET	1818.18 s	53.30 MB	INDET
8	17850	149132	946.04 s	195.01 MB	UNSAT	1881 s	202.65 MB	INDET
9	14560	48075	1543.39 s	179.64 MB	INDET	1843.72 s	126.55 MB	INDET
10	21288	87245	694.386 s	161.01 MB	SAT	1885.32 s	134.71 MB	INDET
11	49869	264805	1686.52 s	463.18 MB	INDET	1852.69 s	1035.09 MB	INDET
12	37005	187726	1677.85 s	806.57 MB	INDET	1789.65 s	475.77 MB	INDET
13	93713	10295409	58.5334 s	1996.39 MB	SAT	146.975 s	1833.28 MB	SAT
14	32273	361039	1330.87 s	798.20 MB	INDET	1784.44 s	1630.84 MB	INDET
15	28830	320272	1621.04 s	1697.41 MB	INDET	1856.52 s	1659.16 MB	INDET
16	14128	46571	1717.94 s	161.27 MB	INDET	1784.44 s	98.07 MB	INDET
17	14174	66704	13.9012 s	43.11 MB	SAT	33.8056 s	29.14 MB	SAT
18	22365	83896	1715.57 s	298.08 MB	INDET	1795.28 s	252.23 MB	INDET
19	1015	21642	1706.28 s	327.14 MB	INDET	1829.38 s	271.76 MB	INDET
20	249117	764929	1314.89 s	2797.59 MB	INDET	1804.97 s	1710.56 MB	INDET
21	282525	1743751	449.656 s	2999.46 MB	INDET	1830.74 s	4562.01 MB	INDET
22	1516	6635	1122.38 s	121.11 MB	UNSAT	1811.21 s	40.80 MB	INDET
23	26286	319289	1691.84 s	823.04 MB	INDET	1834.96 s	1640.22 MB	INDET
24	14945	66518	14.4803 s	38.26 MB	SAT	34.4257 s	33.75 MB	SAT
25	6397	60575	246.293 s	96.22 MB	UNSAT	569.099 s	96.45 MB	UNSAT
26	10198	61395	1673.77 s	198.71 MB	INDET	1814.36 s	153.00 MB	INDET
27	37028	187771	1705.58 s	1323.73 MB	INDET	1841.8 s	716.64 MB	INDET
28	656	11934	0.011122 s	5.35 MB	SAT	0.007044 s	6.55 MB	SAT
29	3890	14187	0.068435 s	6.82 MB	SAT	0.064183 s	7.41 MB	SAT
30	29478	125712	0.106474 s	12.52 MB	SAT	0.170763 s	16.52 MB	SAT
31	145943	625454	20.1524 s	181.53 MB	SAT	73.6448 s	182.39 MB	SAT
32	19502	82348	1709.56 s	191.89 MB	INDET	1815.89 s	153.42 MB	INDET
33	53844	270141	1708.94 s	1572.19 MB	INDET	1832.5 s	1247.98 MB	INDET
34	1275631	1974386	1707.63 s	1204.72 MB	INDET	1821.6 s	686.98 MB	INDET
35	24101	254904	1684.74 s	829.19 MB	INDET	1820.72 s	1059.04 MB	INDET
36	2508	87544	1702.79 s	834.93 MB	INDET	1820.79 s	1702.12 MB	INDET
37	18913	85586	1705.65 s	183.61 MB	INDET	1837.52 s	149.83 MB	INDET
38	8048	54850	1707.95 s	281.02 MB	INDET	1837.36 s	180.82 MB	INDET
39	17850	154655	223.214 s	131.86 MB	UNSAT	544.334 s	158.11 MB	UNSAT
40	14112	166497	1381.68 s	259.28 MB	INDET	1829.94 s	229.40 MB	INDET
41	58907	193418	1675.12 s	400.32 MB	INDET	1851.66 s	690.25 MB	INDET
42	21015	69505	1138.31 s	152.57 MB	SAT	1841.53 s	156.52 MB	INDET
43	2351	13401	1415.36 s	180.14 MB	INDET	1842.87 s	123.18 MB	INDET
44	145862	624664	379.05 s	194.04 MB	SAT	1235.91 s	1593.70 MB	SAT
45	46759	250518	86.6242 s	99.90 MB	SAT	265.449 s	210.96 MB	SAT
46	14751	63112	1707.47 s	174.44 MB	INDET	1842.64 s	119.05 MB	INDET
47	1654	8801	1678.18 s	156.80 MB	INDET	1869.49 s	43.13 MB	INDET
48	13043	51999	1380.45 s	148.19 MB	INDET	1841.23 s	97.95 MB	INDET
49	17854	226553	1694.31 s	800.26 MB	INDET	1871.49 s	697.38 MB	INDET
50	31113	127844	1638.65 s	1227.83 MB	INDET	1851.76 s	380.50 MB	INDET


```

07c50b89d97a55b6e1629999a45a2ba-ctl.4201.555-unsat_pre.cnf
07e6413459f92b613498a719125b6239-j3037_10_mdd_bm1.cnf
09d7add3bf3b75c5d1023a92e752989a-Break_unsat_04_03.xml.cnf
0ac6aaf6d6a0a4ac3a0c6a65d0108f1-pcmac-scheduling-m13-2011-12813-UNSAT.sanitized.cnf
0d3fca9b0b08e821058a3ea4284ea19045_10_gmto_b.cnf
11cc52ecdcdfb10a47e0a889b4867c1b-Break_triple_20_36.xml.cnf
1afa2d7a3d817c3149da432eece66da8-worker_550_550_550_0.3.cnf
207b28c3cfc55b14259222c0f94862-pcmac-scheduling-m37-28831-32436-SAT.sanitized.cnf
22a4d412a811b720ba070106a86c-j3037_10_rggt_b.cnf
29e61b1317804f897e4237613d42bb-worker_30_60_25_0.9.cnf
2a15a30186afdad41a49c5c5366d01be-Timetable_C_392_E_62_Cl_26_S_28.cnf
31e788a12ddc8b43ec20a77d53abaa23-pcmac-scheduling-m40-26287-324155-SAT.sanitized.cnf
32b5a69749f424161eaf8c59f654750-pcmac-scheduling-m25-6398-62377-UNSAT.sanitized.cnf
35a3875d75de839a3ac326c5425c1b5-Break_18_32.xml.cnf
3988a60c6e93167763c6fd2a347d5859-Break_08_24.xml.cnf
43e492bccdf57029b758897b17d704f-pb_300_09_lb_07.cnf
44c25de7963c45e92a2407ad839f68b-Break_20_72.xml.cnf
5432e4d44489e9b66597f0b1752a30-pcmac-scheduling-m24-24102-255206-SAT.sanitized.cnf
579ebc13372c466c20e98792721e1f-j3037_9_rggt_b.cnf
5952b9f1b773dabb531c60a3ac85d26f-ctl_4291_567_9-unsat.cnf
5a06bba2fa30a435a76e2ac80896f4-ITC2021_Middle_1.xml.cnf
5b5dfab02c6c17a509b3bc7d7cc4b7-pcmac-scheduling-m15-2352-13561-SAT.sanitized.cnf
5ea72bdc0cc86bd1a924029f7b81a6c5-atco_anc1_opt1_10_15.cnf
62c8b79f95ecf5618c9d9a00311326c-pcmac-scheduling-m13-1655-9604-UNSAT.sanitized.cnf
6fc528fc3d0fd5a2c50992feb8b0357-pcmac-scheduling-m24-17855-226744-SAT.sanitized.cnf
74145bb035650f5c982d7ec6967045-pcmac-scheduling-m43-38742-385402-SAT.sanitized.cnf
71c7da531539f3ba41e0e916913086-pcmac-scheduling-m19-2974-16501-UNSAT.sanitized.cnf
777109dce621ef361a72b3e8ce9a962-Break_unsat_06_07.xml.cnf
7fc438c38e5b7a70e38778ec145cfe0-ctl_4291_567_2-unsat-sc2013.cnf
83a33b0c34e6d6c5656101ca8338b3c-j3037_1_mdd_b.cnf
94dd280b1562ae7dae44b303b8fed233-Break_unsat_18_31.xml.cnf
9e85aad5707405fb349c6c5548040c4-pcmac-scheduling-m35-32274-371389-SAT.sanitized.cnf
a08e6296d00480e9ccadd79fa88904-j045_4_gmto_b.cnf
b087e195e5b7b19163a701a0ba6d1-Break_unsat_14_23.xml.cnf
bd12a2b66110451b050bd5d2943b1854-ITC2021_Early_10.xml.cnf
bd8bc25be2b36c64b38459c17e815814-pcmac-scheduling-m11-1517-6802-UNSAT.sanitized.cnf
be64114784a3c879686dda807cdc607-j3037_10_mdd_b.cnf
bfadd3471516e950b2a3e8f12ba03-pcmac-scheduling-m19-10199-62102-UNSAT.sanitized.cnf
c801a020a6c8b3c287ea495203b114-worker_20_40_20_0.95.cnf
ca14adcb9296a7b31d7815c2ad16d0f1-ITC2021_Early_3.xml.cnf
cc30e4eedbdcf9c3bc110894d08781246-j3037_9_mdd_bm1.cnf
d02fa1e84cbe799c792ba23708f68-exam_flat_0_04_2018_3.cnf
d7527176904c586934c25c11488e99-worker_40_80_40_0.9.cnf
da929c2aefde5e58878ad87c4323e581-pcmac-scheduling-m12-8049-55035-SAT.sanitized.cnf
dafc03ea784fee849fbcba7b64230558-pcmac-scheduling-m30-14113-167638-UNSAT.sanitized.cnf
e481a97a93d6b430f39024d19c2667-j3037_1_gmto_b.cnf
ed4fae84ee8330e468339f62abba5e-pb_300_05_lb_17.cnf
f7e855b18105170c0718086d3f5b5923-j045_10_rggt_b.cnf
fae08ac5b6d7eeb4f56f1cfac6c51703-j045_4_mdd_bm1.cnf
fb45d41807bcbbbe689dc282c1310796-Break_unsat_16_27.xml

```

Fig. 1: Fișiere din familia *scheduling*.

5 Provocări

-instalare pe Windows
 -instalare pe Linux
 -instalare Mac - doar exe

6 Concluzie

Bibliografie

1. Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.
2. Tobias Fuchs, Jakob Bach, and Markus Iser. Active learning for sat solver benchmarking. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 407–425, Cham, 2023. Springer Nature Switzerland.
3. Marijn JH Heule, Markus Iser, Matti Jarvisalo, and Martin Suda. Proceedings of sat competition 2024: Solver, benchmark and proof checker descriptions. 2024.

7 Împărțirea responsabilităților în echipă

1. Berindeie Adrian

- script pentru descărcare fișiere pe familii
- script pentru pregătirea și rularea benchmark cu diferite configurații de fișiere
- îmbunătățirea minisat-ului prin rularea pe fire de execuție
- prelucrarea și rulare benchmark
- analiza date obtinunte

2. Buzuriu Roxana

- studiu lucrari științifice
- împărțirea pe familii
- structurare referat
- introducere, instalare pe windows

3. Constantinescu Mihai

- studiu cod - analiza structurii generale a codului, structurilor de date folosite, functiilor principale folosite in program

4. Felea Irina

- studiu cod
- împărțirea pe familii
- studiu familii
- rulare benchmark