UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET


Dijana N. Alanović

# KOMBINOVANJE EGZAKTNIH ALGORITAMA I ALGORITAMA LOKALNE PRETRAGE U SAT I SMT REŠAVAČIMA

master rad

Beograd, 2025.

**Mentor**:

doc. dr Milan Banković, docent
Univerzitet u Beogradu, Matematički fakultet

**Članovi komisije**:

prof. dr Predrag janičić, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

prof. dr Vesna Marinković, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

**Datum odbrane**: _____

**Naslov master rada**: Kombinovanje egzaktnih algoritama i algoritama lokalne pretrage u SAT i SMT rešavačima

**Rezime**: U ovom radu se razmatraju mogućnosti kombinovanja egzaktnih algoritama i algoritama lokalne pretrage u okviru SAT i SMT rešavača. Jedna takva tehnika predstavljena je u radu Caija i Zanga na SAT konferenciji 2021. godine. Pored detaljnog prikaza ove tehnike u slučaju SAT rešavača, razmatrana je i adaptacija ove tehnike u kontekstu SMT rešavača. Tehnika je implementirana u okviru SMT rešavača ArgoSMT i otvorenog je koda. Evaluacija je data za više konfiguracija rešavača radi pronalaženja pogodne postavke promenljivih.

**Ključne reči**: SAT rešavač, SMT rešavač, algoritam lokalne pretrage, CDCL algotitam

# Sadržaj

# Glava 1

# Uvod

Problem iskazne zadovoljivosti (engl. *Boolean satisfiability problem*), u nastavku SAT, je fundamentalan problem u računarstvu, a algoritmi za njegovo rešavanje su zabeležili izuzetan napredak tokom prethodnih decenija. Problem podrazumeva ispitivanje da li za datu iskaznu formulu postoje vrednosti iskaznih promenljivih za koje je ta formula tačna. Formula je podrazumevano zadata u konjuktivnoj normalnoj formi (KNF). Na primer formula:

$$(x_1 \vee x_2) \wedge (\neg x_2) \wedge (\neg x_1 \vee x_3)$$

je tačna ako su promenljive $x_1$ i $x_3$ tačne, a promenljiva $x_2$ netačna. SAT je prvi problem za koji je dokazano da je NP-kompletan (Kuk-Levinova teorema [4, 7]). To znači da su svi problemi u klasi složenosti NP najviše jednako teški za rešavanje kao i SAT.

Alati koji implementiraju procedure za rešavanje SAT problema nazivaju se SAT rešavači. Moderni SAT rešavači su po pravilu zasnovani na CDCL algoritmu (engl. *Conflict-Driven Clause Learning*) [6]. Pored teorijskog značaja, njihova primena je posebno značajna u verifikaciji hardvera i softvera, gde omogućavaju otkrivanje grešaka u dizajnu digitalnih kola, procesora i kompajlera, čime se sprečavaju skupi propusti u industrijskoj proizvodnji. Pored toga, SAT rešavači se koriste u problemima planiranja i raspoređivanja, kao što su kreiranje školskih rasporeda, optimizacija proizvodnih procesa ili raspoređivanje avionskih letova. Zahvaljujući svojoj snazi, SAT rešavači nalaze primenu i u generisanju test slučajeva, dijagnostici sistema, pa čak i u rešavanju logičkih slagalica i igara (npr. sudoku), čime se demonstrira njihova široka primenljivost i fleksibilnost. Neki od poznatijih SAT

rešavača su MiniSAT[1], Glucose[2], Lingeling[3], CaDiCaL[4], itd.

SMT problem (engl. *Satisfiability modulo theories*) je problem ispitivanja da li je data formula logike prvog reda zadovoljiva u odnosu na datu teoriju T, odnosno da li postoji model teorije T u kojoj je data formula tačna. Teorije koje se obično razmatraju u praksi uključuju realnu i celobrojnu aritmetiku, teoriju nizova, listi, bitvektora, stringova, induktivnih tipova podataka, i sl. SMT rešavači su alati koji implementiraju procedure za rešavanje SMT problema. Najčešće su zasnovani na kombinaciji SAT rešavača zasnovanog na CDCL algoritmu i specifičnih procedura odlučivanja za datu teoriju (jedna ovakva arhitektura je poznata pod nazivom CDCL(T) [6]). Imaju široku primenuu u verifikaciji softverskih sistema, naročito u domenima gde je kritična bezbednost, poput avionike, automobilskih sistema i medicinskih uređaja. SMT rešavači omogućavaju proveru svojstava koda, detekciju potencijalnih grešaka kao što su prekoračenje bafera ili deljenje nulom, ali i formalnu verifikaciju sigurnosnih protokola i kriptografskih algoritama. Osim toga, koriste se i u optimizacionim problemima u industriji, gde kombinacija logičkih i numeričkih uslova omogućava efikasno raspoređivanje resursa i planiranje proizvodnje. Zahvaljujući toj kombinaciji logike i teorija, SMT rešavači su postali neizostavan alat u savremenom softverskom inženjerstvu i formalnim metodama.

Algoritmi lokalne pretrage (engl. *local search*) su se pokazali veoma efikasnim u rešavanju mnogih složenih problema optimizacije u različitim domenima. Oni iterativno poboljšavaju tekuće rešenje tako što u njega unose lokalne izmene i prihvataju promene samo ako dovode do boljeg rešenja. Algoritmi lokalne pretrage se uspešno koriste i za rešavanje SAT problema. Za razliku od egzaktnih algoritama (poput CDCL algoritma), algoritmi lokalne pretrege su nekompletni, tj. ne garantuju da će rešenje biti pronađeno, ali u praksi često brže dolaze do zadovoljavajuće valuacije od egzaktnih algoritama. Takođe, algoritmi lokalne pretrage nisu u stanju da utvrde nezadovoljivost formule. Neki od značajnijih algoritama lokalne pretrage za SAT problem su GSAT i WalkSAT [2].

U radu Caija i Zanga [10] predložen je novi algoritam za rešavanje SAT problema koji kombinuje CDCL algoritam sa algoritmima lokalne pretrage. CDCL algoritam je potpuna metoda pretraživanja koja funkcioniše tako što obilazi stablo pretrage i uči nove klauze iz konflikata na koje naiđe. Lokalna pretraga je, s druge strane,

[1] http://minisat.se/
[2] https://www.labri.fr/perso/lsimon/research/glucose/
[3] https://fmv.jku.at/lingeling/
[4] https://fmv.jku.at/cadical/

nepotpuna metoda pretraživanja koja iterativno poboljšava dato rešenje unošenjem malih izmena u njega. Predloženi algoritam iz rada [10] kombinuje prednosti obe metode tako što koristi CDCL algoritam kao glavni algoritam pretrage, a lokalnu pretragu koristi za dalje poboljšanje valuacije koju je pronašao CDCL algoritam, u cilju bržeg pronalaženja zadovoljavajuće valuacije.

U sprovedenim eksperimentima autori pokazuju da proširenje CDCL algoritma tehnikama lokalne pretrage dovodi do značajnih poboljšanja u performansama. Na skupu instanci sa SAT takmičenja 2020, modifikovana verzija rešavača Glucose[5], rešila je 62 dodatne instance (oko 15,5% više), dok je Maple-DL[6] rešio 67 dodatnih instanci (oko 16,8% više) u poređenju sa originalnim verzijama. I rešavač Kissat[7] beleži dobitak od 10 dodatnih instanci (oko 2,5%). Poboljšanja su vidljiva i po vremenskoj efikasnosti, merenoj PAR2 metrikom: za Maple-DL prosečno penalizovano vreme rešavanja smanjeno je sa 5835s na 4171s, za glucose sa 6494s na 4977.9s, dok je za Kissat smanjeno sa 4048s na 3896s.

U ovom radu dat je detaljni opis pomenute tehnike ugradnje lokalne pretrage u CDCL algoritam, predstavljene u radu [10]. Glavni doprinos ovog rada je adaptacija ove tehnike u kontekstu SMT rešavača. Tehnika je implementirana u okviru SMT rešavača ArgoSMT [3] i otvorenog je koda. Rad takođe sadrži i evaluaciju dobijene implementacije na korpusima instanci iz SMT-LIB biblioteke [1].

Nastavak ovog rada je organizovan na sledeći način. U glavi 2 uvedene su osnovne definicije, pojmovi i oznake neophodne za razumevanje pomenutih tehnika. U glavi 3 biće dat opis egzaktnih algoritama za rešavanje SAT problema (CDCL algoritam) i SMT problema (CDCL(T) algoritam). U glavi 4 se nalazi opis osnovnih algoritma lokalne pretrage za rešavanje SAT problema, WalkSAT i njegova preteča GSAT. U poglavlju 5 detaljno je predstavljena tehnika ugradnje lokalne pretrage u CDCL algoritam, dok se u poglavlju 6 razmatra njena primena pri ugradnji lokalne pretrage u CDCL(T) algoritam, po ugledu na opis iz rada [10]. U glavi 7 je dat opis implementacije ove tehnike u okviru SMT rešavača ArgoSMT i evaluacija na SMT instancama, kao i diskusija dobijenih rezultata. Na kraju, u glavi 8, kao zaključak, dat je osvrt na postignute rezultate u ovom radu.

[5] https://www.labri.fr/perso/lsimon/research/glucose/
[6] https://maplesat.github.io/solvers.html
[7] https://fmv.jku.at/kissat/

# Glava 2

# Osnove

U ovom delu rada biće date osnovne definicije, pojmovi i oznake potrebne za razumevanje ostatka rada.

## SAT problem

Neka je dat prebrojivi skup iskaznih promenljivih. Literal (engl. *literal*) je ili promenljiva $v$ ili njena negacija $\neg v$. Suprotan literal literalu $l$ označava se sa $\bar{l}$. Ukoliko je literal $l$ promenljiva $v$, njemu suprotan literal je negacija promenljive $v$, odnosno $\neg v$, a ukoliko je literal $l$ negacija promenljive $v$, njemu suprotan literal je upravo $v$. Klauza (engl. *clause*) predstavlja disjunkciju literala. KNF formulu čini konjukcija klauza i biće označavana sa $F$.

Pod valuacijom podrazumevamo pridruživanje istinitosnih vrednosti (tačno, netačno) iskaznim slovima. Ukoliko je promenljiva $v$ tačna u datoj valuaciji, tada je $\neg v$ netačno u toj valuaciji i obrnuto. Klauza je tačna u datoj valuaciji ako je u njoj tačan bar jedan njen literal. KNF formula je tačna u datoj valuaciji ako su sve njene klauze tačne u toj valuaciji. KNF formula F je zadovoljiva ako postoji bar jedna valuacija u kojoj je F tačna, a u suprotnom je nezadovoljiva. Problem ispitivanja zadovoljivosti KNF formule naziva se SAT problem. Ovaj problem je NP kompletan [5]. Alati koji implementiraju procedure odlučivanja za SAT problem nazivaju se SAT rešavači.

## SMT problem

Signatura (ili jezik) $\Sigma$ se sastoji iz skupa funkcijskih i predikatskih simbola datih arnosti. Bazni term je ili funkcijski simbol arnosti 0 (konstanta) ili funkcijski

simbol arnosti $n > 0$ primenjen na $n$ baznih termova. Bazni atom prvog reda je ili predikatski simbol arnosti 0, ili predikatski simbol arnosti $n > 0$ primenjen na $n$ baznih termova. Literal prvog reda je ili atom prvog reda, ili njegova negacija. Klauza prvog reda je disjunkcija literala prvog reda. Bazna KNF formula prvog reda je konjunkcija baznih klauza prvog reda. Struktura $M$ se sastoji iz nepraznog domena $D$ i mapiranja koje funkcijskim simbolima iz $\Sigma$ pridružuje funkcije odgovarajuće arnosti nad $D$, dok predikatskim simbolima pridružuje relacije odgovarajuće arnosti nad $D$. Struktura $M$ indukuje interpretaciju termova i atoma nad $\Sigma$ na uobičajen način, pri čemu se termovi interpretiraju kao elementi skupa $D$, dok se atomi interpretiraju kao istinitosne vrednosti (tačno, netačno). Sada se istinitosne vrednosti literala, klauze i KNF formule u strukturi $M$ definišu analogno kao u slučaju iskazne logike. Pod teorijom prvog reda $T$ nad $\Sigma$ podrazumevamo skup struktura nad $\Sigma$ koje nazivamo modelima teorije $T$. Formula $F$ je zadovoljiva u teoriji $T$ (ili $T$-zadovoljiva) ako postoji model $M$ teorije $T$ u kom je $F$ tačna, u suprotnom je $F$ $T$-nezadovoljiva. Problem ispitivanja zadovoljivosti formule u teoriji $T$ nazivamo SMT problem za teoriju $T$. Alati koji implementiraju procedure za rešavanje SMT problema nazivaju se SMT rešavači.

# Glava 3

# Egzaktni algoritmi

U ovoj glavi će biti prikazani egzaktni algoritmi koji se dominantno koriste u SAT i SMT rešavačima. Algoritam zasnovan na učenju klauza vođenom konfliktima (engl. *conflict driven clause learning* (CDCL)) predstavlja napredan algoritam na kome je zasnovana većina savremenih SAT rešavača. Ovaj algoritam predstavlja unapređenje Davis-Putnam-Logemann-Loveland (DPLL) algoritma i biće detaljno razmotren u poglavlju 3.1.

S druge strane, moderni SMT rešavači su zasnovani na CDCL(T) algoritmu, koji kombinuje SAT rešavač zasnovan na CDCL algoritmu sa procedurama odlučivanja za teorije prvog reda poput aritmetike, bit vektora, i sl. CDCL(T) algoritam će detaljno biti razmotren u glavi 3.2.

## 3.1 CDCL algoritam

U ovom poglavlju razmatra se CDCL algoritam koji predstavlja najčešće korišćen egzaktni algoritam za rešavanje SAT problema. Glavni deo algoritma pokušava da inkrementalno izgradi zadovoljavajuću valuaciju mehanizmom odlučivanja i propagacija, uz detekciju konflikata (tj. klauza koje su netačne u tekućoj parcijalnoj valuaciji). U slučaju da se pojavi konflikt, tada se vrši analiza konflikata, čiji je rezultat učenje klauze koja je stvarni uzrok konflikta i nehronološko vraćanje unazad na osnovu naučene klauze. Navedene komponete algoritma detaljnije su opisane u narednim odeljcima.

Označimo sa $\alpha$ parcijalnu valuaciju koja se formira u okviru SAT rešavača. Literal $l$ je *tačan* u valuaciji $\alpha$, sa oznakom $\alpha \models l$, ukoliko je element valuacije $\alpha$, a *netačan* u valuaciji $\alpha$, sa oznakom $\alpha \models \neg l$, ukoliko je njemu suprotan literal element valuacije

$\alpha$. Ukoliko literal $l$ nije ni tačan ni netačan u valuaciji $\alpha$ , kažemo da je *nedefinisan* u valuaciji $\alpha$, sa oznakom $\alpha \not\models l, \neg l$. Klauza $C$ je *tačna* u valuaciji $\alpha$, sa oznakom $\alpha \models C$, ukoliko postoji bar jedan literal $l \in C$ tako da $\alpha \models l$, a *netačna* u valuaciji $\alpha$, sa oznakom $\alpha \models \neg C$, ukoliko za svaki literal $l \in C$ važi $\alpha \models \neg l$. Netačne klauze u valuaciji često se u praksi nazivaju $konfliktnim\ klauzama$. Formula $F$ je *tačna* u valuaciji $\alpha$, sa oznakom $\alpha \models F$, ukoliko za svaku klauzu $C \in F$ važi $\alpha \models C$, a *netačna* u valuaciji $\alpha$ , sa oznakom $\alpha \models \neg F$, ukoliko postoji $C \in F$ takva da važi $\alpha \models \neg C$. Za formulu i klauzu koje nisu ni tačne ni netačne u valuaciji $\alpha$ kažemo da su *nedefinisane* u $\alpha$.

Parcijalna valuacija će u okviru SAT rešavača biti predstavljena kao stek na kome se nalaze literali koje smatramo tačnim u tekućoj parcijalnoj valuaciji. Ovaj stek se u literaturi obično naziva *trag* (engl. *trail*). Trag je izdeljen na *nivoe odlučivanja* sa početnom numeracijom 0, pri čemu poslednji nivo nazivamo *tekući nivo odlučivanja*. *Literali odlučivanja* su literali kojima počinju nivoi odlučivanja (sem nultog nivoa) i koji se nalaze na steku kao rezultat proizvoljne odluke algoritma (tj. predstavljaju tačke grananja). Preostali literali na svakom od nivoa odlučivanja se nazivaju *izvedeni literali*, a nastali su kao posledica prethodno donetih odluka, mehanizmom propagacije. Nivoi odlučivanja su uvedeni da bi se omogućilo vraćanje unazad, a literali odlučivanja su tačke u koje algoritam može da se vrati i izabere alternativni put pretrage.

Struktura CDCL algoritma data je u Algoritmu 1.

**Algoritam 1** CDCL algoritam, $CDCL(F, \alpha)$

```
1:  dl ← 0;
2:  if UnitPropagation(F, α) = CONFLICT then
3:      return UNSAT;
4:  end if
5:  while ∃ nedefinisana promenljiva v u tragu α do
6:      x ← PickBranchVar(F, α);
7:      v ← PickBranchPolarity(F, x, α);
8:      dl ← dl + 1;
9:      α.push(x, v);
10:     if UnitPropagation(F, α) = CONFLICT then
11:         bl ← ConflictAnaysisAndLearning(F, α);
12:         if bl < 0 then
13:             return UNSAT;
14:         else
15:             Backtrack(F, α, bl);
16:             dl ← bl;
17:         end if
18:     end if
19: end while
20: return SAT
```

U liniji 2, odnosno prvi `if` vrši iscrpnu jediničnu propagaciju (o kojoj će biti reči u narednom odeljku), na nultom nivou odlučivanja. Ako dođe do konflikta odmah se prijavljuje UNSAT, tj. ne postoji zadovoljavajuća valuacija za datu ulaznu formulu. Zatim se ulazi u *while* petlju u kojoj se ostaje sve dok sve promenljive ne dobiju vrednost. Ukoliko se to postigne, a da pritom ne dođe do konflikta, algoritam vraća SAT nakon petlje. Pomoću funkcija $PickBranchVar$ i $PickBranchPolarity$ se biraju promenljiva i njen polaritet koje se dodaju na trag u liniji 9, dok se u liniji 8 povećava tekući nivo odlučivanja. Ovim povećanjem se postiže da dodati literal zapravo bude literal odlučivanja. Nakon postavljanja literala odlučivanja ide se u naredni cikus jedinične propagacije, a ako se tom prilikom desi konflikt, on se analizira u liniji 11 pozivom funkcije $ConflictAnaysisAndLearning$ koja određuje nivo odlučivanja na kome se nalazi pravi uzrok konflikta, a vraćanjem na ovaj nivo izbegava se nepotrebno pretraživanje neperspektivnih grana prostora pretrage. Takođe, analiza konflikta kao rezultat ima klauzu povratnog skoka koja objašnjava prirodu konflikta i koja je posledica klauza formule F, a koja se dodaje u skup klauza, čime se sprečavaju slični konflikti u budućnosti. Ako se konflikt ne prijavi ide se na narednu iteraciju while petlje i postupak izbora literala odlučivanja se ponavlja.

## Jedinična propagacija

Jedinična propagacija (engl. *Unit Propagation*) je mehanizam koji omogućava efikasno smanjenje prostora pretrage tako što identifikuje literale koji moraju biti tačni i dodaje ih na trag. Algoritam analizira skup klauza i identifikuje klauze koje sadrže samo jedan nedefinisani literal, dok su ostali literali netačni u tekućoj parcijalnoj valuaciji. Ove klauze nazivamo jediničnim klauzama. Za svaku jediničnu klauzu, nedefinisani literal dobija istinitosnu vrednost „tačno", kako bi klauza bila zadovoljena. Ova dodela se propagira kroz formulu.

Dok se istinitosne vrednosti propagiraju, algoritam vrši dedukciju identifikujući dodatne jedinične klauze koje proističu iz dodela napravljenih u prethodnim koracima. Na taj način se uspostavlja lanac dedukcija koji sužava prostor mogućih dodela istinitosnih vrednosti i time smanjuje prostor pretrage.

Ako se javi konflikt tokom procesa jedinične propagacije, tj. ako neka od klauza postane netačna u tekućoj parcijalnoj valuaciji, algoritam prelazi u fazu analize konflikta.

Algoritam koji se tipično koristi za pronalaženje jediničnih klauza je shema dva posmatrana literala (engl. *two watched literals scheme*). U svakoj klauzi se biraju dva literala koja nisu netačna u trenutnoj valuaciji. Za svaki literal se održava lista svih klauza u kojima je on posmatrani literal. Kada neki literal postane netačan u tekućoj parcijalnoj valuaciji obilazi se njegova lista posmatranih klauza i za svaku klauzu traži se alternativni posmatrani literal. Ako se nađe novi posmatrani literal ta klauza se dodaje na njegovu listu posmatranih klauza. U suprotnom postoje dve mogućnosti: da je drugi posmatrani literal takođe netačan u tekućoj valuaciji (prijavljuje se konflikt) ili da je drugi posmatrani literal nedefinisan (taj literal će biti propagiran).

## Proces donošenja odluka

Jedan od važnih elemenata CDCL algoritma je proces donošenja odluka, gde se vrši izbor promenljive odlučivanja i njenog polariteta (funkcije $pickBranchVar$ i $pickBranchPolarity$ u algoritmu 1).

Algoritam koristi heuristiku kako bi izabrao promenljivu kojoj će se dodeliti vrednost. Heuristike donošenja odluka mogu se zasnivati na različitim kriterijumima, kao što su najviše pojavljivanja u klauzama, najmanje pojavljivanja ili neka druga metrika koja odgovara karakteristikama problema. Jedna od često korišćenih

heuristika u CDCL algoritmu je VSIDS (engl. *Variable State Independent Decaying Sum*) heuristika [9]. Ona dodeljuje bodove promenljivama na osnovu njihovog učešća u konfliktima tokom pretrage. Promenljive koje su učestvovale u većem broju konflikata u skorije vreme dobijaju više bodova i prioritetno se razmatraju pri donošenju odluka. Ova heuristika omogućava algoritmu da se fokusira na promenljive koje su verovatno ključne za konflikte i da izbegne beskorisno pretraživanje.

Nakon izbora promenljive, algoritam dodeljuje polaritet promenljivoj, a jedna od najčešćih strategija izbora je *strategija sačuvanog polariteta*, kod koje se bira onaj polaritet koju je promenljiva poslednji put imala u prethodnom toku pretrage.

## Analiza konflikata, učenje klauza i povratni skokovi

Analiza konflikata (engl. *conflict analysis*) je komponenta CDCL algoritma koja se koristi za identifikovanje uzroka konflikata i učenje iz njih. Kada se tokom jedinične propagacije otkrije konflikt, algoritam postupkom rezolucije, polazeći od konfliktne klauze, konstruiše klauzu koja objašnjava pravi uzrok konflikta. U pitanju je iterativni postupak tokom koga se polazna konfliktna klauza transformiše na sledeći način: u svakom koraku, identifikuje se izvedeni literal iz trenutne konfliktne klauze koji je poslednji poništen na tragu $\alpha$ i eliminiše se iz konfliktne klauze primenom rezolucije sa klauzom iz koje je taj literal dobijen jediničnom propagacijom (ovu klauzu nazivamo *objašnjenje propagiranog literala*). Ovaj postupak se ponavlja do dostizanja *tačke jednoznačne implikacije* (engl. *unique implication point* (UIP)), u kojoj važi da su svi literali iz trenutne konfliktne klauze osim jednog poništeni na tragu $\alpha$ na nivou manjem ili jednakom nekom $m$, dok je samo jedan literal poništen na nekom nivou većem od $m$ (tipično na tekućem nivou odlučivanja).

Klauza dobijena na kraju opisanog postupka analize konflikta se naziva *klauza povratnog skoka*. Ova klauza se koristi za određivanje nivoa povratnog skoka, a takođe se dodaje u tekući skup klauza, kako bi se izbegli slični konflikti u budućnosti. Ovaj proces dodavanja klauze povratnog skoka se naziva *učenje klauza*.

Povratni skok (engl. *Backjump*) je karakteristika CDCL algoritma koja omogućava izbegavanje neperspektivnih grana prostora pretrage. Na osnovu konfliktne klauze, algoritam identifikuje nivo odlučivanja na tragu na kome se nalazi stvarni uzrok konflikta. Ovaj nivo se naziva *nivo povratnog skoka*. U pitanju je minimalni nivo na kome je klauza povratnog skoka jedinična klauza. Algoritam se vraća na ovaj nivo i na osnovu klauze povratnog skoka vrši propagaciju odgovarajućeg literala.

## 3.2 Algoritam CDCL(T)

SMT rešavač zasnovan na CDCL(T) arhitekturi se sastoji iz SAT rešavača zasnovanog na CDCL algoritmu i *teorijskog rešavača* koji implementira proceduru odlučivanja za ispitivanje zadovoljivosti konjunkcije literala u teoriji T. Struktura CDCL(T) algoritma prikazana je u algoritmu 2, a dodatne metode teorijskog rešavača koje se pozivaju iz SAT rešavača date su u nastavku:

- *newLevel()*: poziva se svaki put kada se na tragu uspostavi novi nivo odlučivanja. Ovo je neophodno, kako bi teorijski rešavač mogao da zapamti svoje interno stanje na kraju svakog nivoa odlučivanja. Zapamćeno stanje se kasnije može rekonstruisati, u slučaju vraćanja unazad.

- *assertLiteral(l)*: poziva se svaki put kada se na trag postavi novi literal, kako bi teorijski rešavač bio informisan o promeni stanja parcijalne valuacije.

- *backjump(m)*: poziva se pri svakom povratnom skoku. Ovim se teorijski rešavač informiše o promeni stanja parcijalne valuacije i ujedno se nalaže teorijskom rešavaču da se vrati na stanje u kome je bio na kraju nivoa $m$.

- *checkConflict(R)*: nalaže teorijskom rešavaču da ispita da li u tekućoj parcijalnoj valuaciji postoji konflikt u teoriji. Ukoliko je odgovor potvrdan, teorijski rešavač vraća podskup literala $R$ sa traga $\alpha$ takav da je $R \models_T \bot$. Ovaj skup nazivamo *objašnjenje teorijskog konflikta* i njegovom negacijom dobija se konfliktna klauza od koje započinje analiza konflikta. Ova procedura se obično poziva nakon završenog ciklusa jedinične propagacije, pod pretpostavkom da isti nije proizveo konflikt. U algoritmu se poziva u okviru funkcije $TheoryPropagation$ - linija 6.

- *checkTheoryPropagation(L)*: ovom procedurom se nalaže teorijskom rešavaču da proveri da li postoje literali nedefinisani u tekućoj parcijalnoj valuaciji, a koji u teoriji T slede iz literala koji se nalaze na tragu. Ukoliko postoje, skup L svih takvih literala se vraća SAT rešavaču koji ih postavlja na trag na tekućem nivou odlučivanja. Opisani postupak se naziva *teorijska propagacija* i predstavlja dodatni mehanizam zaključivanja, analogan mehanizmu jedinične propagacije, s tim što se ovoga puta radi o rezonovanju u teoriji. Ova procedura se obično poziva nakon što se utvrdi da ne postoji konflikt u teoriji. U algoritmu se poziva u okviru funkcije $TheoryPropagation$ - linija 6.

- *explainLiteral(l, E)*: nalaže teorijskom rešavaču da objasni literal $l$ koji je ranije postavljen na trag mehanizmom teorijske propagacije. *Objašnjenje teorijske propagacije* $E$ predstavlja bilo koji skup literala sa traga $\alpha$ koji prethode literalu $l$ takav da važi $E \models_T l$. Ova procedura se poziva tokom analize konflikta, kad god je potrebno iz konfliktne klauze eliminisati literal koji je nastao teorijskom propagacijom (rezolucija se sprovodi nad klauzom $\neg E \vee l$ čime se eliminiše literal $l$ iz tekuće konfliktne klauze). U algoritmu se poziva u okviru funkcije $AnalyzeTheoryConflict$ - linija 13.

**Algoritam 2** CDCL(T) algoritam

```
1: dl ← 0;
2: while true do
3:     i ← 0;
4:     (α, conf) ← UnitPropagation(F, α);
5:     while i < num_t_solvers OR conf = CONFLICT do
6:         (α, conf) ← TheoryPropagation(F, α, i);
7:         i ← i + 1;
8:     end while
9:     if conf = CONFLICT AND check_conflict = true then
10:        if dl = 0 then
11:            return UNSAT;
12:        end if
13:        (learned_clause, bl) ← AnalyzeTheoryConflict(conf, i);
14:        F ← F ∪ {learned_clause};
15:        Backtrack(α, bl), dl ← bl;
16:        j ← 0;
17:        while j < num_t_solvers do
18:            theory_solver[j] → backjump(bl);
19:            j ← j + 1;
20:        end while
21:        continue;
22:    end if
23:    if IsCompleteTrail(α, F) then
24:        break;
25:    end if
26:    decision_var ← ChooseDecisionVariable(α, F);
27:    α ← α ∪ {decision_var = TRUE/FALSE};
28:    j ← 0;
29:    while j < num_t_solvers do
30:        theory_solver[j] → newLevel();
31:        theory_solver[j] → assertLiteral(decision_var);
32:        j ← j + 1;
33:    end while
34: end while
35: return SAT;
```

# Glava 4

# SAT rešavači zasnovani na lokalnoj pretrazi

GSAT i WalkSAT su popularni algoritmi lokalne pretrage koji se koriste za rešavanje SAT problema i u ovom poglavlju će biti više reči o njima.

## 4.1 GSAT

GSAT algoritam (algoritam 3) je randomizovani algoritam lokalne pretrage koji ima za cilj pronalaženje zadovoljavajuće valuacije za datu formulu koja je u KNF formi. Algoritam počinje sa nekom početnom valuacijom, koja može biti nasumična ili zasnovana na nekom heurističkom pristupu. Algoritam iterativno pokušava da poboljša trenutnu valuaciju tako što menja vrednost nasumično izabrane promenljive sa ciljem da maksimizuje broj zadovoljenih klauza formule. GSAT nastavlja sa iteracijama sve dok ne pronađe zadovoljavajuću valuaciju (sve klauze su zadovoljene), ili dok se ne ispuni unapred definisani uslov zaustavljanja (maksimalan broj iteracija ili vremensko ograničenje).

Prilikom menjanja vrednosti, algoritam može da ima bočna kretanja (engl. *sideways moves*). To su koraci pri kojima nema povećanja broja zadovoljenih klauza formule. Kada nema promene u broju zadovoljenih klauza nakon promene vrednosti izabrane promenljive, to znači da je algoritam došao u lokalni maksimum, takozvani plato (engl. *plateau*) jer trenutno ne može da nađe bolje rešenje. Plato može biti privremeno stanje ili može trajati tokom celog izvršavanja algoritma. Ovo se smatra glavnim problemom u GSAT algoritmu jer sprečava dalje poboljšanje valucije i može dovesti do suboptimalnog rešenja. Kada algoritam zapadne u plato, postoji potreba

za mehanizmima ili heuristikama koje mogu prevazići ovaj problem i nastaviti sa traganjem za boljim rešenjem. U većini slučajeva plato vodi ka drugom platou, dok sa velikim brojem promenljivih algoritam ima manju šansu da se zaglavi u lokalnom minimumu, ali postoji mogućnost trošenja velike količine vremena na individualnim platoima.

**Algoritam 3** GSAT(F)

```
for i = 1 to MAX_TRIES do
    σ ← slučajno generisana valuacija
    for j = 1 to MAX_FLIPS do
        if σ zadovoljava F then
            return σ
        end if
        v ← promena vrednosti promenljive koja rezultuje najvećim smanjenjem
              broja nezadovoljenih klauza
        Promeni vrednost v u σ
    end for
end for
return FAIL
```

## 4.2 WalkSAT

WalkSAT algoritam (algoritam 4) je algoritam lokalne pretrage za SAT problem nastao kao poboljšanje GSAT algoritma. Za razliku od prethodnog algoritma, WalkSAT bira promenljivu iz slučajno izabrane klauze koja je nezadovoljena u tekućoj valuaciji. Nakon izabrane nezadovoljene klauze, algoritam primenjuje *slobodan potez* (eng. *freebie move*), tj. bira promenljivu (iz prethodno slučajno izabrane klauze) čijom promenom se neće povećati broj nezadovoljenih klauza. Ideja iza slobodnih poteza je da se koristi informacija o strukturi problema i klauzama kako bi se identifikovali koraci koji vode ka poboljšanju rešenja.

Ako takva promenljiva ne postoji u klauzi, algoritam će sa određenom verovatnoćom $p$ na slučajan način izabrati promenljivu iz izabrane klauze, dok sa verovatnoćom $1-p$ bira promenljivu iz izabrane klauze čija bi promena dovela do minimalnog povećanja broja nezadovoljenih klauza (ova vrednost se obično naziva *break-count* vrednost). Pogodna vrednost parametra $p$ se dobija eksperimentalno. Na primer, za slučajno generisane 3-SAT probleme utvrđeno je da je ta vrednost 0.57 [8]

**Algoritam 4** WalkSAT(F)

**for** $i = 1$ to $MAX\_TRIES$ **do**
  $\sigma \leftarrow$ slučajno generisana valuacija
  **for** $j = 1$ to $MAX\_FLIPS$ **do**
    **if** $\sigma$ zadovoljava $F$ **then**
      **return** $\sigma$
    **end if**
    C $\leftarrow$ slučajno izabrana nezadovoljena klauza
    **if** $\exists$ promenljiva $x \in C$ za koju je break-count $= 0$ **then**
      $v \leftarrow x$ //**slobodan potez**
    **else**
      Sa verovatnoćom $p$: //**potez nasumičnog hoda**
        $v \leftarrow a$ promenljiva iz $C$ izabrana slučajno
      Sa verovatnoćom $1 - p$: //**pohlepan potez**
        $v \leftarrow a$ promenljiva iz $C$ sa najmanjim break-count
    **end if**
    Promeni vrednost $v$ u $\sigma$
  **end for**
**end for**
**return** $FAIL$

# Glava 5

# Ugradnja lokalne pretrage u CDCL algoritam

Nakon opisa CDCL algoritma u glavi 3 i algoritama lokalne pretrage u glavi 4 postavljeni su temelji za razumevanje postupka ugradnje algoritama lokalne pretrage u CDCL algoritam, na način na koji je to opisano u radu [10], a što će biti predmet razmatranja u ovoj glavi. Ideja je da se koristi CDCL algoritam kako bi bila pronađena pogodna valuacija na koju bi zatim bio primenjen algoritam lokalne pretrage. Pretpostavka je da bi za tako formiranu valuaciju bilo potrebno manje koraka u algoritmu lokalne pretrage za pronalaženje zadovoljavajuće valuacije. Ako lokalna pretraga ne nađe zadovoljavajuću valuaciju, CDCL algoritam nastavlja pretragu sa mesta gde je zaustavljen.

## 5.1 Opis postupka ugradnje lokalne pretrage u CDCL algoritam

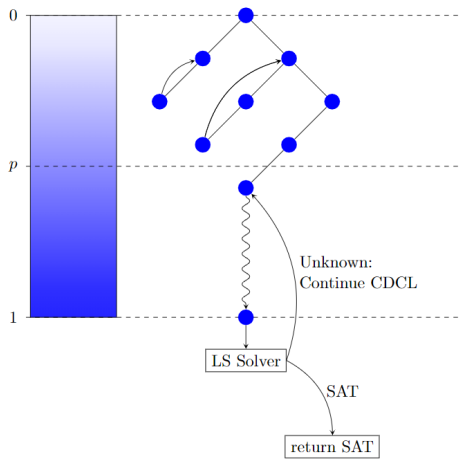
Postupak ugradnje lokalne pretrage u CDCL algoritam prikazan je u algoritmu 5. Kada CDCL algoritam tokom pretrage dođe do čvora sa parcijalnom valuacijom koja zadovoljava neke unapred zadate kriterijume (a koji će biti precizno opisani u nastavku) pretraga se zaustavlja, a taj čvor se pamti kako bi se pretraga kasnije mogla nastaviti. Algoritam ulazi u režim rada bez vraćanja unazad, koji koristi jediničnu propagaciju i postojeću heuristiku grananja da dopuni ostatak valuacije bez vraćanja unazad, potpuno ignorišući eventualne konflikte. Nakon formiranja potpune valuacije, poziva se lokalna pretraga koja polazeći od te valuacije pokušava

da pronađe zadovoljavajuću valuaciju. Ako lokalna pretraga ne uspe da pronađe model za određeno vreme, algoritam se vraća na osnovnu CDCL pretragu od čvora u kome je CDCL algoritam bio zaustavljen. Slika 5.1 ilustruje opisani proces.

Parcijalna valuacija $\alpha$ ispunjava kriterijum za prelazak u režim lokalne pretrage ako ispunjava bar jedan od sledeća dva uslova:

- $\frac{|\alpha|}{|V|} > p$ , gde je $\alpha$ parcijalna valuacija u kojoj nije identifikovan konflikt, $|V|$ ukupan broj promenljivih, a p parametar sa vrednošću između 0 i 1. Vrednost ovog parametra se određuje eksperimentalno (u radu [10] je korišćena vrednost 0.4, dobijena na osnovu preliminarnih eksperimenata na slučajnom uzorku instanci sa SAT takmičenja).

- $\frac{|\alpha|}{|\alpha\_longest|} > q$, gde je $\alpha\_longest$ najduža do tada pronađena parcijalna valuacija, a q parametar sa vrednošću između 0 i 1. Vrednost ovog parametra se takođe određuje eksperimentalno (u radu [10] je korišćena vrednost 0.9.)




Slika 5.1: Grafički prikaz relaksacije CDCL-a, slika je uzeta iz rada [10]

Jedinična propagacija koja se koristi za upotpunjavanje parcijalne valuacije za lokalnu pretragu je identična kao za osnovni CDCL algoritam uz ignorisanje eventualnih konflikata. Pravilo odlučivanja se primenjuje na promenljive koje ostanu nedefinisane nakon iscrpne primene jedinične propagacije. Nakon primene pravila odlučivanja (u skladu sa postojećim heuristikama grananja) ponovo se pokreće jedinična propagacija. Ovaj postupak se ponavlja dok se ne dobije potpuna valuacija.

**Algoritam 5** CDCL sa ugrađenom lokalnom pretragom

```
1: dl ← 0;
2: α ← 0;
3: α_longest ← 0;
4: if UnitPropagation(F, α) = CONFLICT then
5:     return UNSAT;
6: end if
7: while ∃ nedefinisana promenljiva v u tragu α do
8:     x ← PickBranchVar(F, α);
9:     v ← PickBranchPolarity(F, α);
10:    dl ← dl + 1;
11:    α.push(x, v);
12:    if UnitPropagation(F, α) = CONFLICT then
13:        bl ← ConflictAnaysis(F, α);
14:        if bl < 0 then
15:            return UNSAT;
16:        else
17:            α_longest ← max(α_longest, α);
18:            Backtrack(F, α, bl), dl ← bl;
19:        end if
20:    else if (|α|/|V| > p OR |α|/|α_longest| > q) then
21:        β ← α;
22:        while β is not complete do
23:            β ← PickBranchVar(F, β);
24:            β ← PickBranchPolarity(F, β);
25:            β.push(x, v);
26:            UnitPropagation(F, β);
27:        end while
28:        if LocalSearch(β, terminate_condition) then
29:            return SAT
30:        end if
31:        Update(α_longest_LS, α_latest_LS, α_best_LS)
32:        if Ispunjeni kriterijumi za ponovno pokretanje then
33:            Backtrack(F, α, 0);
34:            dl ← 0;
35:            updatePolaritiesFromLocalSearch();
36:            updateVSIDSScoresFromLocalSearch();
37:        end if
38:    end if
39: end while
40: return SAT
```

## 5.2 Uticaj lokalne pretrage na heuristike grananja CDCL algoritma

U prethodnom poglavlju je opisan postupak ugradnje lokalne pretrage u CDCL algoritam, gde CDCL algoritam pomaže lokalnoj pretrazi tako što pruža pogodnu početnu tačku, nakon koje će lokalna pretraga nastaviti traganje za zadovoljivom valuacijom. U ovom poglavlju se razmatra na koji način lokalna pretraga može pomoći CDCL algoritmu, tj. na koji način informacije dobijene lokalnom pretragom možemo iskoristiti za dalje usmeravanje CDCL pretrage.

Postoje dva načina na koji se to može uraditi. Prvi način podrazumeva upotrebu polariteta literala u valuaciji dobijenoj lokalnom pretragom. Ovi polariteti se koriste za ažuriranje sačuvanih polariteta, na osnovu kojih CDCL pridružuje polaritete literalima odlučivanja. Drugi način je uticaj frekvencije učešća promenljivih u konfliktima tokom lokalne pretrage na skorove promenljivih u VSIDS heuristici izbora promenljive odlučivanja. U nastavku ovog poglavlja razmatramo obe navedene tehnike.

Svaki put kada se CDCL rešavač ponovo pokrene (pretraživanje se vraća na nulti nivo odluke), sačuvani polariteti svih promenljivih se ažuriraju na osnovu valuacije dobijene lokalnom pretragom. S tim ciljem se beleži najbolja valuacija (sa najmanje nezadovoljenih klauza) u svakom pokretanju algoritma lokalne pretrage.

Preciznije, razmatraju se sledeće valuacije dobijene u prethodnim pozivima lokalne pretrage:

- $\alpha\_longest\_LS$ - odnosi se na valuaciju dobijenu lokalnom pretragom u kojoj se početno rešenje proširuje na osnovu $\alpha\_longest$, pri čemu je $\alpha\_longest$ najduža parcijalna valuacija prethodne CDCL pretrage. Kada god se $\alpha\_longest$ ažurira, algoritam ažurira i $\alpha\_longest\_LS$.
- $\alpha\_latest\_LS$ - ovo je valuacija dobijena u poslednjem pozivu lokalne pretrage.
- $\alpha\_best\_LS$ - najbolja valuacija (sa najmanje nezadovoljenih klauza) od dosadašnjih valuacija dobijenih lokalnom pretragom.

Kad god se CDCL ponovo pokrene vrši se ažuriranje sačuvanih polariteta promenljivih na osnovu jedne od gore navedenih valuacija, sa verovatnoćama datim u tabeli 5.1. Takve promene su uvek dozvoljene, jer ne utiču na korektnost CDCL algoritma. Opisani proces ažuriranje sačuvanih polariteta pokušava da postigne dva

Tabela 5.1: Verovatnoća upotrebe različitih valuacija prilikom ažuriranja sačuvanih polariteta

| Ime promenljive | $\alpha\_longest\_LS$ | $\alpha\_latest\_LS$ | $\alpha\_best\_LS$ | bez promene |
|---|---|---|---|---|
| Verovatnoća | 20% | 65% | 5% | 10% |

cilja - intenziviranje i diversifikaciju: $\alpha\_longest\_LS$ i $\alpha\_best\_LS$ služe za nalaženje dužih parcijalnih valuacija, dok $\alpha\_latest\_LS$ dodaje diversifikaciju, pošto lokalna pretraga započinje sa početnim valuacijama na različitim granama. S obzirom na to koliko se ponovna pokretanja često dešavaju u savremenim SAT rešavačima, ažuriranje polariteta na osnovu lokalne pretrage se vrši prilično često i sa verovatnoćom od 25% ide u smeru koji odredjuju ili $\alpha\_longest\_LS$ ili $\alpha\_best\_LS$.

Drugi način usmeravanja CDCL pretrage se ogleda u poboljšavanju strategije izbora promenljive odlučivanja korišćenjem frekvencije učešća promenljive u konfliktima tokom lokalne pretrage, sa akcentom na poslednjem pozivu lokalne pretrage. Za svaku promenljivu njena *frekvencija* u konfliktima tokom lokalne pretrage je definisana kao broj koraka u kojima se pojavljuje u bar jednoj nezadovoljenoj klauzi podeljen sa ukupnim brojem koraka lokalne pretrage, a zatim pomnoženo sa konstantom koja je ceo broj (u radu [10] je uzeto 100). Vrednost frekvencije se izračunava prema poslednjem pozivu lokalne pretrage. Nakon svakog ponovnog pokretanja CDCL algoritma, dobijene frekvencije se koriste za ažuriranje VSIDS skorova promenljivih, koji se uvećavaju za vrednosti ovih frekvencija. Time se u budućim grananjima u CDCL algoritmu favorizuju promenljive koje su učestvovale u većem broju konflikata tokom lokalne pretrage.

# Glava 6

# Ugradnja lokalne pretrage u CDCL(T) algoritam

U prethodnoj glavi je dat opis ugradnje lokalne pretrage u CDCL algoritam i uticaj lokalne pretrage na heuristiku grananja CDCL algoritma. U ovoj glavi razmatramo ugradnju ove tehnike u SMT rešavače zasnovane na CDCL(T) algoritmu.

Glavni izazov pri ugradnji lokalne pretrage u CDCL(T) algoritam jeste usklađivanje njenog rada sa teorijskim rešavačima. Naime, uloga teorijskih rešavača je da otkrivaju teorijske konflikte (tj. nezadovoljivost parcijalne valuacije u teoriji) kao i da otkrivaju teorijske propagacije (tj. logičke posledice tekuće parcijalne valuacije u teoriji). Pritom, u standardnom CDCL(T) algoritmu nema smisla ispitivati postojanje propagacija nakon što se otkrije konflikt, već se tada prelazi na objašnjenje konflikta i vraćanje unazad. Zbog toga su procedure odlučivanja u teorijskim rešavačima tako dizajnirane da ne proveravaju postojanje teorijskih propagacija u slučaju da je već utvrdjen konflikt u teoriji. Sa druge strane, u slučaju ugradnje lokalne pretrage, potrebno je obezbediti da teorijski rešavači nastave sa otkrivanjem teorijskih propagacija, ignorišući pritom postojanje konflikta. Ovaj proces se obavlja dok se valuacija ne kompletira, nakon čega se nad njom pokreće lokalna pretraga. Otkrivanje teorijskih propagacija u ovoj fazi je važno, jer želimo da dobijena valuacija bude što je moguće više usaglašena sa logičkim zakonitostima koje važe u teoriji.

Sledeći veliki problem se javlja prilikom izlaska iz lokalne pretrage kada je potrebno interne strukture podataka teorijskih rešavača vratiti na stanje koje su imale pre početka lokalne pretrage, kako bismo mogli da nastavimo sa radom CDCL(T) algoritma tamo gde smo stali. Najelegantnije rešenje je da najpre vratimo teorijske rešavače na stanje koje su imali na nultom nivou (tj. da ih restartujemo), a

da zatim, koristeći prethodno sačuvane literale odlučivanja, uz primenu propagacija ponovo formiramo odgovarajuće stanje struktura podataka. Naravno, ne postoji garancija da će propagacije biti izvršene istim redosledom niti sa identičnim objašnjenjima, ali to nije od suštinskog značaja za korektnost algoritma. Drugo rešenje bi bilo čuvanje internih struktura teorijskih rešavača, što bi bilo previše skupo. U ovom radu, mi smo se odlučili za prvi pristup.

Postupak ugradnje lokalne pretrage u CDCL(T) algoritam prikazan je u algoritmu 6.

**Algoritam 6** CDCL(T) sa ugrađenom lokalnom pretragom

```
1:  dl ← 0;
2:  α ← 0;
3:  α_longest ← 0;
4:  check_conflict ← true
5:  while true do
6:      Propagation;                                  //algoritam 7
7:      if IsCompleteTrail(α, F) then
8:          break;
9:      end if
10:     if (|α|/|V| > p OR |α|/|α_longest| > q) then
11:         α_copy ← α;
12:         check_conflict ← false;
13:         (α, dl) ← completeTrail(F, α, dl);          //algoritam 8
14:         β ← α;
15:         Restart;
16:         flip ← 0;
17:         while MAX_FLIP > flip do
18:             if LocalSearch(β, terminate_condition) then
19:                 for (i = β.indexOfFirstLevel(); i < β.size(); i ← i + 1) do
20:                     α ← α ∪ decision(β[i])
21:                 end for
22:                 Propagation;
23:                 if conf = CONFLICT then
24:                     F ← F ∪ ¬β;
25:                     Restart;
26:                     continue;
27:                 else
28:                     return SAT;
29:                 end if
30:             end if
31:         end while
32:         Restart;
33:         α ← α_copy;
34:         check_conflict ← true;
35:         Propagation;                              //algoritam 7
36:     end if
37:     Update(α_longest_LS, α_latest_LS, α_best_LS);
38:     decision(F, α, dl);                           //algoritam 9
39: end while
40: return SAT;
```

**Algoritam 7** *Propagation*

```
1: (α, conf) ← UnitPropagation(F, α, check_conflict);
2: i ← 0;
3: while i < num_t_solvers OR conf = CONFLICT do
4:         (α, conf) ← TheoryPropagation(F, α, i);
5:         i ← i + 1;
6: end while
7: if conf = CONFLICT AND check_conflict = true then
8:     if dl = 0 then
9:         return UNSAT;
10:    end if
11:    (learned_clause, bl) ← AnalyzeTheoryConflict(conf, i);
12:    F ← F ∪ {learned_clause};
13:    α_longest ← max(α_longest, α);
14:    Backtrack(α, bl), dl ← bl;
15:    continue;
16: end if
```

**Algoritam 8** *completeTrail*$(F, \alpha, dl)$

```
1: while α is not complete do
2:     (α, conf) ← UnitPropagation(F, α, check_conflict);
3:     i ← 0;
4:     while i < num_t_solvers OR conf = CONFLICT do
5:         (α, conf) ← TheoryPropagation(F, α, i);
6:         i ← i + 1;
7:     end while
8:     decision(F, α, dl)                          //algoritam 9
9: end while
10: return (α, dl);
```

**Algoritam 9** *decision*$(F, \alpha, dl)$

```
1: decision_var ← ChooseDecisionVariable(α, F);
2: dl ← dl + 1;
3: α ← α ∪ {decision_var = TRUE/FALSE};
4: return (α, dl);
```

Na početku algoritma se vrši iscrpna propagacija (jedinična i teorijska). Taj deo koda je smešten u algoritam 7 radi preglednosti. U slučaju da dođe do konflikta ulazi se u analizu konflikta, generisanje klauze povratnog skoka i vraćanje unazad. Ukoliko nije došlo do konflikta, pre dodavanja novog literala odlučivanja (algoritam 9) proveravaju se uslovi za ulazak u lokalnu pretragu na liniji 10. Uslovi koji treba da budu ispunjeni su detaljno izloženi u poglavlju 5.1.

Kada se uslovi postignu, kreira se kopija trenutne parcijalne valuacije (označena sa $\alpha\_copy$ u algoritmu). Ova kopija neće sadržati sve literale parcijalne valuacije, već samo one sa nultog nivoa, kao i literale odlučivanja. Ove informacije su dovoljne da nakon lokalne pretrage teorijske rešavače vratimo u pređašnje stanje, kao i da rekonstruišemo parcijalnu valuaciju $\alpha$.

Originalna parcijalna valuacija se proširuje do potpune valuacije i tokom tog procesa pokreće se propagacija, kako jedinična, tako i teorijska, kao i proces odabira literala odlučivanja (algoritam 8). Takođe, u tom koraku se konflikti ignorišu, bez obzira na to da li potiču iz klauza ili iz teorije. Ovo je tehnički omogućeno sa indikatorom $check\_conflict$ koji se postavlja na $false$ pre početka gore navedenog procesa, dok se pri izlasku iz lokalne pretrage indikator vraća na $true$ (linija 33). Potpuna valuacija se kopira u novu promenljivu $\beta$ (beta trag) na koju će se primeniti lokalna pretraga, a glavna valucija se restartuje (vraća na nulti nivo odlučivanja). Prilikom restartovanja, strukture podataka svih teorijskih rešavača se takođe vraćaju na stanje u kom su bile na nultom nivou odlučivanja.

Lokalna pretraga se izvodi po već objašnjenom principu, uz prilagođavanje procesa u slučaju pronalaska potencijalno tačne valuacije. Tada se na glavni trag dodaju literali sa dobijenog beta traga (kao literali odlučivanja), a zatim za tako kompletiranu valuaciju teorijski rešavači proveravaju postojanje konflikata u teorijama. Ako nijedan teorijski rešavač ne prijavi konflikt imamo zadovoljavajuću valuaciju. Sa druge strane, ako neki od teorijskih rešavača prijavi konflikt, ne ulazi se u objašnjavanje konflikta, već se beta trag negira i dodaje u skup naučenih klauza, nakon čega se postupak lokalne pretrage nastavlja.

Nakon završetka petlje lokalne pretrage, glavni trag se ponovo restartuje, a ranije sačuvani literali odlučivanja se dodaju na trag jedan po jedan, praćeni iscrpnom propagacijom, kako bi se vratilo pređašnje stanje rešavača.

# Glava 7

# Implementacija i evaluacija

U prvom poglavlju ove glave biće prikazana implementacija tehnike ugradnje lokalne pretrage u CDCL(T) algoritam. Tehnika je implementirana u okviru SMT rešavača ArgoSMT [3], u programskom jeziku C++, i javno je dostupna [1]. U drugom poglavlju su predstavljeni rezultati evaluacije različitih konfiguracija modifikovanog rešavača, kao i originalnog, radi poređenja.

## 7.1 Implementacija

Centralni deo implementacije nalazi se u klasi *Solver*, koja predstavlja osnovnu komponentu ArgoSMT rešavača. Ova klasa implementira metodu *solve*(), u kojoj se izvršava glavna petlja algoritma CDCL(T). Tokom rada, pozivaju se procedure koje redom sprovode propagaciju, detekciju konflikata, objašnjavanje propagiranih literala i donošenje odluka. Konkretno, klasa sadrži metode poput *apply_propagate*(), *apply_conflict*(), *apply_explain*() i *apply_decide*(), od kojih svaka implementira jedan od osnovnih koraka CDCL(T) algoritma i menja interno stanje rešavača. Metode relevantne za ovaj rad biće prikazane u tabeli 7.1. Na ovaj način se gradi struktura pretrage, rešavaju konflikti i ažurira trag sa odgovarajućim literalima.

[1] `https://github.com/dijanaalanovic/Master/blob/main/argosmt-master-moj`

Tabela 7.1: Metode klase Solver

| Metod | Opis |
|---|---|
| $apply\_decide(l)$ | Primenjuje pravilo grananja za literal $l$ koje podrazumeva postavljanje literala na trag, ako prethodno trag ne sadrži njega ili njegovu negaciju. |
| $apply\_propagate(l, i)$ | Primenjuje pravilo jedinične propagacije ili teorijske za literal $l$ i teoriju sa indeksom $i$ ($i = 0$ za jediničnu propagaciju). |
| $apply\_conflict(conf, i)$ | Poziva se kada se prepozna konflikt i s tim započinje analiza konflikta i formiranje klauze povratnog skoka ( $i = 0$ u slučaju iskaznog konfikta). |
| $apply\_explain(l, expl, i)$ | Izvedeni literal $l$ iz skupa objašnjenja konflikta se zamenjuje svojim objašnjenjem $expl$ za teoriju sa indeksom $i$ ( $i = 0$ za literale izvedene jediničnom propagacijom). |
| $apply\_restart()$ | Restartuje trag na nulti nivo kao i stanja teorijskih rešavača kako bi se izbeglo zapadanje u neperspektivne grane prostora pretrage. |

Pošto smo pojasnili referentne metode klase $Solver$, naredni korak je razmatranje detalja implementacije lokalne pretrage, koja je takođe deo ove klase i implementirana je kroz metodu $local\_search()$. U okviru iste klase dodata je i metoda $chooseVariable\_and\_flip(...)$, koju poziva $local\_search()$ nad odabranom klauzom. Njena uloga je da sprovede logiku izbora pogodnog literala i izvrši promenu njegovog polariteta.

Svaki teorijski rešavač (uključujući i specijalni rešavač zadužen za rezonovanje nad klauzama, tj. za jediničnu propagaciju) poseduje implementiranu metodu $check\_and\_propagate(...)$, koja se u glavnoj petlji metode $solve()$ poziva radi provere konflikata i sprovođenja propagacije u teorijama. U slučaju otkrivanja konflikta (propagacije) u teoriji, ova metoda poziva metodu $apply\_conflict()$ (odnosno $apply\_propagate()$) klase Solver. Za potrebe lokalne pretrage je napravljeno njeno preopterećenje sa dodatim argumentom koji predstavlja indikator. Ako je vrednost indikatora 1 to bi značilo da se funkcija poziva u okviru lokalne pretrage i da bi trebalo ignorisati eventualnu prijavu konflikata i njihovo objašnjavanje. U tu svrhu se indikator propagira kroz sve metode teorijskih rešavača gde imamo mogućnost prijave konflikta. Takođe, ovaj indikator nam govori da li treba pozvati metodu $apply\_conflict()$ u slučaju otkrivanja konflikta u teoriji, ili taj konflikt treba ignorisati i nastaviti sa propagacijama.

Metoda $apply\_restart()$ se koristi u metodi lokalne pretrage na više mesta. Prvo

nailazimo na nju prilikom restartovanja traga nakon kopiranja u radni beta trag kako bi stanje teorijskih rešavača bilo vraćeno na stanje iz nultog nivoa odlučivanja. Sledeće jako bitno mesto je kada se naiđe na potencialno tačnu parcijalnu valuaciju u beta tragu. Literali sa beta traga se dodaju kao literali odlučivanja i pokreće se propagacija (kod 7.1). Ako se prijavi konflikt pokreće se metoda restartovanja koja bi vratila stanje rešavača na stanje pre dodavanja literala sa beta traga i nastavlja se sa traženjem potencijalno tačne parcijalne valuacije sa kojom bi se proces ponovio. Sledeće pokretanje je pri izlasku iz lokalne pretrage u slučaju da zadovoljavajuća valuacija nije pronađena, kada je potrebno vratiti se na stanje rešavača pre ulaska u lokalnu pretragu (kod 7.2).

Beta trag koji predstavlja kopiju potpune valuacije nad kojom se primenjuje lokalna pretraga je definisan kao neuređen skup radi lakše pretrage literala.

Kod 7.1: Propagacija pozvana u okviru lokalne pretrage radi provere potencijalno zadovoljavajuće valuacije.

```
unsigned i = 0;
do {
    _state_changed = false;
    _theory_solvers[i]->check_and_propagate(layer, 1);
    // Second parameter is local search flag

    if(_state_changed && (i != 0 || layer != 0)){
        i = layer = 0;
  }
    else{
     i++;
  }
  if(_conflict_set.is_conflict()){ //conflict
        apply_restart();
        ind_for_theory_solvers = 1;
        break;
    }
}
while(i < _theory_solvers.size());
```

Kod 7.2: Nije pronađena zadovoljavajuća valuacija. Vraćanje internih struktura podataka teorijskih rešavača na stanje pre ulaska u lokalnu pretragu.

```
apply_restart();
_conflict_set.reset_conflict();
for(unsigned j = 0; j < alpha_copy.size(); j++){
    unsigned ind = 0;
    for(unsigned k = lvl_0; k < _trail.size(); k++){
        if(alpha_copy[j] == _trail[k] or get_literal_data(
           alpha_copy[j])->get_opposite() == _trail[k]){
            ind = 1;
      break;
        }
    }
    if (ind == 1)
        continue;

    apply_decide(alpha_copy[j]);
    // After each decide step, propagation is performed
    unsigned i = 0;
    layer = num_of_layers -1;
    do
    {
        _state_changed = false;
        _theory_solvers[i]->check_and_propagate(layer);

        if(_state_changed && (i != 0 || layer != 0))
            i = layer = 0;
        else
          i++;

    }
    while(!_conflict_set.is_conflict() && i < _theory_solvers.
       size());
}
```

## 7.2 Evaluacija

Evaulacija predstavljene tehnike rađena je na računaru sa četiri AMD Opteron 6168 1.6GHz procesora sa po 12 jezgara (tj. 48 jezgara ukupno), i 94GB RAM-a. Korpus instanci nad kojima je evaluacija rađena je iz SMT-LIB biblioteke [3] sa ukupnim brojem od 1083 instanci. Prilikom evaluacije, vremensko ograničenje bilo je 1200 sekundi po instanci, dok je memorijsko ograničenje bilo implicitno određeno hardverskim ograničenjima samog sistema.

Rezultati će biti predstavljeni za nekoliko najboljih vrednosti koeficijenata, a to su pomenuti $p$, $q$ i ograničenje broja iteracija petlje lokalne pretrage, pri čemu će se za svaku kombinaciju analizirati broj rešenih instanci, prosečno vreme rešavanja i vreme trajanja same lokalne pretrage, a nakon toga će se najbolji rezultat uporediti sa rezultatom originalnog rešavača.

Prvi koeficijent koji se analizira je $p$ i za njega su izabrane tri vrednosti sa najboljim rezultatima koji se mogu naći u tabeli 7.2.

Tabela 7.2: Rezultati postignuti za različite vrednosti koeficienta $p$. Vremena su izražena u sekundama.

| Vrednost $p$ | 0.40 | 0.45 | 0.50 |
|---|---|---|---|
| Br. rešenih instanci | 112 | 111 | 119 |
| Srednje vreme rešavanja [3] | 2163.71 | 2163.87 | 2150.07 |
| Srednje vreme rešavanja po instanci [4] | 115.14 | 96.12 | 125.45 |
| Srednje vreme lokalne pretrage | 90.52 | 60.13 | 82.9 |

Podsetimo se da se koeficijent $p$ koristi u uslovu za ulazak u lokalnu pretragu, tj. da bi se započela lokalna pretraga $p$ treba da bude manje od odnosa dužine parcijalne valuacije i ukupnog broja promenljivih. Veće vrednosti parametra $p$ pomeraju prag za ulazak u lokalnu pretragu, što dovodi do toga da se algoritam ređe oslanja na stohastične komponente i više na sistematičnu pretragu. Sa druge strane, manje vrednosti $p$ omogućavaju češće aktiviranje lokalne pretrage, što povećava mogućnost pronalaska rešenja u slučajevima gde CDCL(T) sam po sebi zapada u neperspektivne grane. Najbolji rezultati postignuti su za vrednost $p = 0.5$ za koju je rešeno najviše instanci, uz relativno stabilno prosečno vreme rešavanja.

Koeficijent $q$ se nalazi u drugom delu uslova za ulazak u lokalnu pretragu i on treba da bude manji od odnosa dužine trenutne parcijalne valuacije i najduže parcijalne valuacije do tada nađene. U tabeli 7.3 prikazane su 4 različite vrednosti koeficijenta sa najboljim rezultatima.

Tabela 7.3: Rezultati dobijeni za različite vrednosti koeficienta $q$. Vremena su izražena u sekundama.

| Vrednost $q$ | 0.87 | 0.9 | 0.93 | 0.97 |
|---|---|---|---|---|
| Br. rešenih instanci | 114 | 119 | 114 | 112 |
| Srednje vreme rešavanja | 2158.95 | 2150.07 | 2159.58 | 2162.4 |
| Srednje vreme rešavanja po instanci | 110 | 125.45 | 116.02 | 125.45 |
| Srednje vreme lokalne pretrage | 57.34 | 82.9 | 52.59 | 51.68 |

Rezultati pokazuju da vrednost $q = 0.9$ donosi najbolju kombinaciju između broja rešenih instanci i prosečnog vremena. Suviše niska vrednost q vodi ka preuranjenom uključivanju lokalne pretrage, dok previsoka vrednost produžava vreme čekanja do njenog aktiviranja i smanjuje efikasnost tehnike. Ovo potvrđuje hipotezu da lokalna pretraga najbolje funkcioniše kada se aktivira tek nakon što CDCL(T) obezbedi dovoljno „dobru“ parcijalnu valuaciju.

[3] Odnos ukupnog vremena i ukupnog broja istanci

[4] Vreme rešavanja istance za koju je nađeno rešenje

Poslednja analiza odnosi se na koeficijent koji predstavlja ograničenje broja iteracija petlje lokalne pretrage, u nastavku $MAX\_FLIPS$. On predstavlja maksimalan broj promena polariteta literala tokom izvršavanja funkcije lokalne prtrage. U tabeli 7.4 je dat prikaz rezultata za različite vrednosti koeficijenta.

Tabela 7.4: Rezultati dobijeni za različite vrednosti parametra $MAX\_FLIPS$. Vremena su izražena u sekundama.

| Vrednost $MAX\_FLIPS$ | 500 | 450 | 400 |
|---|---|---|---|
| Br. rešenih instanci | 114 | 113 | 119 |
| Srednje vreme rešavanja | 2158.28 | 2160.27 | 2150.07 |
| Srednje vreme rešavanja po instanci | 103.71 | 102.28 | 125.45 |
| Srednje vreme lokalne pretrage po instanci | 46.24 | 45.43 | 82.9 |

Prevelik broj iteracija može uzrokovati bespotrebno trošenje vremena na lokalnu pretragu bez garancije uspeha, dok premali broj iteracija ograničava njenu moć da izbegne lokalne minimume. Najbolji rezultati postignuti su za vrednost $MAX\_FLIPS = 400$, za koju je dobijena najbolja kombinacija broja rešenih instanci i vremena izvršavanja.

Najbolji rezultat je dobijen za vrednosti $p = 0.5$, $q = 0.9$ i $MAX\_FLIPS = 400$, stoga se vrši njegovo poređenje sa rezultatom originalnog rešavača.

Vrednosti originalnog rešavača se mogu naći u tabeli 7.5.

Tabela 7.5: Rezultati poređenja izvršavanja rešavača $ArgoSMT$ sa i bez lokalne pretrage. Vremena su izražena u sekundama.

| Rešavači | $ArgoSMT$ | $ArgoSMT\_local\_search$ |
|---|---|---|
| Br. rešenih instanci | 616 | 119 |
| Br. zadovoljivih instanci | 432 | 62 |
| Srednje vreme rešavanja | 1105.421 | 2150.07 |
| Srednje vreme rešavanja po instanci | 124 | 125.45 |

Iz datog se zaključuje da je originalni rešavač rešio 616, a modifikovani 119 od ukupno 1083 istance, od kojih je 591 zadovoljivih, 373 nezadnovoljivih, a ostale nerešene. Prevedeno u procente, originalni rešavač je rešio 56.88% instanci dok je modifikovana verzija rešila 10.99%.

Ako se uđe u dublju analizu vremena 119 rešenih instanci dolazi se do sledećeg:

- 17.65% (21 instanca) - rešene su brže u odnosu na original,
- 59.66% (71 instancu) - vreme rešavanja se razlikuje za manje od 1s,

- 17.65% (21 instancu) - razlika u vremenu rešavanja se nalazi u intervalu od 1s do 100s,
- 5% (6 instanci) - razlika u vremenu rešavanja je veća od 100s.

Iako je originalni ArgoSMT rešio znatno veći broj instanci, modifikovani rešavač sa ugrađenom lokalnom pretragom pokazuje određene prednosti. Analiza rešenih instanci otkriva da je skoro 18% slučajeva rešeno brže nego kod originala, dok se kod dodatnih 60% vreme razlikuje za manje od 1s. Ovo sugeriše da hibridni pristup, iako trenutno inferioran u ukupnom broju rešenja, poseduje potencijal za optimizaciju i poboljšanje performansi.

Glavni nedostatak modifikovanog rešavača jeste drastičan pad ukupne uspešnosti u odnosu na originalni ArgoSMT. Međutim, činjenica da je kod značajnog broja instanci vreme rešavanja uporedivo ili bolje ukazuje da se daljim unapređenjem mehanizama integracije lokalne pretrage može postići veća konkurentnost.

Vidimo da glavna ideja, odnosno upotreba lokalne pretrage, ima značajan potencijal za dalje unapređivanje. Deo algoritma koji najviše utiče na vreme izvršavanja lokalne pretrage jeste izbor promenljive čijom promenom polariteta se neće povećati broj nezadovoljenih klauza (algotitam 4 - *slobodan potez*). Konkretno, kada literal klauze prilikom promene polariteta postane netačan, mora se proveri da li među preostalim literalima te klauze postoji bar jedan koji će u odnosu na trenutni trag održati klauzu tačnom. Ova provera se sprovodi za sve literale izabrane klauze u svim klauzama formule i predstavlja usko grlo u performansama. Potencijalna unapređenja mogu se ostvariti ubrzavanjem ili modifikovanjem ovog procesa, na primer izborom efikasnijih struktura podataka. Dalje unapređenje može se postići uvođenjem tehnika mašinskog učenja, koje bi omogućile pozivanje mehanizma lokalne pretrage u situacijama u kojima to može biti povoljno za ukupne performanse rešavača.

# Glava 8

# Zaključak

U okviru ovog istraživačkog rada razmatrana je mogućnost kombinovanja egzaktnih algoritama i algoritama lokalne pretrage u okviru SAT i SMT rešavača, sa ciljem poboljšanja njihove efikasnosti. U neophodnoj meri dat je opis CDCL i CDCL(T) algoritma, kao i dva popularna algoritma lokalne pretrage za rešavanje SAT problema, GSAT i WalkSAT. Posebna pažnja posvećena je postupku ugradnje lokalne pretrage u CDCL algoritam, rađenom na osnovu referentnog rada [10], dok je centralni doprinos ovog rada adaptacija tehnike u kontekstu SMT rešavača i njena implementacija u okviru rešavača ArgoSMT.

Eksperimentalna evaluacija pokazala je da, iako originalni rešavač značajno nadmašuje modifikovani po broju rešenih instanci, postoji određeni broj slučajeva gde je modifikovani pristup dao komparativne rezultate ili čak nadmašio osnovnu verziju. Posebno je značajan podatak da je kod skoro 77% rešenih instanci vreme rešavanja bilo približno kao kod originalnog rešavača, što ukazuje na potencijal tehnike i otvara prostor za dodatne optimizacije.

Kao pravac za dalja istraživanja izdvajaju se: unapređenje komunikacije između lokalne pretrage i teorijskih rešavača, ispitivanje drugih heuristika i kriterijuma za ulazak u lokalnu pretragu, kao i integracija naprednijih algoritama lokalne pretrage. Pored toga, zanimljivo bi bilo ispitati primenu kombinovanog pristupa i u drugim domenima.

Na osnovu svega izloženog, može se zaključiti da iako trenutni rezultati ne donose neposredna poboljšanja u odnosu na originalni rešavač, sama ideja kombinovanja egzaktnih i stohastičkih algoritama predstavlja perspektivno polje istraživanja koje uz dalja usavršavanja može doprineti razvoju efikasnijih hibridnih rešavača.

# Bibliografija


[1] SMT-LIB. on-line at: `http://smtlib.cs.uiowa.edu/`.

[2] Marin Heule Armin Biere and Hans van Maaren. Incomplete algorithms. In *Handbook of satisfiability.IOS press*, pages 187–189, 2019.

[3] Milan Banković. Smt rešavač argosmt. 2016. on-line at: `https://github.com/milanbankovic/argosmt`.

[4] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing – New York, NY, United States, 3th May, 1971*, pages 151–158.

[5] Rivest L.Ronald Stein Clifford Cormen H. Thomas, Leiserson E. Charles. Np-completeness. In *Introduction to algorithms 4rd ed.*, pages 1343–1422, 2022.

[6] Ines Lynce Joao Marques-Silva and Sharad Malik. Conflict-driven clause learning sat solvers. In *Handbook of satisfiability.IOS press*, pages 131–155, 2009.

[7] Leonid Levin. Universal'nye perebornye zadachi [universal search problems](in russian). In *Problems of Information Transmission – Russia, 1973, Poceedings (9 (3): pp. 265–266).*

[8] Ashish Sabharwal Lukas Kroc1 and Bart Selman1. An empirical study of optimal noise and runtime distributions in local search. In *Pro. 13th Int. Conf. on Theory and Applications of Satisfiability Testing, Edinburgh, Scotland, 34. July 2010.*

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

[10] Cai Shaowei and Xindi Zhang. Deep cooperation of cdcl and local search for sat. In *Theory and Applications of Satisfiability Testing – SAT 2021, 24th International Conference, Barcelona, Spain, July 5-9, 2021*, pages 64–81, 2021.

# Biografija autora

Dijana Alanović rođena je 18.04.1998. u Šapcu. Odrasla je u obližnjem naselju pod imenom Zminjak gde je 2013. godine završila osnovnu školu „Jovan Cvijić" kao nosilac Vukove diplome. Nakon toga je upisala srednju medicinsku školu „Dr Andra Jovanović" u Šapcu i završila je 2017. godine, takođe kao nosilac Vukove diplome. Odmah po završetku srednje škole, upisala je osnovne akademske studije matematike na Matematičkom fakultetu u Beogradu, pod programom „Računarstvo i informatika". Diplomirala je u roku 2021. godine sa prosekom 8.05 i stekla zvanje Diplomirani matematičar. Obrazovanje je nastavila odmah na istom fakultetu, gde u oktobru 2021. godine upisuje master studije matematike, pod istim studijskim programom. Tokom studiranja se zapošljava kao softvreski inženjer u firmi Logit Solution.