УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Давид С. Шћепановић

ИМПЛЕМЕНТАЦИЈА И ЕВАЛУАЦИЈА ТЕХНИКЕ ЧУВАЊА ПАРЦИЈАЛНЕ ВАЛУАЦИЈЕ ПРИ ПОВРАТНИМ СКОКОВИМА У ОКВИРУ MINISAT SAT РЕШАВАЧА

мастер рад

Београд, 2021.

Ментор:

др Милан Банковић, доцент
Универзитет у Београду, Математички факултет

Чланови комисије:

др Предраг Јаничић, редовни професор
Универзитет у Београду, Математички факултет

др Филип Марић, ванредни професор
Универзитет у Београду, Математички факултет

Датум одбране: _____

Наслов мастер рада: Имплементација и евалуација технике чувања парцијалне валуације при повратним скоковима у оквиру MiniSat SAT решавача

Резиме: У овом раду разматра се једна од актуелних техника за унапређење перформанси SAT решавача, која се заснива на чувању парцијалне валуације при повратним скоковима. Поред њеног детаљног описа и анализе, представљен је и предлог њеног унапређења, што је уједно и највећи допринос рада. Он је поткрепљен доказом коректности, као и темељном евалуацијом, како унапређене тако и основне технике, чији резултати указују да предложено унапређење позитивно утиче на ефикасност рада решавача. Имплементација и евалуација је извршена у оквиру MiniSat SAT решавача, а критични детаљи имплементације се такође разматрају у раду.

Кључне речи: SAT проблем, SAT решавачи, MiniSat SAT решавач, CDCL процедура, техника чувања валуације

Садржај

Глава 1

Увод

У рачунарству, SAT проблем (енгл. *Boolean satsifiabilty problem*) познат је као први проблем за који је доказана NP-комплетност [3]. SAT проблем представља проблем испитивања задовољивости исказних формула, прецизније, утврђивања постојања вредности исказних променљивих за које је дата формула тачна. Притом, подразумевамо да су формуле које разматрамо у *конјуктивној нормалној форми* (КНФ). На пример, формула $(x_1 \vee \neg x_3) \wedge (\neg x_1) \wedge (x_3 \vee x_2)$ је исказна формула у КНФ форми и задовољена је акко су променљиве x_1 и x_3 нетачне, а променљлива x_2 тачна.

Осим теоријског, SAT проблем има и немерљив практични значај, с обзиром на велики број проблема из науке и индустрије који се могу решавати свођењем на SAT. Неке од индустријских области у којима се SAT проблем успешно примењује су верификација софтвера и хардвера, проблеми планирања и распоређивања, организација међузависности софтверских пакета, проблеми комбинаторног дизајна, биоинформатика, криптографија итд. [2].

Алати који имплементирају процедуре за решавање SAT проблема зову се *SAT решавачи*. Њихова широка примена, као и све захтевнији проблеми који се срећу у индустрији, стварају све већу потребу за унапређењем њихових перформанси. Ипак велики корак у овом смеру је већ постигнут. Наиме, од њиховог настанка почетком 21. века, CDCL SAT решавачи (енгл. *Conflict-Driven Clause Learning*) [7] су се показали изузетно успешним у решавању широког корпуса SAT проблема, пре свега у домену индустријских примена. Због тога данас, CDCL алгоритам, унапређена верзија DPLL алгоритма (енгл. *David-Putnam-Loveland-Logemann*) [4], представља темељ сваког ефикасног приступа SAT проблему.

MiniSat SAT решавач[1], један од најпознатијих и најчешће коришћених CDCL SAT решавача, у овом раду служи као основа за имплементацију и евалуацију једне од актуелних техника за побољшање перформанси SAT решавача, *технике чувања парцијалне валуације при повратним скоковима*. Овај приступ првобитно је разматран на SAT конференцији 2020. године [5], а ослања се на чињеницу да SAT решавачи често након повратног скока (енгл. *backjumping*) настављају претрагу на сличан начин као и пре повратног скока, понављајући притом временски захтевне процедуре попут пропагације литерала, градећи исту или сличну валуацију као и пре повратног скока. Чувањем дела валуације који треба поништити пре повратног скока, ова техника омогућава да се обим дуплираног посла смањи, у циљу побољшања перформанси SAT решавача. Очекивано, заједно са новим приступом, повећава се комплексност саме процедуре и откривају се нове потешкоће са којима се упознајемо у овом раду. Зарад читљивости, уместо пуног имена ове технике, у даљем тексту користићемо скраћени назив - *техника чувања валуације*.

Главни допринос овог рада је детаљни опис и анализа технике чувања валуације, као и предлог неких алгоритамских и имплементационих унапређења ове технике. Додатни допринос је детаљна евалуација имплементације, како основне, тако и унапређене технике у оквиру MiniSat SAT решавача. Додатно, како се MiniSat често користи као језгро других SAT и SMT решавача, имплементација и приступ у овом раду може бити од помоћи при интеграцији ове технике како у поменуте решаваче, тако и у решаваче сличних спецификација и ограничења.

Наставак овог рада је организован на следећи начин. У глави 2 уведени су основни појмови неопходни за разумевање и опис поменутих техника. Специјално, у њој је представљена CDCL процедура у оном облику и мери која је неопходна за разумевање основа савременог приступа решавању SAT проблема. У глави 3 детаљно је представљена оригинална верзија технике чувања валуације, као и њене модификације предложене од стране оригиналних аутора. У глави 4 приказан је предлог побољшања оригиналног алгоритма и доказ његове коректности. У глави 5 представљени су неки детаљи имплементације у решавачу MiniSat, као и њена темељна евалуација. На крају, глава 6, као закључак, предлаже могућности за даљи рад и истраживање, уз осврт на постигнуте резултате у овом раду.

[1] `http://minisat.se/`

Глава 2

Основе

2.1 Нотација и основни појмови

У овом раду разматрамо задовољивост формула у конјуктивној нормалној форми (КНФ). Нека је V пребројив скуп променљивих. Литералом (енгл. *literal*) l називамо или променљиву v или њену логичку негацију $\neg v$. Супротан литерал литералу l означавамо са $\bar{l}$. Уколико је литерал l променљива v, њему супротан литерал је негација променљиве v, односно $\neg v$, а уколико је литерал l негација променљиве v, њему супротан литерал је управо v. КНФ формулу чини конјукција чланова, који представљају дисјункције литерала, а зовемо их клаузама (енгл. *clause*). Појединачну клаузу представљамо скупом литерала и означавамо ознаком C, док формулу у КНФ форми (у наставку само КНФ формула или формула)[1] представљамо скупом клауза са ознаком $\mathcal{F}$. Скуп свих променљивих формуле $\mathcal{F}$ означавамо са $V_{\mathcal{F}}$.

У овом раду, од посебне користи су коначне секвенце елемената које називамо листама (енгл. *list*). Припадност елемента листи означавамо уобичајеном ознаком $\in$, док листу без елемената ознаком $\varnothing$. Број елемената листе L означавамо са $|L|$. Позицију првог појављивања елемента e листе L, означавамо ознаком $\iota_L(e)$ (или само $\iota(e)$), при чему подразумевамо да припада скупу $\{0, 1, ..., |L| - 1\}$. Специјално, уколико су e_1 и e_2 елементи листе L и важи $\iota(e_1) < \iota(e_2)$ кажемо да се елемент e_1 појављује у листи L пре елемента e_2 или да e_1 претходи елементу e_2 у L. Слично, са $L[i]$, при чему важи $0 \leq i < |L|$, означавамо елемент који се налази на позицији i листе L. Користимо ознаку

[1] У овом раду, зарад читљивости, када год је из контекста јасно на који се појам одређени елемент или својство односи, користиће се краћи назив или запис.

$L - \{e\}$ за подлисту листе L добијену изостављањем првог појављивања елемента e. Супротно, листу добијену додавањем елемента e на крај листе L означавамо са $L + \{e\}$. Са $L_1 + L_2$ означавамо листу добијену конкатенацијом листи L_1 и L_2.

Под (парцијалном) валуацијом (енгл. *valuation*), подразумевамо листу непротивречних литерала, односно листу литерала где никоја два нису супротна. Означавамо је са $\mathcal{T}$ (често се користи и термин *трāг* - енгл. *trail*). Литерал l је тачан у валуацији $\mathcal{T}$, са ознаком $\mathcal{T} \models l$, уколико је елемент валуације $\mathcal{T}$, а нетачан у валуацији $\mathcal{T}$, са ознаком $\mathcal{T} \models \neg l$, уколико је њему супротан литерал елемент валуације $\mathcal{T}$. Уколико литерал l није ни тачан ни нетачан у валуацији $\mathcal{T}$, кажемо да је литерал l недефинисан у валуацији $\mathcal{T}$, са ознаком $\mathcal{T} \not\models l, \neg l$. Клауза C је тачна у валуацији $\mathcal{T}$, са ознаком $\mathcal{T} \models C$, уколико постоји бар један литерал $l \in C$ тако да $\mathcal{T} \models l$, а нетачна у валуацији $\mathcal{T}$, са ознаком $\mathcal{T} \models \neg C$, уколико за сваки литерал $l \in C$ важи $\mathcal{T} \models \neg l$. Нетачне клаузе у валуацији често се у пракси називају и конфликтним клаузама. Формула $\mathcal{F}$ је тачна у валуацији $\mathcal{T}$, са ознаком $\mathcal{T} \models \mathcal{F}$, уколико за сваку клаузу $C \in \mathcal{F}$ важи $\mathcal{T} \models C$, а нетачна у валуацији $\mathcal{T}$, са ознаком $\mathcal{T} \models \neg\mathcal{F}$, уколико постоји $C \in \mathcal{F}$ таква да важи $\mathcal{T} \models \neg C$. Формулу (клаузу) која није ни тачна ни нетачна у валуацији $\mathcal{T}$, називамо недефинисаном у $\mathcal{T}$.

У овом раду, од посебног значаја су неке особине које валуација може имати, а које уводимо у наставку:

- Валуација је *нередунданīна* уколико се сваки литерал који та валуација садржи појављује тачно једном.
- Валуација $\mathcal{T}$ је *īпōīпуна* у односу на формулу $\mathcal{F}$, уколико не постоји литерал над $V_{\mathcal{F}}$ који је недефинисан у валуацији $\mathcal{T}$.
- Валуација $\mathcal{T}$ је *безконфликīна* у односу на формулу $\mathcal{F}$, уколико не постоји нетачна (конфликтна) клауза $C \in \mathcal{F}$ у валуацији $\mathcal{T}$.

Такође, за валуацију кажемо да је *модел* формуле (клаузе) уколико је формула (клауза) у њој тачна. Приметимо да је валуација модел формуле акко је уједно и модел сваке њене клаузе. Формула (клауза) је *задовољива* ако има бар један модел. Проблем испитивања задовољивости исказне формуле назива се *SAT īпроблем*.

Као што је поменуто у уводу овог рада, најзначајнији напредак у решавању SAT проблема постигнут је захваљујући проналаску CDCL алгоритма [7] и његовој имплементацији у оквиру савремених SAT решавача. Отуда, овај алгоритам данас поредставља почетну тачку у развоју већине ефикасних техника за решавање SAT проблема. Том правилу није изузетак ни техника чувања валуације. Због тога, пре него што пређемо на њен детаљни опис, најпре ћемо у наставку ове главе описати CDCL алгоритам, у оној мери у којој је то неопходно за разумевање даљег излагања.

2.2 CDCL алгоритам

CDCL алгоритам [7] подразумева инкременталну конструкцију валуације додавањем једног по једног литерала, уз враћање уназад (у општем случају нехронолошко) уколико нека од клауза формуле у текућој валуацији постане нетачна (такву ситуацију називамо *конфликт*). У случају појаве конфликта, врши се *анализа конфликта*, којом се одређује тачка у претрази на коју се враћамо и од које процес претраге настављамо даље, другим путем.

Литерали се у валуацију додају или у виду претпостављених литерала (енгл. *decision literals*) или као литерали који су логичке последице претпостављених литерала (које називамо и *пропагирани литерали*). Приликом појаве конфликта, а у зависности од резултата анализе конфликта, одређени број претпостављених литерала са краја валуације ће бити уклоњен из валуације, заједно са свим пропагираним литералима који су њихове логичке последице. Уколико се конфликт појави у ситуацији у којој у валуацији нема претпостављених литерала, формула је незадовољива. Са друге стране, ако је валуација потпуна, а не постоји конфликт, формула је задовољива. Наиме, с обзиром да не постоји клауза која је у тој валуацији нетачна, а сви литерали имају дефинисану вредност, следи да свака клауза има бар један тачан литерал у себи (под претпоставком да формула не садржи празну клаузу, у ком случају је формула тривијално незадовољива).

Ознаком l^d означаваћемо да је l претпостављени литерал у валуацији $\mathcal{T}$. За сваки литерал $l \in \mathcal{T}$, дефинишемо *ниво одлучивања литерала l у валуацији $\mathcal{T}$*, у ознаци $decLvl_{\mathcal{T}}(l)$ (или само $decLvl(l)$) који је једнак броју претпостављених литерала $l^d \in \mathcal{T}$ таквих да важи $\iota(l^d) \leq \iota(l)$. Приметимо да, по дефиницији, за први претпостављени l^d важи $decLvl(l^d) = 1$, док за сваки литерал l који

му претходи у $\mathcal{T}$ важи $decLvl(l) = 0$. Често ће нам од значаја бити највећи (најдубљи) ниво литерала који се појављују у валуацији $\mathcal{T}$, са ознаком $L^{\mathcal{T}}_{deep}$ (или само L_{deep}). Приметимо да сви литерали истог нивоа k чине подсеквенцу узастопних литерала у валуацији $\mathcal{T}$. Ову подсеквенцу ћемо називати *k-тим нивоом одлучивања валуације* $\mathcal{T}$. Сви нивои валуације осим нултог садрже тачно један претпостављени литерал, којим тај ниво почиње. Нека је $\mathcal{T}[[i]]$ ознака за i-ти ниво одлучивања валуације $\mathcal{T}$. Дефинишемо $\mathcal{T}[[i...j]]$, тако да представља конкатенацију редом листи $T[[k]]$, за $i \leq k \leq j$.

Сваки пропагирани литерал има *разлог пропагације*, у ознаци $reason(l)$, а то је управо клауза на основу које је тај литерал додат у валуацију $\mathcal{T}$. С обзиром да претпостављени литерали немају разлог, дефинишемо $reason(l^d) = \varnothing$ за сваки претпостављени литерал $l^d \in \mathcal{T}$.

Пропагирани литерали се откривају у процесу који се назива *јединична пропагација* (енгл. *unit propagation*), а који се заснива на чињеници да, уколико су сви литерали неке клаузе осим једног нетачни, тада тај преостали литерал мора бити тачан, да би клауза била задовољена. За произвољну клаузу $C \in \mathcal{F}$ кажемо да је *јединична у односу на* $\mathcal{T}$ *за литерал* l, када је сваки литерал $l' \in C$, различит од l, нетачан у валуацији $\mathcal{T}$, док је l у истој недефинисан.

Приметимо да валуација $\mathcal{T}$ мора имати *сагласне разлоге*, што значи да ако је l пропагирани литерал и $reason(l) = C$, тада за сваки литерал $l' \in C$ различит од l, његов супротни литерал $\bar{l}'$ мора претходити l у валуацији $\mathcal{T}$. Такође, CDCL SAT решавачи по правилу врше тзв. *исцрпну пропагацију* (енгл. *exaustive propagation*), што значи да за сваку клаузу која постане јединична за неки литерал l у односу на валуацију $\mathcal{T}$ на неком нивоу k, литерал l ће бити пропагиран на том истом нивоу k. За валуацију добијену исцрпном пропагацијом кажемо да је *пропагацијски комплетна*.

У циљу ефикасног откривања литерала које треба пропагирати, претпоставимо да су за сваку клаузу $C \in \mathcal{F}$, такву да важи $|C| \geq 2$, изабрана два различита литерала $l', l'' \in C$, које називамо *посматраним литералима клаузе* C (енгл. *watched literals*). Скуп посматраних литерала клаузе C означавамо са $watches(C)$. За сваки литерал l, одржавамо листу $watchlist(l)$ свих клауза у којима је $\bar{l}$ тренутно један од посматраних литерала клаузе.

Упознати са неопходном нотацијом и нама значајним својствима валуације, можемо представити CDCL процедуру (алгоритам 1).

Размотримо основне елементе CDCL процедуре, са посебним акцентом на

Алгоритам 1 CDCL алгоритам

```
1:  CDCL(F)
2:      (C, T) ← PREPROCESS(F)
3:      if C ≠ ∅ then
4:          return UNSAT
5:      idx ← 0          ▷ обрађујемо watchlist(l) претходно додатих литерала
6:      while true do
7:          C ← PROPAGATE()
8:          if C ≠ ∅ then
9:              if L_deep = 0 then
10:                 return UNSAT
11:             (L_back, C_1−UIP, l_deep) ← ANALYSECONFLICT(C, T)
12:             BACKTRACK(L_back)
13:             F ← F ∪ {C_1−UIP}      ▷ T ⊭ l_deep, ¬l_deep, T ⊨ ¬(C_1−UIP \ {l_deep})
14:             T ← T + {l_deep};  reason(l_deep) ← C_1−UIP
15:         else
16:             if |T| = |V_F| then
17:                 return SAT                    ▷ Валуација T је потпуна
18:             l^d ← PICKBRANCHINGVARIABLE(T, F)
19:             T ← T + {l^d};  reason(l^d) ← ∅          ▷ L_deep ← L_deep + 1
20:
21: BACKTRACK(L_back)
22:     T ← T[[0...L_back]]                          ▷ L_deep ← L_back
23:     idx ← |T|
24:
25: PROPAGATE()
26:     while idx < |T| do
27:         l ← T[idx]
28:         for all C ∈ watchlist(l) do
29:             UPDATEWATCHEDLITERALS(C)
30:             if (∃l′ ∈ C)(T ⊭ l′, ¬l′ ∧ T ⊨ ¬(C \ {l′})) then
31:                 T ← T + {l′};  reason(l′) ← C          ▷ l′ ∈ watches(C)
32:             else if T ⊨ ¬C then
33:                 return C
34:         idx ← idx + 1
35:     return ∅
```

помоћне функције:

- PREPROCESS функција за сваку клаузу $C \in \mathcal{F}$, такву да важи $|C| \geq 2$, бира два различита литерала $l', l'' \in C$ за посматране литарале клаузе C и ажурира редом листе $watchlist(\bar{l}')$ и $watchlist(\bar{l}'')$. Празне клаузе су незадовољиве и уколико их формула садржи она је такође незадовољива. Додатно, за клаузу коју чини само један литерал, тај литерал се додаје у валуацију, или се пријављује незадовољивост, уколико њему супротни литерал већ припада валуацији. На крају, ако постоји клауза која је на овај начин постала нетачна, повратна вредност функције је управо та клауза (и формула је незадовољива). Иначе функција враћа $\varnothing$. Претпроцесирање, у пракси, најчешће укључује и различита поједностављивања почетне формуле.

- PROPAGATE врши јединичну пропагацију литерала. Уместо да решавач при сваком додавању новог литерала l у валуацију проверава за све клаузе формуле да ли су јединичне или конфликтне, довољно је проверити само клаузе у којима је супротан литерал $\bar{l}$ један од посматраних литерала (ово су управо клаузе из листе $watchlist(l)$). Заиста, како клауза која садржи бар два различита литерала која нису нетачна не може бити ни конфликтна ни јединична, проверавање таквих клауза је непотребно. За сваку од клауза $C \in watchlist(l)$, решавач тражи кандидата за нови посматрани литерал унутар клаузе (који у том тренутку није нетачан). Уколико не успе, а други посматрани литерал l' клаузе C је недефинисан, клауза је јединична и литерал l' се додаје у валуацију. Уколико је и литерал l' нетачан, PROPAGATE пријављује конфликт враћањем конфликтне клаузе C као повратне вредности. Ако решавач ипак пронађе новог кандидата за посматрани литерал клаузе C (означимо га са l^n), литерал $\bar{l}$ у $watches(C)$ се замењује тим новим литералом. На крају, клауза C се избацује из $watchlist(l)$ и додаје у $watchlist(\bar{l^n})$. UPDATEWATCHEDLITERALS помоћна функција имплементира описани механизам проналажења и постављања посматраних литерала појединачне клаузе, као и ажурирање листи. Ако се на овај начин за сваки додати литерал успешно заврши обрада одговарајуће листе клауза без појаве конфликта, PROPAGATE враћа $\varnothing$. Да би то било могуће, SAT решавач прати променљивом idx колико је листи $watchlist(l)$

новододатих литерала l валуације $\mathcal{T}$ до сада обрађено. Више детаља о схеми два посматрана литерала може се наћи у [7, 9].

- AnalyseConflict функција врши анализу конфликта. Приметимо да, ако смо наишли на конфликт на $L_{deep} \neq 0$ нивоу, претпоставка l^d којом тај ниво почиње је сигурно погрешна. Одатле можемо закључити да треба отклонити l^d и све пропагиране литерале тог нивоа и на тако добијену валуацију $T[[0...L_{deep} - 1]]$ додати $l_{deep} = \bar{l^d}$. Техникама попут анализе конфликтног графа [11] полазећи од конфликтне клаузе, могуће је пронаћи бољег кандидата за $l_{deep} \in T[[L_{deep}]]$ и заједно са њим *ниво повратног скока* L_{back}, који може бити значајно мањи од нивоа $L_{deep} - 1$. На овај се начин претрага значајно убрзава, а такође се проналазе и додатне логичке међузависности литерала у виду нове клаузе C_{1-UIP}, која се обично зарад спречавања понављања сличних конфликата у другим деловима простора претраге додаје (учи) у скуп клауза (уз постављање посматраних литерала по потреби). Треба посебно нагласити да се l_{deep} увек на крај добијене валуације $\mathcal{T}[[0...L_{back}]]$ додаје као логичка последица новонаучене јединичне клаузе C_{1-UIP}, у виду пропагираног литерала, тако да је $reason(l_{deep}) = C_{1-UIP}$[2]. Специјалан случај је када је клауза C_{1-UIP} клауза са једним литералом (управо l_{deep}). У том случају је ниво L_{back} једнак 0, а посматрани литерали клаузе C_{1-UIP} се не постављају.

- Backtrack функција врши повратни скок на ниво L_{back} који је одређен у фази анализе конфликта.

- PickBranchingVariable бира недефинисани литерал l^d који ће се додати у тренутну валауацију $\mathcal{T}$ као нови претпостављени литерал, чиме се започиње нови ниво одлучивања валуације $\mathcal{T}$. У пракси, SAT решавачи користе разне хеуристике за одабир новог претпостављеног литерала [9]. У овом раду оне неће бити представљене. Треба приметити да је, услед пропагацијске комплетности, последњи литерал потпуне валуације увек пропагирани литерал.

[2]Сагласност разлога се задржава, јер је циљ анализе конфликтног графа управо проналазак јединичне клаузе C_{1-UIP} у односу на $\mathcal{T}$ за литерал l_{deep}. Овакви чворови у конфликтном графу познати су под термином *јединствене тачке импликације* (енгл. *Unique Implication Point* (UIP)) [11].

$(\varnothing, [v_8]) \leftarrow$ PREPROCESS($\mathcal{F}$)
$v_6^d \leftarrow$ PICKBRANCHINGVARIABLE() $\mathcal{T}:\ \ v_8, v_6^d$
$\varnothing \leftarrow$ PROPAGATE() $\mathcal{T}:\ \ v_8, v_6^d, v_1, v_2$
$v_7^d \leftarrow$ PICKBRANCHINGVARIABLE() $\mathcal{T}:\ \ v_8, v_6^d, v_1, v_2, v_7^d$
$v_3^d \leftarrow$ PICKBRANCHINGVARIABLE() $\mathcal{T}:\ \ v_8, v_6^d, v_1, v_2, v_7^d, v_3^d$
$\{\neg v_2, \neg v_4, \neg v_5\} \leftarrow$ PROPAGATE() $\mathcal{T}:\ \ v_8, v_6^d, v_1, v_2, v_7^d, v_3^d, v_4, v_5$
$(1, \{\neg v_1, \neg v_2, \neg v_3\}, \neg v_3) \leftarrow$ ANALYSECONFLICT($\{\neg v_2, \neg v_4, \neg v_5\}, \mathcal{T}$)
BACKTRACK(1) $\mathcal{T}:\ \ v_8, v_6^d, v_1, v_2$ $\mathcal{F}:\ \ \mathcal{F}_0 \cup \{\neg v_1, \neg v_2, \neg v_3\}$ $\mathcal{T}:\ \ v_8, v_6^d, v_1, v_2, \neg v_3$
$\{\neg v_1, v_3, \neg v_5, \neg v_6\} \leftarrow$ PROPAGATE() $\mathcal{T}:\ \ v_8, v_6^d, v_1, v_2, \neg v_3, v_5$
$(0, \{\neg v_6\}, \neg v_6) \leftarrow$ ANALYSECONFLICT($\{\neg v_1, v_3, \neg v_5, \neg v_6\}, \mathcal{T}$)
BACKTRACK(0) $\mathcal{T}:\ \ v_8$ $\mathcal{F}:\ \ \mathcal{F}_0 \cup \{\neg v_1, \neg v_2, \neg v_3\} \cup \{\neg v_6\}$ $\mathcal{T}:\ \ v_8, \neg v_6$
$v_1^d \leftarrow$ PICKBRANCHINGVARIABLE() $\mathcal{T}:\ \ v_8, \neg v_6, v_1^d$
$\varnothing \leftarrow$ PROPAGATE() $\mathcal{T}:\ \ v_8, \neg v_6, v_1^d, v_2, \neg v_3$
$v_7^d \leftarrow$ PICKBRANCHINGVARIABLE() $\mathcal{T}:\ \ v_8, \neg v_6, v_1^d, v_2, \neg v_3, v_7^d$
$v_4^d \leftarrow$ PICKBRANCHINGVARIABLE() $\mathcal{T}:\ \ v_8, \neg v_6, v_1^d, v_2, \neg v_3, v_7^d, v_4^d$
$\varnothing \leftarrow$ PROPAGATE() $\mathcal{T}:\ \ v_8, \neg v_6, v_1^d, v_2, \neg v_3, v_7^d, v_4^d, \neg v_5$
return SAT

Табела 2.1: Приказ извршавања из примера 2.2.1

Пример 2.2.1. Нека је $\mathcal{F} = \{\{\neg v_1, v_2\}, \{\neg v_3, v_4\}, \{\neg v_1, \neg v_3, v_5\}, \{\neg v_2, \neg v_4, \neg v_5\}, \{\neg v_2, v_3, v_5, \neg v_6\}, \{\neg v_1, v_3, \neg v_5, \neg v_6, \neg v_8\}, \{v_8\}, \{v_1, \neg v_6, \}, \{v_1, v_7, \neg v_8\}\}$. Једно од могућих извршавања CDCL алгоритма дато је у табели 2.1.

На крају, наглашавамо да је детаљни опис неких делова CDCL алгоритма

(попут технике поновног започињања претраге, заборављања научених клауза, анализе конфликтног графа, описа хеуристике гранања и сл.) намерно изостављен. Сврха описа CDCL алгоритма датог у овом поглављу је да читаоца упозна са оним аспектима CDCL алгоритма који ће бити од значаја за даље излагање. Потпунији опис CDCL алгоритма може се наћи у [7, 11, 6].

Глава 3

Техника чувања валуације

3.1 Мотивација

У стандардном CDCL алгоритму, разлика између конфликтног нивоа L_{deep} и нивоа повратног скока L_{back} може бити прилично велика. Као што је раније наглашено, у таквим случајевима SAT решавачи често настављају претрагу на сличан начин као и пре повратног скока, добрим делом обнављајући поништени део валуације [5, 8, 10]. Најједноставније објашњење узрока ове појаве је велики број поновљених претпостављених литерала (услед хеуристике гранања која фаворизује литерале који су учествовали у скорашњим конфликтима) [9] и недовољно снажан утицај новонаучених логичких међузависности литерала (у виду научених клауза). Наиме, како проблеми на које се савремени SAT решавачи примењују садрже и до неколико десетина хиљада клауза, јасно је да уз понављање великог броја претпостављених литерала, додавање само једне клаузе C_{1-UIP} у формулу и њеног пропагираног литерала l_{deep} у валуацију, стање претраге ретко може да се одједном значајно промени. Овај феномен се може искористити за повећање ефикасности рада SAT решавача, тако што се поништени део валуације сачува, а затим се информације сачуване у њему користе за убрзавање даље изградње валуације [5]. Техника чувања валуације коју разматрамо у овом раду заснована је на овом приступу.

Други приступ је да се приликом удаљених повратних скокова не поништава заиста већи део валуације, већ да литерали на појединим нивоима вишим од нивоа повратног скока остану у валуацији и након повратног скока. Овај приступ се у литератури означава са *C-bt* [10, 8] (што је скраћено од

енглеског термина *chronological backtracking*, иако је приступ заправо комбинација хронолошког и нехронолошког враћања уназад). Ова техника дозвољава решавачу да се врати на било који ниво j, тако да важи $L_{back} \leq j < L_{deep}$ и тиме разреши конфликт на L_{deep} нивоу. Приметимо да на тако изабраном нивоу, научена клауза C_{1-UIP} је и даље јединична у односу на $\mathcal{T}$ за l_{deep}, због чега се као и иначе поставља за разлог његове пропагације. Ниво $decLvl(l_{deep})$ такође се поставља на ниво повратног скока L_{back} који је одређен анализом конфликта, чак и за $j \neq L_{back}$. На тај начин се пропагацијска комплетност валуације одржава, али последица је да валуација није више подељена на непрекидне секвенце литерала истог нивоа, што доводи до, како имплементационих, тако и теоријских компликација, с обзиром да поједине уобичајене инваријанте алгоритма престају да важе. Ипак, показано је у [8] да ова техника не нарушава коректност CDCL алгоритма.

Са друге стране, код технике чувања валуације фокус је на убрзавању пропагације литерала након повратног скока. У наставку главе ова техника ће бити приказана у свом оригиналном облику, први пут објављеном на SAT конференцији 2020. године [5]. Ипак, треба напоменути да је оригинални алгоритам овде у мањој мери измењен и прилагођен, како би се лакше уклопио у раније уведену нотацију код CDCL алгоритма. Такође, техника је представљена у нешто прецизнијој форми, тако да модификације (и унапређења дата у наставку рада) заједно са основним обликом алгоритма чине компактнију целину у теоријском смислу, што у оригиналном раду, услед ограниченог простора, није било могуће.

3.2 Приказ технике

Претпоставимо да имамо конфликт на L_{deep} нивоу, као и да је у процесу анализе овог конфликта одређено да је потребно вратити се на ниво L_{back}. Приликом повратног скока, поред текућег L_{deep} нивоа, биће поништен и део валуације $\mathcal{T}[[L_{back} + 1...L_{deep} - 1]]$. Техника чувања валуације подразумева чување овог дела валуације у помоћној листи $\mathcal{T}_{save}$, пре његовог уклањања. Приликом чувања валуације, уз сваки сачувани литерал l_{save} из $\mathcal{T}_{save}$, чува се и његов разлог $reason(l_{save})$ (сачувани разлог означаваћемо са $reason_{save}(l_{save})$). Литерали сачувани у листи $\mathcal{T}_{save}$ могу бити искоришћени за убрзавање процеса изградње валуације у наставку претраге, под одређеним условима који

ће бити детаљније објашњени у овом поглављу.

Детаљан поступак чувања валуације, као и употребе сачуваних литерала за убрзавање пропагације дат је у алгоритму 2. Заправо, алгоритам 2 приказује на који начин су поједине процедуре основног CDCL алгоритма модификоване у циљу имплементације ове технике. Претпостављамо да је листа сачуваних литерала $\mathcal{T}_{save}$ на почетку празна.

Размотримо сада детаљније основне елементе у имплементацији ове технике:

- Backtrack функција је допуњена тако да чува део тренутне валуације $\mathcal{T}$, почев од $L_{back} + 1$ нивоа, у виду листе $\mathcal{T}_{save}$. Треба обратити пажњу да се у представљеној основној варијанти алгоритма, листа сачуваних литерала $\mathcal{T}_{save}$, као и одговарајући сачувани разлози, након сваког конфликта поново постављају на нове вредности.

- Propagate функција је такође допуњена тако да, пре обраде сваке од листи $watchlist(l)$, прво позива UseSavedTrail. На овај начин када год CDCL дода литерале у валуацију $\mathcal{T}$, проверава се прво да ли је могуће искористити листу $\mathcal{T}_{save}$ како би се уштедело на времену при пропагацији литерала. Овај приступ почива на претпоставци да је једноставно копирање пропагираних литерала из сачуване валуације, временски мање захтеван процес од механизма којим CDCL иначе проланази пропагиране литерале из јединичних клауза.

- UseSavedTrail представља срж технике чувања валуације и служи да додаје у валуацију пропагиране литерале из $\mathcal{T}_{save}$ који су недефинисани у $\mathcal{T}$ или да у случају појављивања пропагираног литерала из $\mathcal{T}_{save}$ који је нетачан у $\mathcal{T}$ пријави конфликт. Литерали из листе $\mathcal{T}_{save}$ се обрађују редом у ком се налазе у листи, а литерал који тренутно обрађујемо називамо *пивотом*. Како је листа сачуваних литерала $\mathcal{T}_{save}$ подељена на делове који представљају поништене нивое одлучивања претходне валуације, она за сваки ниво, садржи најпре претпостављени литерал l^d_{save}, а затим и све његове логичке последице (у односу на стање формуле $\mathcal{F}_{old}$ и валуације $\mathcal{T}_{old}$ непосредно пре последњег повратног скока). У случају да приликом обраде сачувани претпостављени литерал l^d_{save} (тренутни пивот) није тачан у валуацији $\mathcal{T}$, пропагација недефинисаних литерала

Алгоритам 2 Техника чувања валуације

1: BACKTRACK(L_{back})
2: $\mathcal{T}_{save} \leftarrow \mathcal{T}[[L_{back}+1...L_{deep}-1]]$ ▷ Не чувамо конфликтни ниво L_{deep}
3: **for all** $l_{save} \in \mathcal{T}_{save}$ **do**
4: $reason_{save}(l_{save}) \leftarrow reason(l_{save})$
5: $\mathcal{T} \leftarrow \mathcal{T}[[0...L_{back}]]$
6: $idx \leftarrow |\mathcal{T}|$
7:
8: PROPAGATE()
9: **while** idx $< |\mathcal{T}|$ **do**
10: $C \leftarrow$ USESAVEDTRAIL()
11: **if** $C \neq \varnothing$ **then**
12: **return** C
13: $l \leftarrow \mathcal{T}[idx]$
14: **for all** $C \in watchlist(l)$ **do**
15: UPDATEWATCHEDLITERALS(C)
16: **if** $(\exists l' \in C)(\mathcal{T} \not\models l', \neg l' \wedge \mathcal{T} \models \neg(C \setminus \{l'\}))$ **then**
17: $\mathcal{T} \leftarrow \mathcal{T} + \{l'\}$; $reason(l') \leftarrow C$
18: **else if** $\mathcal{T} \models \neg C$ **then**
19: **return** C
20: $idx \leftarrow idx + 1$
21: **return** $\varnothing$
22:
23: USESAVEDTRAIL()
24: **while** $|\mathcal{T}_{save}| > 0$ **do** ▷ Док нису сви сачувани литерали тачни у $\mathcal{T}$
25: $l_{save} \leftarrow \mathcal{T}_{save}[0]$
26: **if** $reason_{save}(l_{save}) = \varnothing$ **then**
27: **if** $\mathcal{T} \models l_{save}$ **then**
28: $\mathcal{T}_{save} \leftarrow \mathcal{T}_{save} - \{l_{save}\}$ ▷ Не чувамо већ тачне литерале
29: **else**
30: **return** $\varnothing$ ▷ Не утичемо на доношење претпоставки
31: **else** ▷ $\mathcal{T} \models \neg(reason_{save}(l_{save}) \setminus \{l_{save}\})$
32: **if** $\mathcal{T} \models l_{save}$ **then**
33: $\mathcal{T}_{save} \leftarrow \mathcal{T}_{save} - \{l_{save}\}$ ▷ Не чувамо већ тачне литерале
34: **else if** $\mathcal{T} \models \neg l_{save}$ **then**
35: **return** $reason_{save}(l_{save})$ ▷ $\mathcal{T} \models \neg(reason_{save}(l_{save}))$
36: **else** ▷ $\mathcal{T} \not\models l_{save}, \neg l_{save}$
37: $\mathcal{T} \leftarrow \mathcal{T} + \{l_{save}\}$; $reason(l_{save}) \leftarrow reason_{save}(l_{save})$
38: $\mathcal{T}_{save} \leftarrow \mathcal{T}_{save} - \{l_{save}\}$ ▷ Унапред бришемо литерал
39: **return** $\varnothing$

који следе за l^d_{save} у $\mathcal{T}_{save}$ није логички оправдана, јер не представљају нужно логичке последице тренутне формуле $\mathcal{F}$ и валуације $\mathcal{T}$, којој l^d_{save} не припада. Како не желимо да утичемо на избор претпостављених литерала, у том случају l^d_{save} се не додаје у валуацију, већ се обрада литерала из $\mathcal{T}_{save}$ прекида. У супротном, ако је сачувани претпостављени литерал l^d_{save} ипак тачан у $\mathcal{T}$ (као претпостављени или као пропагирани литерал), можемо наставити са обрадом сачуваних литерала који се у $\mathcal{T}_{save}$ налазе на истом нивоу као и l^d_{save}. Приметимо да унутар позива UseSavedTrail функције важи да је $\mathcal{T}_{old}[[0...L^{old}_{back}]]$ префикс валуације $\mathcal{T}$, где је L^{old}_{back} ниво последњег повратног скока и $\mathcal{T}_{old}$ валуација непосредно пре истог. Имајући то у виду, како тренутна формула, уз новонаучену клаузу C_{1-UIP}, садржи исте клаузе као и пре повратног скока, знамо да за све остале литерале $l_{save} \in \mathcal{T}_{save}$ који су на истом нивоу као и l^d_{save} листе $\mathcal{T}_{save}$ важи $\mathcal{T} \models \neg(reason_{save}(l_{save}) \setminus \{l_{save}\}))$ у тренутку његове обраде у листи $\mathcal{T}_{save}$. Заиста, приметимо да је то важило непосредно пре последњег повратног скока у тренутку када је префикс валуације био $\mathcal{T}_{old}[[0...L^{old}_{back}]] + l^d_{save}$ (у односу на тадашњу формулу $\mathcal{F}_{old} = \mathcal{F} \setminus \{C_{1-UIP}\}$), за којом су следили сви литерали из T_{save} који претходе датом литералу l_{save}. Како услед секвенцијалне обраде литерала из $\mathcal{T}_{save}$, све те литерале имамо и сада у валуацији $\mathcal{T}$, јасно је да клауза $reason_{save}(l_{save})$ мора бити или јединична, или конфликтна, или тачна, у зависности од вредности литерала l_{save} у тренутној валуацији. Сходно томе, обраду литерала настављамо редом за сваки сачувани ниво из $\mathcal{T}_{save}$, док не наиђемо на конфликт или на претпостављени литерал који није тачан у тренутној валуацији $\mathcal{T}$.

Пример 3.2.1. Нека је $\mathcal{F} = \{\{\neg v_{13}, \neg v_{14}\}, \{\neg v_1, v_7, v_{12}\}, ..., \{\neg v_3, \neg v_7, v_9, v_{12}\}\}$. Део једног од могућих извршавања CDCL алгоритма са техником чувања валуације дат је у табели 3.1.

3.3 Доказ коректности

У наставку следи доказ коректности алгоритма у сличном облику какав је представљен у оригиналном раду [5]. Како уведена техника чувања валуације утиче само на додавање литерала у валуацију и откривање конфликата који су узроковани литералима сачуваним у $\mathcal{T}_{save}$, доказ коректности технике

...
$v_{12}^d \leftarrow$ PickBranchingVariable()
$\mathcal{T}:\ \ v_1^d, v_2, v_3^d, v_4, v_5^d, v_6, v_7^d, v_8, v_9^d, v_{10}, v_{11}, v_{12}^d$
$\{\neg v_{13}, \neg v_{14}\} \leftarrow$ Propagate()
$\mathcal{T}:\ \ v_1^d, v_2, v_3^d, v_4, v_5^d, v_6, v_7^d, v_8, v_9^d, v_{10}, v_{11}, v_{12}^d, v_{13}, v_{14}$
$(2, \{\neg v_1, \neg v_3, \neg v_{12}\}, \neg v_{12}) \leftarrow$ AnalyseConflict$(\{\neg v_{13}, \neg v_{14}\}, \mathcal{T})$
Backtrack(2)
$\mathcal{T}_{save}:\ \ v_5^d, v_6, v_7^d, v_8, v_9^d, v_{10}, v_{11}$
$\mathcal{T}:\ \ v_1^d, v_2, v_3^d, v_4$
$\mathcal{F}:\ \ \mathcal{F}_0 \cup \{\neg v_1, \neg v_3, \neg v_{12}\}$
$\mathcal{T}:\ \ v_1^d, v_2, v_3^d, v_4, \neg v_{12}$
$\varnothing \leftarrow$ Propagate()
$\mathcal{T}:\ \ v_1^d, v_2, v_3^d, v_4, \neg v_{12}, v_7, v_9$
$v_{13}^d \leftarrow$ PickBranchingVariable()
$\mathcal{T}:\ \ v_1^d, v_2, v_3^d, v_4, \neg v_{12}, v_7, v_9, v_{13}^d$
$\varnothing \leftarrow$ Propagate()
$\mathcal{T}:\ \ v_1^d, v_2, v_3^d, v_4, \neg v_{12}, v_7, v_9, v_{13}^d, \neg v_{14}$
$v_5^d \leftarrow$ PickBranchingVariable()
$\mathcal{T}:\ \ v_1^d, v_2, v_3^d, v_4, \neg v_{12}, v_7, v_9, v_{13}^d, \neg v_{14}, v_5^d$
$\varnothing \leftarrow$ Propagate()
$\mathcal{T}:\ \ v_1^d, v_2, v_3^d, v_4, \neg v_{12}, v_7, v_9, v_{13}^d, \neg v_{14}, v_5^d, v_8$
$\varnothing \leftarrow$ UseSavedTrail
$\mathcal{T}_{save}:\ \ \varnothing$
$\mathcal{T}:\ \ v_1^d, v_2, v_3^d, v_4, \neg v_{12}, v_7, v_9, v_{13}^d, \neg v_{14}, v_5^d, v_8, v_6, v_{10}, v_{11}$
...

Табела 3.1: Приказ дела извршавања из примера 3.2.1

се своди на доказивање да описане модификације CDCL алгоритма не нарушавају сагласност разлога валуације $\mathcal{T}$ током рада решавача, па самим тим ни својство сагласности самог алгоритма (што значи да за сваку формулу за коју алгоритам пријави незадовољивост, та формула је заиста незадовољива). С обзиром на начин рада технике чувања литерала, довољно је показати да су за сваки сачувани пропагирани литерал $l_{save} \in \mathcal{T}_{save}$, сви литерали клаузе $reason_{save}(l_{save})$ осим самог литерала l_{save} нетачни у валуацији $\mathcal{T}$ у тренутку обраде l_{save}.

Нека је, као и раније, са $\mathcal{T} + \mathcal{T}_{save}$ означено надовезивање ове две листе литерала, при чему подразумевамо да су „надовезани" и њихови разлози. Наиме, подразумеваћемо да литерали l и l_{save} који се редом појављују у подлистама $\mathcal{T}$ и $\mathcal{T}_{save}$ имају редом разлоге $reason(l)$ и $reason_{save}(l_{save})$, при чему

може да важи $l = l_{save}$ и $reason(l) \neq reason_{save}(l_{save})$. Дакле, с обзиром да се исти литерал може појављивати и у $\mathcal{T}$ и у $\mathcal{T}_{save}$, разлог који придружујемо том литералу може зависити од тога које појављивање литерала разматрамо. Имајући ово у виду, својство сагласности разлога за листу $\mathcal{T} + \mathcal{T}_{save}$ се дефинише на уобичајен начин, с тим што се за сваку појаву неког литерала разматра онај разлог који је придружен тој појави. Ако није из контекста јасно, разлог (појављивања) литерала ће увек бити експлицитно наглашен.

Теорема 1. Нека $\mathcal{T} + \mathcal{T}_{save}$ има својство сагласности разлога. Ако првих i литерала из $\mathcal{T}_{save}$ уједно припада и $\mathcal{T}$, и ако је $\mathcal{T}_{save}[i] = l$ пропагирани литерал у $\mathcal{T}_{save}$ са разлогом $reason_{save}(l) = C$, онда важи $\mathcal{T} \models \neg(C \setminus l)$.

Доказ. Како $\mathcal{T} + \mathcal{T}_{save}$ има сагласне разлоге, супротан литерал $\bar{l'}$ сваког литерала l' клаузе C различитог од l претходи литералу l у $\mathcal{T} + \mathcal{T}_{save}$. Отуд за $l' \in C \setminus \{l\}$, важи или $\bar{l'} \in \mathcal{T}$ или $\bar{l'} \in \{\mathcal{T}_{save}[j] | 0 \leq j < i\}$. У другом случају $\mathcal{T} \models \bar{l'}$ важи из претпоставке теореме. Из свега наведеног, непосредно важи $\mathcal{T} \models \neg(C \setminus \{l\})$. □

Из претходне теореме следи да уколико SAT решавач одржава сагласност разлога за $\mathcal{T} + \mathcal{T}_{save}$, тада ће и сама валуација $\mathcal{T}$ имати сагласне разлоге у сваком тренутку, услед секвенцијалне обраде сачуваних литерала листе $\mathcal{T}_{save}$.

Дакле, за потпуни доказ коректности, довољно је показати да решавач увек одржава својство сагласности разлога за $\mathcal{T} + \mathcal{T}_{save}$.

Став 1. Ако $\mathcal{T} + \mathcal{T}_{save}$ има сагласне разлоге, онда и $\mathcal{T}' + \mathcal{T}'_{save}$ има сагласне разлоге у свим наредним случајевима:

1. $\mathcal{T}' = \mathcal{T}$, $\mathcal{T}'_{save} = \mathcal{T}_{save} \setminus \{\mathcal{T}_{save}[0]\}$ и $\mathcal{T}_{save}[0] \in \mathcal{T}$
2. $\mathcal{T}' = \mathcal{T} + \mathcal{T}_{save}[0]$ и $\mathcal{T}'_{save} = \mathcal{T}_{save} \setminus \{\mathcal{T}_{save}[0]\}$
3. $\mathcal{T}' = \mathcal{T} + \mathcal{T}_{new}$, $\mathcal{T}'_{save} = \mathcal{T}_{save}$ и $\mathcal{T}'$ има сагласне разлоге

Доказ. (1) Како се литерал $\mathcal{T}_{save}[0]$ већ појављује у $\mathcal{T}$, можемо га уклонити из $\mathcal{T}_{save}$, тако да разлози литерала у $\mathcal{T}_{save}$ којима је избачени литерал претходио остану сагласни. (2) Очигледан случај, јер је $\mathcal{T} + \mathcal{T}_{save} = \mathcal{T}' + \mathcal{T}'_{save}$ (3) $\mathcal{T} + \mathcal{T}_{new}$ има сагласне разлоге по претпоставци. Због тога, сагласност разлога за $\mathcal{T}_{save}$ зависи само од појављивања литерала у $\mathcal{T}$ и $\mathcal{T}_{save}$, па $\mathcal{T}' + \mathcal{T}'_{save}$ такође има сагласне разлоге, с обзиром да садржи неопходне литерале. □

Теорема 2. Ако је $\mathcal{T} + \mathcal{T}_{save}$ генерисана техником чувања валуације, онда $\mathcal{T} + \mathcal{T}_{save}$ има својство сагласности разлога.

Доказ. $\mathcal{T}$ и $\mathcal{T}_{save}$ су у почетку празне, па је тривијално $\mathcal{T} + \mathcal{T}_{save}$ сагласних разлога. Претпоставимо да у неком тренутку рада решавача листа $\mathcal{T} + \mathcal{T}_{save}$ има сагласне разлоге и докажимо да ће сагласност разлога бити сачувана и након следеће операције процедуре. Размотримо понаособ сваку од операција процедуре које утичу на валуацију $\mathcal{T}$ и листу сачуваних литерала $\mathcal{T}_{save}$:

1. Решавач може да додаје литерале у $\mathcal{T}$ без коришћења $\mathcal{T}_{save}$, градећи валуацију $\mathcal{T}' = \mathcal{T} + \mathcal{T}_{new}$, па $\mathcal{T}'$ има сагласне разлоге, с обзиром да је генерисана искључиво применом операција основног CDCL алгоритма. Отуда, добијена листа $\mathcal{T}' + \mathcal{T}_{save}$ такође има сагласне разлоге (став 1 – случај 3).

2. У функцији UseSavedTrail литерали се или бришу са почетка $\mathcal{T}_{save}$, јер већ припадају тренутној валуацији $\mathcal{T}$, или се премештају са почетка $\mathcal{T}_{save}$ на крај валуације $\mathcal{T}$. У оба случаја сагласност разлога за добијену листу $\mathcal{T}' + \mathcal{T}'_{save}$ се одржава (став 1 – случајеви 1 и 2).

3. Backtrack функција повратним скоком генерише листу $\mathcal{T}' + \mathcal{T}'_{save}$, која је једнака префиксу $\mathcal{T}[[0...L_{deep} - 1]]$ валуације $\mathcal{T}$. Како за валуацију $\mathcal{T}$ важи да је сагласна као префикс листе са сагласним разлозима $\mathcal{T} + \mathcal{T}_{save}$, то исто важи и за њен префикс $\mathcal{T}[[0...L_{deep} - 1]]$.

Дакле, $\mathcal{T} + \mathcal{T}_{save}$ задржава својство сагласности разлога при свакој од операција процедуре, па отуда ово својство остаје да важи током читавог рада решавача. □

На крају, напоменимо да би, формално, за комплетан доказ коректности модификованог алгоритма било потребно доказати да и својства заустављања и потпуности (која CDCL поседује [6]) остају на снази и након модификације алгоритма. Својство заустављања значи да се претрага увек зауставља након одређеног броја корака, док потпуност подразумева да се за сваку незадовољиву формулу тада пријави незадовољивост. Ипак, како се сачувана валуација користи искључиво за ефикасније проналажење пропагираних литерала и нетачних клауза, потпуност и својство заустављања директно следе из одговарајућих својстава CDCL алгоритма који преузима процес претраге

на уобичајени начин, након што се искористе литерали из сачуване валуације. Једини утицај модификације на остатак претраге је у евентуално измењеном редоследу којим се пропагирани литерали додају у валуацију, као и промењени разлози њиховог додавања, што не утиче на потпуност и заустављање алгоритма.

3.4 Модификације

Надовезивање сачуваних валуација

Како се у раду SAT решавача конфликти често дешавају, у основном облику технике чувања валуације неретко можемо очекивати да добар део сачуване валуације остаје неискоришћен и замењен новим литералима. Из овог разлога, решавач не успева да у потпуности искористи сачуване литерале. Једна од модификација предложених у оригиналном раду покушава да реши овај проблем тако што омогућава надовезивање сачуваних делова валуације, уместо да претходно сачувану валуацију брише при првом следећем позиву Backtrack функције. Овако модификована техника чувања валуације приказана је у алгоритму 3. Подразумевамо да idx_{pivot} на почетку претраге почиње са вредношћу 0.

Унутар функција Propagate и UseSavedTrail измене су минималне. Уместо моменталног брисања успешно обрађеног литерала $l_{save} \in \mathcal{T}_{save}$, ова модификација користи вредност idx_{pivot} за праћење пивота у $\mathcal{T}_{save}$. На овај начин $\mathcal{T}_{save}$ остаје непромењена све док се не заврши потпуна пропагација на тренутном безконфликтном L_{deep} нивоу. Додатне линије унутар Propagate функције омогућавају да тек тада измене листе $\mathcal{T}_{save}$ постану трајне, одбацивањем свих успешно обрађених литерала. Приметимо да по структури UseSavedTrail функције, литерали који претходе пивоту већ припадају валуацији $\mathcal{T}$ у том тренутку. С друге стране, у случају конфликта, потврђивање измена листе $\mathcal{T}_{save}$ се прескаче. Ово је од посебне важности за омогућавање надовезивање нивоа сачуваних валуација. Наиме, уколико би направљене измене листе $\mathcal{T}_{save}$ одмах биле трајне, услед изостављања конфликтног L_{deep} нивоа одлучивања из $\mathcal{T}_{save}$ при позиву функције Backtrack, постојала би могућност да неки од несачуваних литерала са L_{deep} нивоа наруше сагласност разлога листе $\mathcal{T}_{save}$ на коју се надовезује нова. Уколико, у жељи да заобиђемо

Алгоритам 3 Техника чувања валуације - Модификација 1

```
1:  Backtrack(L_back)
2:      if L_deep = L_back^old then
3:          T_save ← ∅                    ▷ Надовезивање не одржава нужно сагласност
4:      L_back^old ← L_back               ▷ Памтимо ниво последњег повратног скока
5:      T_save ← T[[L_back + 1...L_deep − 1]] + T_save   ▷ Изостављамо L_deep ниво
6:      for all l_save ∈ T[[L_back + 1...L_deep − 1]] do   ▷ Чувамо нове разлоге
7:          reason_save(l_save) ← reason(l_save)
8:      T ← T[[0...L_back]]
9:      idx ← |T|
10:     idx_pivot ← 0
11:
12: Propagate()
13:     while idx < |T| do
14:         ...
15:         idx ← idx + 1
16:     while idx_pivot > 0 do
17:         T_save ← T_save − {T_save[0]}          ▷ Потврђујемо измене T_save
18:         idx_pivot ← idx_pivot − 1
19:     return ∅
20:
21: UseSavedTrail()
22:     while idx_pivot < |T_save| do   ▷ Док постоји необрађени литерал l_save
23:         l_save ← T_save[idx_pivot]
24:         if reason_save(l_save) = ∅ then
25:             if T ⊨ l_save then
26:                 idx_pivot ← idx_pivot + 1
27:             else
28:                 return ∅
29:         else
30:             if T ⊨ l_save then
31:                 idx_pivot ← idx_pivot + 1
32:             else if T ⊨ ¬l_save then
33:                 return reason_save(l_save)
34:             else
35:                 T ← T + {l_save}; reason(l_save) ← reason_save(l_save)
36:                 idx_pivot ← idx_pivot + 1
37:     return ∅
```

овај проблем, модификујемо функцију Backtrack тако да се приликом чувања валуације чува и L_{deep} ниво валуације $\mathcal{T}$, стари део листе $\mathcal{T}_{save}$ постаје

бескорисан, јер му увек претходи сачувани конфликт.

Најзад, ова модификација укључује и линије 5-7 у функцији Backtrack. Оне омогућавају да се на стару листу сачуваних литерала $\mathcal{T}_{save}$ (која укључује и непотврђене измене), надовеже новопоништени део валуације $\mathcal{T}[[L_{back}+1...L_{deep}-1]]$, уз чување разлога сачуваних литерала. Приметимо да овако добијена листа $\mathcal{T}_{save}$, за разлику од валуације $\mathcal{T}$ може садржати редундантне литерале, као и међусобно супротне литерале. Штавише, сачувани разлози литерала се увек замењују новим разлозима из управо поништеног дела валуације. Срећом ово не представља проблем. С обзиром да се сачувани литерали листе $\mathcal{T}_{save}$ обрађују секвенцијално унутар UseSavedTrail функције, вишеструка појављивања литерала се једноставно прескачу, док се међусобно супротни литерали никада не пропагирају оба, већ се приликом обраде каснијег од та два она зауставља, уз могуће пријављивање конфликта. Отуд додавање новог појављивања неког литерала у листу $\mathcal{T}_{save}$ и памћење његовог новог разлога уместо старог увек представља безбедну операцију. Ипак због надовезивања, дужина листе $\mathcal{T}_{save}$ није ограничена бројем променљивих које се појављују у формули (као што је случај са валуацијом $\mathcal{T}$), па може неограничено да расте. Због тога, неопходно је обезбедити механизам који повремено „чисти" листу сачуваних литерала $\mathcal{T}_{save}$, отклањајући из ње дупликате. Са друге стране, уколико постоји литерал l коме претходи њему супротан литерал $\bar{l}$ у листи $\mathcal{T}_{save}$, сви литерали који следе иза l могу бити уклоњени из $\mathcal{T}_{save}$. Уколико је, додатно, литерал l претпостављени литерал у $\mathcal{T}_{save}$, тада се и он може одбацити.

Приметимо да унутар Backtrack функције имамо специјални случај уколико је тренутни конфликтни ниво L_{deep} уједно и ниво последњег повратног скока L_{back}^{old} (тј. када се нови конфликт деси одмах након повратног скока, пре првог следећег претпостављеног литерала). У овом случају сагласност разлога за $\mathcal{T}+\mathcal{T}_{save}$ може бити нарушена. Наиме, како увек одбацујемо конфликтни ниво, а како је у овом случају то управо ниво на који смо се вратили повратним скоком и који је могао садржати литерале који су заслужни за пропагацију литерала из $\mathcal{T}_{save}$, изостављање литерала са тог нивоа може утицати на сагласност разлога литерала из $\mathcal{T}_{save}$. Аутори оригиналног рада [5] су били свесни овог проблема, иако га нису споменули у самом раду. Њихово једноставно решење је имплементирано у оквиру Cadical SAT решавача[1], а сводило

[1] `https://github.com/rgh000/cadical-trail.git`

се на то да решавач прати број претпостављених литерала које направи од последњег повратног скока. Уколико је тај број једнак нули, конфликтни ниво је управо ниво последњег повратног скока L^{old}_{back} и $\mathcal{T}_{save}$ се брише непосредно пре надовезивања нове валуације (алгоритам 3, линије 2-3). У супротном, повратни скок се одвија уобичајеним током. Следећи пример илуструје дати случај, као и функционалност модификованог алгоритма.

Пример 3.4.1. Нека је $\mathcal{F} = \{\{\neg v_{13}, \neg v_{14}\}, \{\neg v_1, v_7, v_{12}\}, ..., \{\neg v_3, \neg v_7, v_9, v_{12}\}\}$. Једно од могућих извршавања CDCL алгоритма са надовезивањем сачуваних валуација дато је у табели 3.2. Посебну пажњу треба обратити на брисање сачуване валуације кад важи $L_{deep} = L^{old}_{back}$.

Ово једноставно решење, иако функционише, може имати неке недостатке у смислу ефикасности, које следећа глава истражује. У њој је предложен и нови ефикаснији приступ, који је уједно и оригинални допринос овог рада.

Доказ коректности

Докажимо да уведена модификација, која подразумева надовезивање сачуваних валуација, задржава коректност алгоритма. Приметимо да је довољно прилагодити само теорему 2 новој модификацији, као и увести следећи став који осликава нови начин уношења елемената из листе $\mathcal{T}_{save}$ у тренутну валуацију $\mathcal{T}$.

Став 2. Ако $\mathcal{T} + \mathcal{T}_{save}$ има сагласне разлоге и важи да су првих i литерала листе $\mathcal{T}_{save}$ уједно и елементи валуације $\mathcal{T}$, тако да пропагирани литерал $\mathcal{T}_{save}[i] = l$ из $\mathcal{T}_{save}$ не припада валуацији $\mathcal{T}$, онда $\mathcal{T}' + \mathcal{T}_{save}$ има сагласне разлоге, где је $\mathcal{T}' = \mathcal{T} + \{l\}$, при чему важи $reason(l) = reason_{save}(l)$ у проширеној валуацији $\mathcal{T}'$.

Доказ. Сагласност разлога валуације $\mathcal{T}'$ следи директно из теореме 1. Одатле следи и сагласност разлога за $\mathcal{T}' + \mathcal{T}_{save}$ (став 1 – случај 3). □

Теорема 3. Ако је $\mathcal{T} + \mathcal{T}_{save}$ генерисана модификованом техником чувања валуације, онда $\mathcal{T} + \mathcal{T}_{save}$ има својство сагласности разлога.

Доказ. У почетку процедуре, не постоје сачувани литерали, због чега у прво време валуацију решавач генерише искључиво операцијама CDCL алгоритма.

...
BACKTRACK(2) $\mathcal{T}_{save}: \quad v_5^d, v_6, v_7^d, v_8, v_9^d, v_{10}, v_{11}$ $\mathcal{T}: \quad v_1^d, v_2, v_3^d, v_4$ $\mathcal{F}: \quad \mathcal{F}_0 \cup \{\neg v_1, \neg v_3, \neg v_{12}\}$ $\mathcal{T}: \quad v_1^d, v_2, v_3^d, v_4, \neg v_{12}$
$\varnothing \leftarrow$ PROPAGATE() $\mathcal{T}: \quad v_1^d, v_2, v_3^d, v_4, \neg v_{12}, v_7, v_9, \neg v_{11}$
$v_{13}^d \leftarrow$ PICKBRANCHINGVARIABLE() $\mathcal{T}: \quad v_1^d, v_2, v_3^d, v_4, \neg v_{12}, v_7, v_9, \neg v_{11}, v_{13}^d$
$\varnothing \leftarrow$ PROPAGATE() $\mathcal{T}: \quad v_1^d, v_2, v_3^d, v_4, \neg v_{12}, v_7, v_9, \neg v_{11}, v_{13}^d, \neg v_{14}$
$v_{15}^d \leftarrow$ PICKBRANCHINGVARIABLE() $\mathcal{T}: \quad v_1^d, v_2, v_3^d, v_4, \neg v_{12}, v_7, v_9, \neg v_{11}, v_{13}^d, \neg v_{14}, v_{15}^d$
$\{\neg v_5, \neg v_9, v_{11}\} \leftarrow$ PROPAGATE() $\mathcal{T}: \quad v_1^d, v_2, v_3^d, v_4, \neg v_{12}, v_7, v_9, v_{13}^d, \neg v_{14}, v_{15}^d, v_5$ $\{\neg v_5, \neg v_9, v_{11}\} \leftarrow$ USESAVEDTRAIL $\mathcal{T}_{save}: \quad v_5^d, v_6, v_7^d, v_8, v_9^d, v_{10}, v_{11}$ $\mathcal{T}: \quad v_1^d, v_2, v_3^d, v_4, \neg v_{12}, v_7, v_9, v_{13}^d, \neg v_{14}, v_{15}^d, v_5, v_6, v_8, v_{10}$
$(3, \{\neg v_3, v_{14}, \neg v_{15}\}, \neg v_{15}) \leftarrow$ ANALYSECONFLICT$(\{\neg v_5, \neg v_9, v_{11}\}, \mathcal{T})$
BACKTRACK(3) $\mathcal{T}_{save}: \quad v_5^d, v_6, v_7^d, v_8, v_9^d, v_{10}, v_{11}$ $\mathcal{T}: \quad v_1^d, v_2, v_3^d, v_4, \neg v_{12}, v_7, v_9, v_{13}^d, \neg v_{14}$ $\mathcal{F}: \quad \mathcal{F}_0 \cup \{\neg v_1, \neg v_3, \neg v_{13}\} \cup \{\neg v_3, v_{14}, \neg v_{15}\}$ $\mathcal{T}: \quad v_1^d, v_2, v_3^d, v_4, \neg v_{12}, v_7, v_9, v_{13}^d, \neg v_{14}, \neg v_{15}$
$\{v_2, v_{15}, \neg v_{16}\} \leftarrow$ PROPAGATE() $\mathcal{T}: \quad v_1^d, v_2, v_3^d, v_4, \neg v_{12}, v_7, v_9, v_{13}^d, \neg v_{14}, \neg v_{15}, v_{16}$
$(1, \{v_1, v_2, \neg v_{16}\}, \neg v_{16}) \leftarrow$ ANALYSECONFLICT$(\{v_2, v_{15}, \neg v_{16}\}, \mathcal{T})$
BACKTRACK(1) $\mathcal{T}_{save}: \varnothing$ $\mathcal{T}_{save}: \quad v_3^d, v_4, \neg v_{12}, v_7, v_9$ $\mathcal{T}: \quad v_1^d, v_2$ $\mathcal{F}: \quad \mathcal{F}_0 \cup \{\neg v_1, \neg v_3, \neg v_{13}\} \cup \{\neg v_3, v_{14}, \neg v_{15}\} \cup \{v_1, v_2, \neg v_{16}\}$ $\mathcal{T}: \quad v_1^d, v_2, \neg v_{16}$
$\varnothing \leftarrow$ PROPAGATE() $\mathcal{T}: \quad v_1^d, v_2, \neg v_{16}, v_{15}$
$(0, \{v_{16}\}, v_{16}) \leftarrow$ ANALYSECONFLICT$(\{\neg v_2, \neg v_{15}, v_{16}\}, \mathcal{T})$
BACKTRACK(0) $\mathcal{T}_{save}: \varnothing$
...

Табела 3.2: Приказ дела извршавања из примера 3.4.1

Листа $\mathcal{T} + \mathcal{T}_{save}$, добијена непосредно након првог повратног скока, има својство сагласности разлога (теорема 2 – случај 3). Претпоставимо сада да у неком тренутку листа $\mathcal{T} + \mathcal{T}_{save}$ има сагласне разлоге и докажимо да сагласност разлога остаје да важи након произвољне операције процедуре. Приметимо да важи $\mathcal{T} = \mathcal{T}_{back} + \mathcal{T}_{new}$, где је $\mathcal{T}_{back}$ валуација генерисана непосредно након последњег повратног скока и $\mathcal{T}_{new}$ део валуације изграђен након истог. Довољно је прилагодити само случајеве 2 и 3 доказа теореме 2 новом модификованом облику:

2. Коришћењем UseSavedTrail функције, решавач прескаче сачуване литерале или их додаје у тренутну валуацију $\mathcal{T}$ искључиво након обраде свих претходних сачуваних литерала. У оба случају, сви претходно обрађени литерали припадају тренутној валуацији $\mathcal{T}$. Због тога, листа $\mathcal{T} + \mathcal{T}_{save}$ или остаје непромењена или се тренутној валуацији додаје пивот листе $\mathcal{T}_{save}$ који је недефинисан у $\mathcal{T}$. У оба случаја, важи сагласност разлога за генерисану листу $\mathcal{T}' + \mathcal{T}_{save}$ (став 1 – случај 2 и став 3). На крају потпуног безконфликтног нивоа, уклањају се посећени литерали са почетка $\mathcal{T}_{save}$. Како такви литерали припадају тренутној валуацији $\mathcal{T}'$, добијена листа $\mathcal{T}' + \mathcal{T}'_{save}$ задржава својство сагласности разлога (став 1 – случај 1).

3. Уколико имамо конфликт на нивоу L_{deep} који је једнак нивоу последњег повратног скока L^{old}_{back}, непосредно пред надовезивање сачуваних валуација, унутар Backtrack функције, брише се $\mathcal{T}_{save}$. Одатле важи сагласност разлога за $\mathcal{T}' + \mathcal{T}'_{save}$, коју генерише решавач непосредно након Backtrack функције (теорема 2 – случај 3). У супротном, уколико имамо конфликт на нивоу L_{deep}, али имали смо нових претпостављених литерала након последњег повратног скока, тада знамо да је део валуације $\mathcal{T}_{new}$ генерисан искључиво додавањем литерала. С обзиром да измене у $\mathcal{T}_{save}$ на конфликтном нивоу L_{deep} нису потврђене, листа $\mathcal{T}_{save}$ остаје иста каква је била непосредно пре почетка нивоа L_{deep}. Како је тада, за претходно генерисану листу $\mathcal{T}[[0...L_{deep} - 1]] + \mathcal{T}_{save}$ важило својство сагласности разлога, следи да сагласност разлога важи и за $\mathcal{T}' + \mathcal{T}'_{save} = \mathcal{T}[[0...L_{back}]] + (\mathcal{T}[[L_{back} + 1...L_{deep} - 1]] + \mathcal{T}_{save}) = (\mathcal{T}[[0...L_{back}]] + \mathcal{T}[[L_{back} + 1...L_{deep} - 1]]) + \mathcal{T}_{save}$ (став 1 – случај 2) добијену непосредно након Backtrack функције.

Одавде следи трврђење у целости, јер се остали случајеви могу доказати на исти начин као и у теореми 2. □

Предувид конфликта

Предувид конфликта (енгл. *conflict lookahead*) је модификација која подразумева употребу листе $\mathcal{T}_{save}$ у сврху ранијег откривања конфликта и учења одговарајуће клаузе. Наиме, ако у листи сачуваних литерала $\mathcal{T}_{save}$ постоји пропагирани литерал l_{save} који је нетачан у тренутној валуацији $\mathcal{T}$, решавач ће обрадом свих његових претходника на крају открити конфликт $reason_{save}(l_{save})$.

Имајући ово у виду, код ове модификације, решавач на крају сваког бесконфликтног нивоа проверава наредних k нивоа листе $\mathcal{T}_{save}$, у потрази за тренутно нетачним пропагираним литералом сачуване валуације. Решавач рачуна унапред колико би морао претпостављених литерала l^d_{save} из $\mathcal{T}_{save}$ да изабере као следеће претпостављене литерале валуације $\mathcal{T}$, да би сигурно изазвао конфликт. Како тачни претпостављени литерали l^d_{save} већ припадају тренутној валуацији, они се игноришу и не броје се при овом прорачуну. Слично, уколико при провери решавач наиђе на претпостављени литерали l^d_{save} који је нетачан, употреба литерала који за њим следе у листи сачуваних литерала није могућа, па се претрага заршава неуспешно.

Са друге стране, ако решавач успешно пронађе пропагирани литерал l_{save} који је нетачан у $\mathcal{T}$, а налази се у наредних k нивоа листе $\mathcal{T}_{save}$, као први следећи претпостављени литерал претраге (и валуације $\mathcal{T}$) узима се управо $\mathcal{T}_{save}[0]$. Приметимо да је по комплетирању сваког бесконфликтног нивоа валуације $\mathcal{T}$, литерал $\mathcal{T}_{save}[0]$, ако постоји, увек претпостављени литерал у $\mathcal{T}_{save}$, док у валуацији $\mathcal{T}$ може бити једино или нетачан или недефинисан. Заиста, у листу $\mathcal{T}_{save}$ функција Backtrack чува увек комплетне нивое, док у оквиру UseSavedTrail функције, ако је претпостављени литерал листе $\mathcal{T}_{save}$ тачан у $\mathcal{T}$, тада ће обрада његовог нивоа у $\mathcal{T}_{save}$ бити потпуна, осим уколико је прекине откривени конфликт, при чему остаје неизмењена.

Приметимо да ова модификација утиче једино на избор претпостављених литерала (њена функционалност се може имплементирати у оквиру функције PickBranchingVariable). Како начин доношења претпоставки унутар CDCL алгоритма не утиче на коректност алгоритма, алгоритам остаје коректан и уз ову модификацију.

Филтрирање разлога по квалитету

Савремени SAT решавачи користе разне стратегије за поједностављање научене клаузе C_{1-UIP}, генерисане током анализе конфликта. Основна мотивација за поједностављање клауза је чињеница да је корист клаузе унутар претраге обично обрнуто сразмерна њеној величини. Заиста, празне клаузе су незадовољиве, па се претрага може одмах зауставити. Клаузе са само једним литералом l, одређују нужну вредност његове променљиве, чиме је елиминишу у потпуности из формуле, и тиме поједностављају претрагу. Клауза која се састоји из два литерала одсеца једну четвртину простора претраге, јер забрањује једну комбинацију вредности своја два литерала. Сличним разматрањем, видимо да се даљим повећањем величине клаузе, њен утицај на претрагу смањује.

Ипак, величина клаузе није једина мера њеног квалитета. Наиме, често се као мера квалитета (у обрнуто пропорционалном смислу) користи и *lbd* (енгл. *Literals block distance*) клаузе C, коју дефинишемо као број међусобно различитих нивоа литерала који се појављују унутар клаузе C ($lbd(C) = |\{decLvl(l)|l \in C\}|$) [1]. Приметимо да за сваку клаузу C важи $lbd(C) < |C|$.

Како клауза C_{1-UIP} зависи од разлога литерала који се користе у анализи конфликта, пожељно је и да сами разлози пропагација такође буду што квалитетније клаузе. Зато приликом обраде недефинисаног пропагираног литерала $l_{save} \in \mathcal{T}_{save}$, уколико је његов сачувани разлог $reason_{save}(l_{save})$ лошег квалитета, обраду листе $\mathcal{T}_{save}$ можемо зауставити у нади да ће приликом обраде листе $watchlist(l)$ за неки од новододатих литерала l валуације $\mathcal{T}$ бити пронађен бољи разлог за дати литерал l_{save}. Наиме, имплементација ове модификације подразумева додавање следеће две линије у алгоритам 3:

```
35: ...
36: if LowReasonQuality(C) then
37:     return ∅
38: ...
```

Коректност ове модификоване верзије алгоритма следи директно из својстава CDCL алгоритма који преузима процес пропагације литерала у случају наиласка на неквалитетан сачувани разлог. Притом, сачувани пропагирани литерал са разлогом ниског квалитета у међувремену се може додати у валуацију од стране CDCL алгоритма, чиме се омогућава даља обраде листе

сачуваних литерала $\mathcal{T}_{save}$. Ипак, треба имати у виду да накнадно пронађени разлог пропагације у оквиру CDCL алгоритма може бити идентичан или лошији од сачуваног, у ком случају је непотребно заустављено коришћење листе $\mathcal{T}_{save}$.

Глава 4

Унапређење технике чувања валуације

У претходној глави представљен је основни облик технике чувања валуације, заједно са њене три модификације предложене у оригиналном раду [5]. Унапређење које предлажемо у овој глави, а које је оригинални допринос овог рада, ослања се на прву од ове три модификације, тј. надовезивање сачуваних валуација. Иако је суштина ове модификације једноставна, она представља природно проширење основног алгоритма. Ипак приликом њене имплементације треба бити изузетно пажљив. У наставку ове главе детаљно приказујемо предложена унапређења, као и доказ коректности алгоритма који укључује наша унапређења.

4.1 Приказ предложеног унапређења

У претходној глави, приликом приказа технике надовезивања сачуваних валуација, представљен је и случај када то надовезивање не мора бити безбедно, јер потенцијално нарушава сагласност разлога. Једноставно решење које подразумева брисање претходне сачуване валуације увек у случају конфликта на нивоу последњег повратног скока, иако задржава коректност алгоритма, може довести до тога да се сачувана валуација брише и када то није неопходно. Управо пример 3.4.1 дат у претходној глави савршено илуструје овај проблем. Може се приметити да само последњи повратни скок заиста захтева брисање претходне сачуване валуације, док би у осталим случајевима својство сагласности разлога листе $\mathcal{T}_{save}$ добијене надовезивањем следило из

сагласности разлога претходног стања листе.

У наставку, од посебне важности су повратни скокови који не модификују сачувану листу $\mathcal{T}_{save}$, односно они чији је ниво повратног скока за један мањи од тренутног. Такви повратни скокови не додају нове литерале у $\mathcal{T}_{save}$ унутар функције Backtrack, јер је валуација која се надовезује $\mathcal{T}[[L_{back}+1...L_{deep}-1]]$ у том случају празна. У даљем тексту, такве повратне скокове називамо *некритичним повратним скоковима*, док остале повратне скокове називамо *критичним повратним скоковима*. У основи, потребно је пратити само ниво последњег критичног повратног скока. У случају конфликта на таквом нивоу, он се изоставља из новог префикса сачуване валуације добијене надовезивањем дела тренутне валуације $\mathcal{T}[[L_{back}+1...L_{deep}-1]]$ и $\mathcal{T}_{save}$, чиме се може угрозити својство сагласности разлога за $\mathcal{T}_{save}$. Да бисмо избегли овај већ познат проблем, бришемо садржај листе $\mathcal{T}_{save}$ непосредно пред надовезивање. Такве нивое, на којима у случају наиласка на конфликт бришемо садржај листе $\mathcal{T}_{save}$, називамо *критичним нивоима*[1]. Како се након некритичног повратног скока нови литерали не додају у сачувану валуацију, вредност претходног критичног нивоа није неопходно мењати. Прецизније, наш предлог подразумева измену линија 2-4. алгоритма 3 прве модификације претходне главе:

```
1: Backtrack(L_back)
2:     if L_deep = L_crit then
3:         T_save ← ∅      ▷ Надовезивање не одржава нужно сагласност разлога
4:     if L_deep − L_back > 1 then
5:         L_crit ← L_back      ▷ Памтимо ниво новог критичног повратног скока
6:     T_save ← T[[L_back + 1...L_deep − 1]] + T_save
7:     ...
```

На овај начин смањили смо број непотребних одбацивања листе $\mathcal{T}_{save}$. Међутим, унета измена није довољна да би се задржала коректност модификованог облика алгоритма. Наиме, Backtrack функција није једина која утиче на $\mathcal{T}_{save}$. Након сваког неконфликтног нивоа, све непотврђене измене листе $\mathcal{T}_{save}$ се потврђују. Уколико желимо да наше унапређење одржава својство сагласности разлога за $\mathcal{T}+\mathcal{T}_{save}$, неопходно је да и овакве нивое сматрамо критичним. Заиста, иако претрага напредује започињањем нових нивоа, она

[1]Приметимо да у основном облику модификације надовезивања сачуваних валуација, сваки ниво повратког скока се сматра критичним нивоом.

се може вратити на поменути ниво правећи вишеструке некритичне повратне скокове, при чему се стара вредност нивоа последњег критичног повратног скока не мења. Како су потврђене измене сада део поменутог нивоа валуације $\mathcal{T}$, слично претходном проблему, услед одбацивања тренутног нивоа при наиласку на конфликт, постоји могућност да се угрози сагласност сачуваних разлога преосталих литерала из $\mathcal{T}_{save}$. Из свега наведеног, природно следи да је потребно прогласити критичним и неконфликтне потпуне нивое који су успешно модификовали $\mathcal{T}_{save}$. Другим речима, предлаже се додавање нове две линије између линија 15-16. алгоритма 3 прве модификације (уз задржавање горееуведене измене):

```
15: ...
16: if idx_pivot > 0 then
17:     L_crit ← L_deep  ▷ Памтимо последњи ниво са потврђеним изменама T_save
18:     while idx_pivot > 0 do
19:         ...
20: ...
```

Приметимо да и са овим изменама основна идеја модификованог алгоритма није знатно промењена. При сваком (критичном) повратном скоку вршимо надовезивање сачуваних валуација, при чему старе сачуване литерале првобитно одбацујемо акко је конфликт пронађен на критичном нивоу. Разлика је једино у томе што је сада дефиниција критичног нивоа пажљивије одабрана, тако да се смањи учесталост непотребног брисања сачуване валуације.

4.2 Доказ коректности

Да бисмо доказали коректност, неопходно је прилагодити само теорему 3 основне модификације описане у поглављу 3.4 која укључује надовезивање сачуваних валуација. Ипак, најпре ћемо увести појам *критичне листе*, а затим ћемо доказати ставове који ће нам бити од значаја при доказу главне теореме. У даљем тексту, под *унапређеном техником чувања валуације* сматраћемо модификовану технику чувања валуације која укључује надовезивање сачуваних валуација и употребу критичних нивоа.

Под *критичном листом*, у одређеном тренутку (стању) претраге, сматрамо $\mathcal{T} + \mathcal{T}_{save}$, тако да важи $L_{deep} = L_{crit}$, односно најдубљи ниво валуације

$\mathcal{T}$ уједно је и критични.

Став 3. Ако је $\mathcal{T} + \mathcal{T}_{save}$ генерисана унапређеном техником чувања валуације и важи $\mathcal{T}_{save} \neq \varnothing$, онда важи $\mathcal{T} = \mathcal{T}_{crit} + \mathcal{T}_{new}$ и $\mathcal{T}_{save} = \mathcal{T}_{save}^{crit}$, где је $\mathcal{T}_{crit} + \mathcal{T}_{save}^{crit}$ последња генерисана критична листа, а $\mathcal{T}_{new}$ произвољна листа непротивречних литерала.

Доказ. Нека је $\mathcal{T} + \mathcal{T}_{save}$ тренутно генерисана листа са најдубљим нивоом L_{deep}, тако да важи $\mathcal{T}_{save} \neq \varnothing$. Уколико је $\mathcal{T}$ критична валуација, тврђење тривијално важи. У супротном, претпоставимо да је $\mathcal{T}_{crit} + \mathcal{T}_{save}^{crit}$ последња генерисана критична листа, тако да је $\mathcal{T} \neq \mathcal{T}_{crit}$. Валуација $\mathcal{T}$ је очигледно добијена применом накнадних операција над $\mathcal{T}_{crit}$. Да би за свако $\mathcal{T}_{new}$ важила неједнакост $\mathcal{T} \neq \mathcal{T}_{crit} + \mathcal{T}_{new}$ неопходно је у неком тренутку отклонити литерале из валуације $\mathcal{T}_{crit}$. То је могуће искључиво некритичним повратним скоковима, услед претпоставке да је $\mathcal{T}_{crit} + \mathcal{T}_{save}^{crit}$ последња генерисана критична листа. Због редног отклањања нивоа, немогуће је прескочити L_{crit} критични ниво валуације (који се није мењао), услед чега се брише $\mathcal{T}_{save}$. Међутим $\mathcal{T}_{save}$ мора остати празна, јер једино накнадни критични повратни скок може у њу додати литерале. Одатле контрадикцијом закључујемо да важи $\mathcal{T} = \mathcal{T}_{crit} + \mathcal{T}_{new}$ за неку листу непротивречних литерала $\mathcal{T}_{new}$. Приметимо да мора да важи и $\mathcal{T}_{save} = \mathcal{T}_{save}^{crit}$, јер у супротном решавач би морао да модификује листу $\mathcal{T}_{save}^{crit}$. То је искључиво могуће критичним повратним скоковима, потврђивањем измена $\mathcal{T}_{save}^{crit}$ или конфликтом на критичном нивоу. У сва три случаја би морале бити нарушене претпоставке о валуацији $\mathcal{T}_{crit} + \mathcal{T}_{save}^{crit}$ као последњој критичној листи процедуре. Дакле, закључујемо да важи и $\mathcal{T} = \mathcal{T}_{crit} + \mathcal{T}_{new}$ (за неко $\mathcal{T}_{new}$) и $\mathcal{T}_{save} = \mathcal{T}_{save}^{crit}$. □

Теорема 4. Ако је $\mathcal{T} + \mathcal{T}_{save}$ генерисана унапређеном техником чувања валуације, онда $\mathcal{T} + \mathcal{T}_{save}$ има својство сагласности разлога.

Доказ. У почетку процедуре, не постоје сачувани литерали, због чега у прво време валуацију решавач генерише искључиво операцијама CDCL алгоритма. Листа $\mathcal{T} + \mathcal{T}_{save}$, добијена непосредно након првог (критичног) повратног скока, има својство сагласности разлога (теорема 2 – случај 3). Претпоставимо сада да у неком тренутку листа $\mathcal{T} + \mathcal{T}_{save}$ има сагласне разлоге и докажимо да сагласност разлога остаје да важи након примене произвољне операције процедуре. Уколико важи $\mathcal{T}_{save} = \varnothing$, тврђење очигледно важи. Ако

је $\mathcal{T}_{save} \neq \varnothing$, приметимо да тада важи $\mathcal{T} = \mathcal{T}_{crit} + \mathcal{T}_{new}$ и $\mathcal{T}_{save} = \mathcal{T}_{save}^{crit}$, где је $\mathcal{T}_{crit} + \mathcal{T}_{save}^{crit}$ последња генерисана критична листа и $\mathcal{T}_{new}$ произвољна листа непротивречних литерала (став 3). Довољно је прилагодити само случај 3 доказа теореме 3:

3. Уколико је ниво L_{deep} уједно и ниво L_{crit}, непосредно пред надовезивање сачуваних листи унутар Backtrack функције, брише се $\mathcal{T}_{save}$. Одатле важи и сагласност разлога за листу $\mathcal{T}' + \mathcal{T}'_{save}$, генерисану непосредно након Backtrack функције (теорема 2 – случај 3). Уколико је ниво L_{deep} конфликтан, али је притом $L_{deep} \neq L_{crit}$, листа $\mathcal{T}_{new}$ садржи барем један претпостављени литерал. Валуација $\mathcal{T}[[0...L_{deep} - 1]]$ има сагласне разлоге као префикс листе $\mathcal{T}+\mathcal{T}_{save}$ са сагласним разлозима. Специјално, измене листи $\mathcal{T}_{save} = \mathcal{T}_{save}^{crit}$ на тренутном L_{deep} нивоу нису потврђене, одакле следи да она остаје непромењена. Како важи сагласност разлога за критичну листу $\mathcal{T}_{crit} + \mathcal{T}_{save}^{crit}$, тако важи и за $\mathcal{T}' + \mathcal{T}'_{save} = \mathcal{T}[[0...L_{back}]] + (\mathcal{T}[[L_{back}+1...L_{deep}-1]]+\mathcal{T}_{save}^{crit}) = (\mathcal{T}_{crit}+\mathcal{T}[[L_{crit}+1...L_{deep}-1]])+\mathcal{T}_{save}^{crit}$ (став 1 – случајеви 2 и 3) добијену непосредно након Backtrack функције.

Одавде следи трврђење у целости, јер се остали случајеви могу доказати на исти начин као и у теореми 3. □

Глава 5

Имплементација и евалуација

У првом поглављу ове главе приказујемо неке специфичности имплементације технике чувања валуације и њених модификација у оквиру MiniSat SAT решавача. Модификовани решавач, под називом `MiniSat-trail`, доступан је на веб адреси `https://github.com/david-scepanovic/minisat-trail.git`. У другом поглављу представљени су резултати евалуације различитих конфигурација решавача `MiniSat-trail`, као и оригиналног решавача MiniSat, зарад поређења.

5.1 Имплементација

Као основа за решавач `MiniSat-trail` изабрана је верзија 2.2.0 решавача MiniSat[1]. Од модификација које имплементирају технику чувања валуације најбитније су оне унутар фајлова `core/Solver.h` и `core/Solver.cc`.

Размотримо неке имплементационе детаље решавача `MiniSat-trail`:

- Логика додавања литерала у листу $\mathcal{T}_{save}$ и отклањања литерала из ње имплементирана је унутар помоћне класе `TrailSaver`. Поред осталих функционалности, она управља пивотом листе $\mathcal{T}_{save}$, а садржи и метод `clean`, који се користи за повремено „чишћење" листе $\mathcal{T}_{save}$.

- Унутар класе `Solver`, методе које су додате у циљу интеграције технике чувања валуације су `saveTrail` и `useSavedTrail`, док метода `lookAhead` имплементира предувид конфликта. Треба такође истаћи да су неке већ

[1] `https://github.com/niklasso/minisat/releases/tag/releases`

постојеће методе, попут `pickBranchLit`, `propagate` и `search`, у одређеној мери модификоване у циљу имплементације технике.

- Имена променљивих су најчешће изабрана тако да почињу префиксом `ts` зарад лакшег препознавања. Имплементација укључује и додатне променљиве за скупљање статистичких резултата. Променљива `ts_crit_lvl` чува вредност последњег дефинисаног критичног нивоа.

- Додате су 4 нове опције командне линије, које подешавају конфигурацију претраге при покретању решавача: `ts-cap-fact`, `ts-max-look`, `ts-max-csize` и `ts-max-clbd`. Опција `ts-cap-fact` контролише капацитет листе $\mathcal{T}_{save}$, `ts-max-look` контролише број сачуваних нивоа које решавач претражује у циљу ранијег откривања конфликта, док `ts-max-csize` и `ts-max-clbd` контролишу избор одговарајућег критеријума при филтрирању сачуваних разлога. Детаљан опис ових опција приказује се опцијом `--help` при покретању[2].

- Подешавање конфигурације решавача може бити извршено и укључивањем претпроцесорских ознака: TS_BRUTE_CRIT_LVLS, TS_CLBD и TS_SMART_CLEANS. Дефинисањем TS_BRUTE_CRIT_LVLS макроа искључује се оригинално унапређење представљено у претходној глави. Макро TS_CLBD омогућава филтрирање сачуваних разлога по њиховом *lbd*-у, уместо по њиховој величини, док TS_SMART_CLEANS омогућава темељније „чишћење" сачуваних литерала, засновано на употреби критичних нивоа.

У имплементацији технике чувања валуације, својом комплексношћу, посебно се истичу функције `useSavedTrail` и `propagate`. Имплементација ових метода је нарочито осетљива услед одређених специфичности решавача MiniSat које описујемо у наставку.

У глави 2, зарад једноставнијег излагања, клауза је представљена као скуп литерала. Ипак, у савременим SAT решавачима, много је чешћа реализација клауза у облику листе литерала. Посебно, постоје специфичности Minisat SAT решавача везане за поредак литерала које могу битно утицати на рад решавача. Наиме, у решавачу Minisat увек важи да уколико је из клаузе C

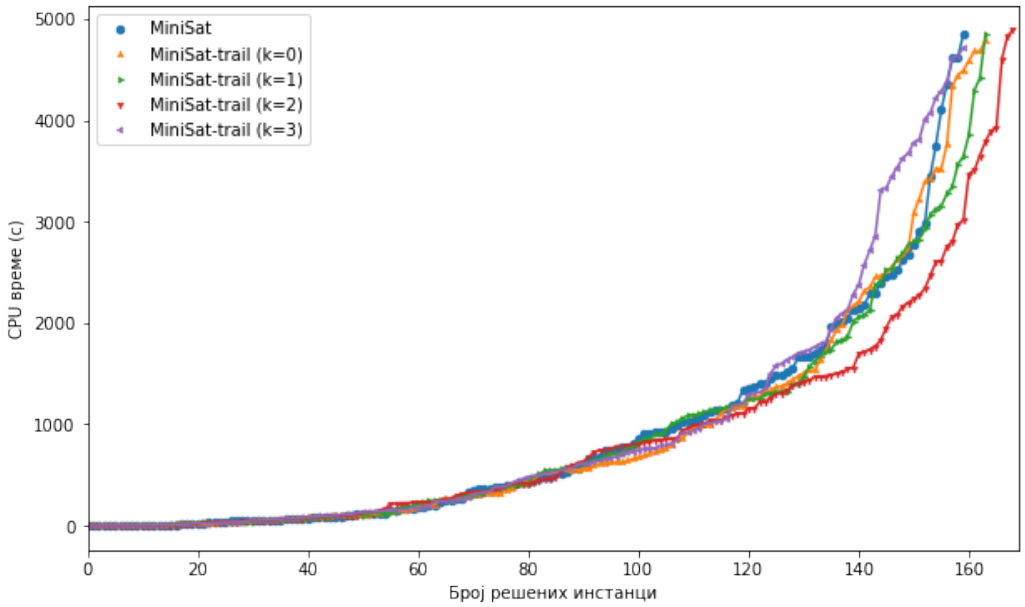
[2]За више информација о самом покретању програма посетити документацију пројекта, као и званичну веб-страницу решавача MiniSat: `http://minisat.se/`.

извршена пропагација литерала l, тада се литерал l налази на почетној позицији унутар клаузе C. Слично, посматрани литерали l' и l'' клаузе C се увек налазе на две почетне позиције у клаузи C. Последица свега наведеног је да је пре додавања сачуваног пивота l_{save} у тренутну валуацију, неопходно прво преместити l_{save} на почетну позицију унутар клаузе $reason_{save}(l_{save})$. Уколико се због тога, један од посматраних литерала, на пример l', измести са прве две почетне позиције клаузе $reason_{save}(l_{save})$, неопходно је и отклонити њу из $watchlist(\bar{l'})$ листе посматраних клауза за $\bar{l'}$.

Директно отклањање сачуваног разлога из одговарајуће листе клауза учинило би додавање сачуваних литерала у валуацију веома скупом операцијом и на тај начин угрозило смисао саме технике чувања валуације. Алтернативни приступ, који је искоришћен у оквиру наше имплементације, је да се ове клаузе не избацују одмах, већ да се само означе за касније брисање приликом обраде листе $watchlist(\bar{l'})$ у оквиру уобичајене пропагације литерала у CDCL алгоритму. Приметимо да сачувана клауза у тренутку обраде пивота још увек није посећена, иначе би он већ био додат у валуацију као јединични литерал клаузе. Такође, приликом обраде листе посматраних литерала, клаузе које су означене за брисање се лако препознају по томе што посматрани литерал који одговара тој листи није ни на нултој ни на првој позицији у клаузи. Одатле, њено отклањање може се постићи на исти начин на који CDCL иначе отклања посматране клаузе из одговарајуће листе. Ипак, уколико пропагацију прекине конфликт, може доћи до тога да се листе клауза у којима је нарушен уобичајени интегритет на горенаведени начин уопште не обраде. Због тога треба обезбедити и механизам који накнадно отклања све означене клаузе које нису успешно отклоњене. У оквиру наше имплементације, уколико након наиласка на конфликт такве клаузе постоје, њихово директно отклањање извршава се одмах. Срећом, испостави се да ову потенцијално скупу процедуру у пракси није нужно обављати често, што се може закључити из резултата евалуације у наредном поглављу.

5.2 Евалуација

За евалуацију представљених техника искоришћен је рачунар са 16GB RAM-а и AMD Ryzen 5 3600 процесором са 6 језгара (12 нити) и фреквенцијом 3.6GHz (4.2GHz Max Boost). Корпус проблема коришћен за евалуацију чини



Слика 5.1: „Cactus" приказ резултата евалуације

Application („MiniSat Hack track") група од 300 инстанци, са званичног SAT Competition 2014 такмичења 2014. године[3]. Приликом евалуације, није коришћена могућност извоза резолуцијског доказа незадовољивости, ради провере његове исправности помоћу одговарајућих алата. Ипак, ова могућност је искоришћена накнадно, на одређеном броју случајно изабраних инстанци, како бисмо повећали поверење у исправност наше имплементације (коришћен је алат *Drat-trim*[4]). Приликом евалуације, временско ограничење било је 5,000 секунди по инстанци, док је меморијско ограничење било имплицитно одређено хардверским ограничењима самог система.

На графикону 5.1 представљени су резултати евалуације немодификованог решавача MiniSat, као и решавача `MiniSat-trail` за различите вредности k, где је k број сачуваних нивоа који се претражују у циљу ранијег откривања конфликта. У наставку, ако није наглашено другачије, подразумевамо да решавач `MiniSat-trail` користи унапређену технику чувања валуације, приказану у претходној глави.

[3]Група инстанци може се преузети са следеће веб адресе `http://www.satcompetition.org/2014/index.shtml`. Поменуто такмичење представља последње такмичење са категоријом „MiniSat Hack track" чији циљ је унапређење MiniSat SAT решавача. Више о томе на приложеној страници.

[4]`https://www.cs.utexas.edu/~marijn/drat-trim/`

	MiniSat	MiniSat-trail	MiniSat-trail	MiniSat-trail	MiniSat-trail
k	\	0	1	2	3
sat	91	94	94	98	91
unsat	69	70	70	71	69
Σ	160	164	164	169	160
PAR-2	5,133	5,036	5,022	4,870	5,162

Табела 5.1: Резултати евалуације

У табели 5.1 приказан је број решених инстанци, као и просечно време извршавања. Уколико се за неку инстанцу извршавање алгоритма прекине након 5,000 секунди, време потпуног извршавања решавача на тој инстанци остаје непознато. Због тога се у тим случајевима користи PAR-2 оцена времена која такве инстанце „кажњава", тако да се као коначно време извршавања такве инстанце узима $2 \cdot 5,000$ секунди. Укупна PAR-2 оцена времена целог корпуса представља просек свих времена извршавања појединачних инстанци укључујући и инстанце које су прекинуте. За инстанце које су прекинуте услед недостатка меморије се такође користи иста PAR-2 временска оцена.

На основу приказане табеле и графикона, видимо да се за $k < 3$ `MiniSat-trail` показао успешнијим од немодификованог решавача MiniSat који је успео да реши укупно 160 инстанци са датим временским ограничењем. Већ за $k = 0$, без модификације предувида конфликта, `MiniSat-trail` је успео да реши 4 инстанце више. С друге стране, за $k = 1$ побољшала се само PAR-2 оцена. Ипак, са 169 решених инстанци, најуспешнијим се показао решавач `MiniSat-trail` за $k = 2$ вредност[5]. Употреба већих вредности k учинило је претрагу значајно неуспешнијом, што поновно потврђује да превелико одузимање контроле решавачу над доношењем претпоставки утиче лоше на саму претрагу. Посебно проблематична појава за предувид конфликта за велику вредност k је то што нека од тих k претпоставки у току претраживања сачуваних нивоа може постати нетачна, што зауставља даљи процес наметања претпоставки. При томе је претрага већ усмерена у неком смеру у коме је принуђена да напредује до следећег конфликта. Просечно време извршавања (у складу са PAR-2 методологијом) такође потврђује да вредност параметра

[5]Интересантно је да је и у оригиналном раду [5] за ову вредност параметра k постигнут најбољи резултат.

	MiniSat-trail	
k	2	3
конфликата	4,362,388	4,754,230
претпостављаних литерала	14,533,701	17,056,431
пропагација литерала	1,716,072,214	1,636,214,827
сачуваних валуација	722,231	905,310
сачуваних литерала	277,540,663	269,149,943
сачуваних претпостављених литерала	10,168,443	12,299,195
сачуваних конфликата	11,131	15,339
пропагација сачуваног пивота	91,413,714	88,316,942
прескакања сачуваног пивота	44,558,923	41,017,728
накнадних отклањања клауза	483	597
претрага сачуване валуације	6,502,363	8,185,501
наметнутих претпостављених литерала	12,915	27,898

Табела 5.2: Темељна евалуација успешно решених инстанци

$k = 2$ даје најбоље резултате.

Интересантно је осврнути се и на друге статистичке податке приказане у табели 5.2[6], који нам могу бити од помоћи у разумевању рада решавача `MiniSat-trail`. Услед недостатка простора, приказани су подаци искључиво за инстанце које је успешно решио решавач `MiniSat-trail`, за $k = 2$ и $k = 3$.

Приметимо да је удео пропагација из сачуване валуације изузетно мали у односу на укупни број пропагација. Ипак, по броју пута када је сачувани пивот приликом обраде већ припадао валуацији видимо да је сачувана валуација чешће од користи него што није. С друге стране, наилажење на сачуване конфликте је знатно ређе. Додатно, приметимо да је број наметнутих претпостављених литерала веома низак у односу на број претрага сачуване валуације. Ипак и поред тога, утицај саме модификације на рад решавача је значајан.

У претходном поглављу, представљен је један имплементациони детаљ везан за накнадно уклањање посматраних клауза. На основу резултата приказаних у табели, може се закључити да решење представљено у претходном поглављу нема негативан утицај на перформансе решавача, с обзиром да се ова, потенцијално скупа процедура, позива релативно ретко.

[6]Број пропагираних литерала приказан у табели 5.2 је једна од метрика оригиналног решавача MiniSat и заправо представља број листи $watchlist(l)$ литерала $l \in \mathcal{T}$ које је решавач успешно посетио. Дакле, број заиста пропагираних литерала је вероватно већи од приказаног.

Најзад, у претходом излагању је поменуто да је потребно повремено „чистити" сачувану валуацију од дупликата, како би се ограничио њен раст. Евалуација је показала да је ово чишћење у просеку потребно вршити 0 до 2 пута по инстанци, што нам говори да се приликом имплементације не морамо превише обазирати на ову изузетно ретку појаву.

Модификација заснована на филтрирању разлога по квалитету показала се корисном над искоришћеним корпусом проблема само за једну од испробаних конфигурација. Као критеријум филтрирања коришћене су, као и у оригиналном раду [5], величина клаузе и *lbd* вредност клаузе. Слично резултатима пријављеним у оригиналном раду, при евалауцији решавача MapleSat (који интерно користи MiniSat као језгро[7]), филтрирање разлога се и у решавачу `MiniSat-trail` показало неефикасним за већину испробаних конфигурација. Наиме, једини изузетак представља конфигурација којом решавач одбацује све сачуване разлоге чија је *lbd* вредност већа од 2. Иако је са таквим критеријумом, решавач `Minisat-trail` успешно решио исто 169 инстанци, постигао је бољу PAR-2 оцену од 4,848 секунди. Од додатних података, ову конфигурацију очекивано истиче релативно слабији однос броја додавања литерала из сачуване валуације у тренутну (просечно 77,241,398 пута) и њиховог прескакања (просечно 57,533,065 пута). Последица тога је и знатно нижи просечни број накнакдних отклањања клауза (38 пута).

На крају, као можда најзначајнији део евалуације, две најефикасније добијене конфигурације решавача `MiniSat-trail` за $k = 2$ (без филтрирања разлога и са филтрирањем уз критеријум $lbd(reason_{save}(l_{save})) \leq 2$) искоришћене су за евалуацију оригиналног алгоритма чувања валуације без коришћења критичних нивоа. Другим речима, алгоритам је прилагођен тако да најбоље осликава оригинални приступ из рада [5]. Треба узети у обзир да овакав приступ евалуацији није најобјективнији с обзиром да параметри који су се показали оптималним за модификовану верзију технике надовезивања валуација са критичним нивоима не морају нужно бити оптимални за оригиналну технику. Ипак, због временских ограничења такав приступ је изабран са надом да ипак довољно добро осликава разлику ове две варијанте. Решавач `MiniSat-trail` без критичних нивоа за поменуту конфигурацију без

[7]Иако је у оригиналном раду евалуирана оригинална техника чувања валуације над решавачом MapleSat, само за решавач Cadical је омогућен јавни приступ. Оригинални рад не наводи ниједан начин да се јавно приступи имплементацији решавача MapleSat са техником чувања валуције.

филтрирања разлога решио је укупно 161 инстанцу, односно само једну инстанцу више од оригиналног решавача MiniSat, а чак 8 инстанци мање од варијанте са критичним нивоима. Очекивано, PAR-2 оцена је такође лошија, са просечним временом од 5,119 секунди. Међутим, укључивањем филтрирања разлога за $lbd(reason_{save}(l_{save})) \leq 2$, решавач `MiniSat-trail` је успешно решио чак 167 инстанци, само 2 инстанце мање од најуспешније конфигурације са критичним нивоима. Са друге стране, поред ниског просечног броја накнадних отклањања клауза (6 пута) и сличног односа додавања и прескакања сачуваних литерала, PAR-2 оцена од 4,982 секунди је знатно лошија. Свеукупно, резултати евалуације указују да последице непотребног одбацивања сачуване валуације у неким случајевима могу бити значајне, али и да у зависности од решавача и његове конфигурације, утицај увођења критичних нивоа на перформансе решавача може бити и знатно скромнији. Ипак, овај утицај је, свеукупно гледано, позитиван, а будући да увођење критичних нивоа представља природно проширење описане технике, њихово даље проучавање може представљати перспективан правац будућег рада.

Глава 6

Закључак

У овом раду детаљно је описана и анализирана техника чувања валуације, заједно са њеним модификацијама. У неопходној мери, представљен је и CDCL алгоритам, као основа за имплементацију ове технике. Посебна пажња посвећена је недостацима модификације која омогућава надовезивање сачуваних валуација. Због тога, као главни допринос рада, представљен је и предлог њеног унапређења. Он уводи појам критичних нивоа, чијим се пажљивим одабиром, може смањити број непотребних одбацивања сачуване валуације. Представљен је и доказ коректности како унапређене тако и оригиналне технике чувања валуације. Такође, представљени су и неки имплементациони детаљи нашег `MiniSat-trail` решавача у коме је имплементирана техника чувања валуације. Евалуација приказана у раду указује на то да унапређена верзија технике чувања валуације има потенцијал да пружи боље перформансе у односу на основну верзију технике. Поред перформанси, темељном евалуацијом измерени су и други статистички подаци који нам помажу да боље разумемо шта чини технику успешном.

Ипак, поред свега наведеног, остаје доста простора за даљи рад. Један правац истраживања би могао дати одговор на питање ли се критични нивои могу још прецизније дефинисати, као и на које све начине се могу додатно употребити и у којој мери могу утицати на перформансе других решавача. Слично, могуће је истражити да ли би унапређена техника чувања валуације интегрисана у C-bt алгоритам [10, 8], омогућила побољшање перформанси решавача. Најзад, уместо једноставних критеријума попут *lbd*-а и величине клаузе, можемо истражити и евалуирати модификацију филтрирања сачуваних разлога са сложенијим критеријумима филтрирања.

Библиографија

[1] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solvers. In *Twenty-first International Joint Conference on Artificial Intelligence*, 2009.

[2] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

[3] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[4] Martin Davis, George Logemann, and Donald Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[5] Randy Hickey and Fahiem Bacchus. Trail saving on backtrack. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 46–61. Springer, 2020.

[6] Filip Marić. Formalization and implementation of modern sat solvers. *Journal of Automated Reasoning*, 43(1):81–119, 2009.

[7] Joao Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning sat solvers. In *Handbook of satisfiability*, pages 133–182. ios Press, 2021.

[8] Sibylle Möhle and Armin Biere. Backing backtracking. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 250–266. Springer, 2019.

[9] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Annual ACM IEEE Design Automation Conference*, pages 530–535. ACM, 2001.

[10] Alexander Nadel and Vadim Ryvchin. Chronological backtracking. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 111–121. Springer, 2018.

[11] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL (T). *Journal of the ACM (JACM)*, 53(6):937–977, 2006.

Биографија аутора

Давид Шћепановић рођен је 16.09.1996. у Београду. У Београду је и одрастао, где је завршио основну школу „Стеван Дукић" 2011. године и друштвени смер Четрнаесте београдске гимназије 2015. године. Одмах по завршетку средње школе, уписао је основне академске студије математике на Математичком факултету у Београду, под програмом „Рачунарство и информатика". Дипломирао је у року 2019. године са просеком 9.00 и стекао звање Дипломирани математичар. Образовање је наставио одмах на истом факултету, где у октобру 2019. године уписује мастер студије математике, под истим програмом студија. Закључно са септембром 2020. године, положио је све испите на мастер студијама и тиме стекао услов за одбрану мастер рада.

Од 2021. године, званично је започео своју професионалну каријеру као софтверски инжењер.

Београд, 2021.

Давид Шћепановић