UNIVERZITET U BEOGRADU

MATEMATIČKI FAKULTET

MASTER RAD

Iskazne neuronske mreže za aproksimaciju kombinatornih kola na osnovu uzorka ulaza i izlaza

Student:
Marjana ŠOLAJIĆ

Mentor:
dr Mladen NIKOLIĆ

Članovi komisije:
prof. dr Predrag Janičić
Matematički fakultet, Beograd
prof. dr Filip Marić
Matematički fakultet, Beograd
dr Mladen Nikolić
Matematički fakultet, Beograd

Beograd, 2017

Sadržaj

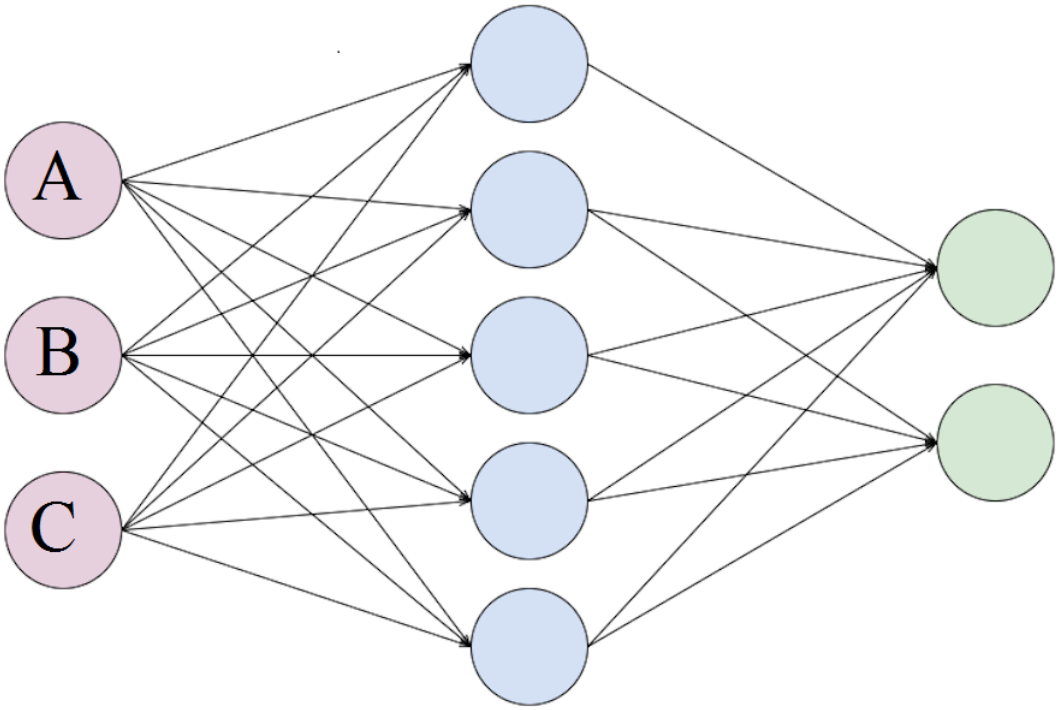
1 *Uvod*

Dizajn kombinatornih kola predstavlja važan korak u proizvodnji računarskog hardvera i drugih elektronskih uređaja. On se u značajnoj meri oslanja na procedure koje računari sprovode automatski. U ovom radu se predlaže metod aproksimacije kombinatornih kola, inspirisan metodama mašinskog učenja, koji je u stanju da sintetiše kola na osnovu datog uzorka ulaza i izlaza. Izraz aproksimacija može biti shvaćen na dva načina. Prvi se odnosi na sintezu kola sa n ulaza u slučaju uzorka manje veličine od 2^n. Imajući u vidu ovaj eksponencijalni rast, ukoliko bi ovakav model mogao da uoči zakonitosti iz skupa manjeg obima, to bi olakšalo sintezu kola. Drugi se odnosi na sintezu kola koje je saglasno sa delom zadatih instanci. U ovom trenutku interesovanje za uspešnost ovakve vrste aproksimacije je više teorijskog karaktera.

U prethodnih deset do petnaest godina mašinsko učenje vrlo brzo napreduje i pronalazi sve više primena. Neke od njih su prepoznavanje objekata na slikama i u videu, trodimenzionalna rekonstrukcija prostora na osnovu snimaka, prepoznavanje govora, mašinsko prevođenje prirodnih jezika, autonomna vožnja automobila i dronova, igranje igara, medicinska dijagnostika i slično. U nekim od ovih primena, postignuti su rezultati koji prevazilaze rezultate ljudskih eksperata. Verovatno najzanimljivi skorašnji uspeh mašinskog učenja je pobeda računara nad svetskim šampionom u igri go. Uspesi mašinskog učenja u prethodnih deset do petnaest godina se pre svega mogu pripisati razvoju metoda neuronskih mreža, koje su korišćene u navedenim oblastima.

Neuronske mreže se sastoje iz jedinica koje se nazivaju neuronima. Neuroni sadrže parametre čijim se podešavanjem može menjati ponašanje neuronske mreže. Neuroni su među sobno povezani, tako da ulazi koji se prosleđuju jednom neuronu predstavljaju izlaze drugih neurona ili ulazne podatke. Postoje algoritmi pomoću kojih se neuronske mreže mogu obučiti da prozivoljno dobro aproksimiraju bilo koju neprekidnu funkciju, ukoliko se koristi dovoljno velika količina podataka. Obično je cilj obučiti mrežu da za date ulazne vrednosti na izlazima daje vrednosti koje im odgovaraju.

Struktura elektronskih kola je obično takva da se logički elementi nadovezuju na način koji je sličan nadovezivanju neurona u neuronskim mrežama: ulazi logičkih elemenata su ili izlazi drugih logičkih elemenata ili ulazi kola. Za date ulaze, izlazi kola treba da daju vrendosti koje odgovaraju nekoj logičkoj funkciji koju kolo implementira. U ovom radu će biti razmatrana samo kombinatorna kola — kola čiji izlaz predstavlja samo funkciju ulaza, a ne i trenutnog stanja kola (ukoliko to nije ispunjeno, radi se o sekvencijalnim kolima). Ova sličnost navodi na ideju primene neuronskih mreža za modeliranje elektronskih kola na osnovu primera njihovih ulaza i izlaza. Ovaj problem je već poznat kao sinteza kombinatornih kola i već postoje neke metode koje se bave njegovim rešavanjem. Kako su se neuronske mreže pokazale vrlo uspešnim u drugim oblastima, može se očekivati da bi se one ili neki njima inspirisan pristup mogli uspešno primeniti i u sintezi kombinatornih kola. Medjutim,



SLIKA 1.1: Struktura iskazne mreže sa tri ulaza i dva izlaza.

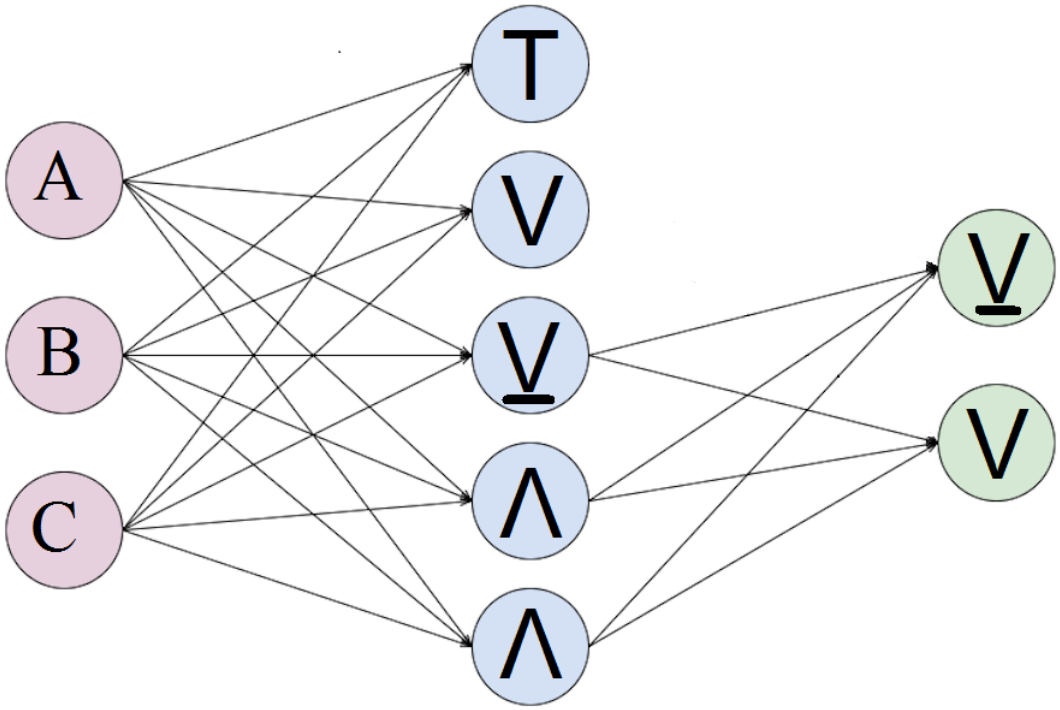
dok si izlazi logičkih elemenata samo 0 ili 1, neuroni imaju neprekidne izlaze i stoga nisu adekvatni za modeliranje logičkih elemenata.

Jedan pristup zasnovan na ideji neuronskih mreža bi bio da se osmisli reprezentacija neurona koji bi za ulaze i izlaze imao samo vrednosti 0 i 1, a koji bi imao parametre kojima se može podešavati koju funkciju predstavlja. Osnovni problem takvog pristupa je nemogućnost primene poznatih algoritama za obučavanje neuronskih mreža koji se zasnivaju na gradijentima, zato što parcijalni izvodi po parametrima takvog neurona ne bi bili definisani. S druge strane, ako i parametri uzimaju vrednosti 0 i 1, onda je moguće predstaviti problem učenja iskaznom formulom čija bi zadovoljavajuća valuacija predstavljala vrednosti takvih parametara, a za pronalaženje zadovoljavajućih valuacija iskaznih formula mogu se koristiti SAT rešavači koji koriste napredne tehnike pretrage i koji su se već pokazali vrlo uspešnim u rešavanju mnogih praktičnih problema, poput problema raspoređivanja časova, planiranja, verifikacije softvera i hardvera i drugih.

U ovom radu, predložen je jedan model *iskaznog neurona* koji u zavisnosti od parametara može predstavljati različite jednostavne logičke funkcije. Nadovezivanjem iskaznih neurona u duhu neuronske mreže dobija se *iskazna neuronska mreža*.

Na slici 1.1 shematski je prikazana struktura iskazne mreže sa tri ulaza označena slovima A, B i C i dva izlaza. Neoznačeni elementi predstavljaju iskazne neurone koji mogu da se podese tako da predstavljaju različite logičke funkcije.

Ovakva mreža može predstaviti iskazna kola poput punog sabirača koji sabira dva bita A i B uzimajući u obzir prenos C. Na slici 1.2 shematski je prikazana jedna takva iskazna mreža, pri čemu su na neuronima nacrtani simboli iskaznih veznika koji odgovaraju logičkim funkcijama koje neuroni računaju. Ova mreža nije optimalna jer sadrži neurone koji nemaju nikakvu funkciju ($\bot$ i $\vee$), ali to je prirodna posledica nepoznavanja adekvatne arhitekture pre učenja. Uklonjene strelice na ovoj slici sugerišu da se vrednosti nepovezanih neurona ne uzimaju u obzir od



SLIKA 1.2: Obučena iskazna mreža koja predstavlja puni sabirač.

strane drugih neurona, što bi bilo diktirano određenim izborom vrednosti parametara drugih neurona.

U radu je predložen postupak kojim se formira iskazna formula koja odgovara ovoj mreži i zadatim podacima za obučavanje i čije zadovoljavajuće valuacije pružaju vrednosti parametara iskaznih neurona za koje se iskazna neuronska mreža slaže sa podacima za obučavanje. Za pronalaženje tih valuacija koriste se SAT rešavači. Kako u problemima mašinskog učenja postoji tolerancija na greške takva tolerancija je omogućena i u ovom pristupu. Korišćenjem PMAX-SAT rešavača može se relaksirati SAT problem, tako da neke (unapred određene) klauze formule ne moraju biti zadovoljene, a pronalazi se rešenje koje zadovoljava maksimalan broj takvih klauza. Oslanjanje na PMAX-SAT problem omogućava saglasnost iskazne neuronske mreže sa što više instanci (uzoraka datih ulaza i izlaza), ali ne nužno sa svim. Pored navedenog, omogućeno je i korišćenje regularizacije, modifikacije problema kojom se model čini manje fleksibilnim, što se često koristi u mašinskom učenju.

U okviru ovog rada implementiran je program koji omogućava generisanje iskaznih formula u konjunktivnoj normalnoj formi koje odgovaraju određenoj arhitekturi mreže i određ enim podacima, kao i program koji za datu arhitekturu mreže, date vrednosti parametara i date ulaze izračunava izlaze iskazne neuronskre mreže. Implementacija je urađena u programskom jeziku C++.

Sprovedeni su eksperimenti u kojima se obučavaju iskazne neuronske mreže tako da prepoznaju osnovna kombinatorna kola poput sabirača, multipleksera, dekodera, komparatora i drugih, kako bi se proverila uspešnost i vremenska efikasnost njihovog obučavanja. Dobijeni rezultati su ohrabrujući, ali i sugerišu da je potrebno još unapređivanja kako bi se ovja pristup mogao koristiti za nešto veća kola.

Organizacija rada je sledeća. U glavi 2 opisani su osnovni koncepti mašinskog učenja sa akcentom na neuronskre mreže. U glavi 3 opisani su osnovni pojmovi iskazne logike, Cajtinova transformacija, ograničenja kardinalnosti, DPLL procedura i na njoj zasnovani SAT i MAX-SAT rešavači. U glavi 4 je dat kratak osvrt na

problem sinteze kombinatornih kola. U glavi 5 je opisan model iskaznih neuronskih mreža. U glavi 6 je opisana implementacija predloženog modela i dati su rezultati njegove eksperimentalne evaluacije. Na kraju su izvedeni zaključci i predložene su mogućnosti za dalji rad.

2 *Mašinsko učenje*

2.1 Osnovni pojmovi

Mašinsko učenje (eng. Machine learning (ML)) se bavi proučavanjem generalizacije i konstrukcijom i analizom algoritama koji generalizuju [13]. Pod generalizacijom se podrazumeva donošenje opštih zaključaka na osnovu pojedinačnih primera. Cilj je dizajnirati algoritme koji generalizuju i stoga mogu da vrše predviđanje. Generalizacija se obavlja na osnovu prethodnog iskustva, tj. skupa podataka o objektima koji su predmet učenja. Dobijeno znanje se koristi kako bi se dali odgovori na pitanja za objekte koji nisu ranije viđeni. U mnogim oblastima se kontinuirano prikupljaju podaci sa ciljem da se iz njih sazna nešto novo. Analiza podataka ovog tipa zahteva pristupe koji će omogućiti da se otkriju pravilnosti, zakonitosti u podacima koje nisu ni poznate, ni očigledne, a mogu biti korisne. U zavisnosti od načina analiziranja i nakon toga donošenja odluka o akcijama koje će inteligentni sistem preduzeti, može se izvršiti klasifikacija strategija mašinskog učenja.

Klasifikacija strategija učenja zasniva se na stepenu zaključivanja koji se traži kod onog koji uči (računara, robota, itd.) i deli se na sledeće:

- mehaničko učenje
- učenje na osnovu instrukcija
- deduktivno učenje
- induktivno učenje
- učenje na osnovu analogije

Mehaničko učenje je najniži nivo mašinskog učenja. Ovakvo učenje se zasniva na znanju koje je ugrađeno u inteligentni sistem, bilo to kroz programiranje ili kroz implementaciju baze podataka. Nije potrebno dodatno procesiranje ili transformacije podataka kako bi se sistem koristio. Analogija mehaničkog učenja je „učenje napamet" koje je prepoznatljivo u nekim slučajevima ljudskog učenja.

Učenje na osnovu instrukcija je zasnovano na znanju stečenom od učitelja, tj. od onog ko daje instrukcije, a transformisano je u interni oblik kroz zaključke koje učenik ili sistem mora da izvrši, držeći se strogo instrukcija koje su mu date.

Deduktivno učenje podrazumeva da učenik ili sistem koji uči, mora da izvrši transformaciju znanja deduktivnim zaključcima i da kroz reformulaciju i organizacione procedure dođe do novih formulacija.

Induktivno učenje je proces koji se zasniva na generalizaciji, tj. proces u kome se znanje koje važi za skup slučajeva prenosi na njegov nadskup. Da bi generalizacija bila uspešna, određeni aspekti objekata o kojima se rezonuje moraju biti zanemareni ukoliko nisu u od značaja za učenje. Induktivno učenje predstavlja jedan od osnovnih načina o formiranju predstave o okruženju, situacijama i dr. odnosno za

pravljenje modela podataka iz iskustva. Algoritmi induktivnog učenja omogućuju donošenje zaključaka i bez kompletnog poznavanja domena na koji se primenjuju, naravno ukoliko je u određenom domenu dozvoljena greška pri odlučivanju.

Učenje na osnovu analogije kombinuje deduktivne i induktivne vidove učenja. Učenje se može podeliti na dva koraka. Prvi korak koristi induktivno zaključivanje koje je neophodno da bi se našla zajednička podstruktura između domena problema koji se rešava i jednog od analognih domena koji su memorisani kao postojeća baza znanja. Drugi korak podrazumeva preslikavanje mogućeg rešenja iz određenog analognog domena koji najbliže odgovara rešenju, u domen problema korišćenjem deduktivne logike.

Modeli zakonitosti u podacima Učenje u stvari predstavlja pronalaženje zakonitosti, odnosno zavisnosti među raspoloživim podacima. Matematičke reprezentacije zavisnosti među promenljivim nazivamo *modelima*. Uglavnom je unapred definisana forma modela koji se razmatraju, pa je tako moguće uočiti *skup mogućih (dopustivih) modela*. Izbor modela iz skupa mogućih modela na osnovu podataka se naziva *obučavanje.*

2.2 Vrste mašinskog učenja

Vrste mašinskog učenja su:

- *nadgledano učenje (eng. supervised learning)*
- *nenadgledano učenje (eng. unsupervised learning)*
- *učenje uslovljavanjem (eng. reinforcement learning)*

2.2.1 Nadgledano učenje

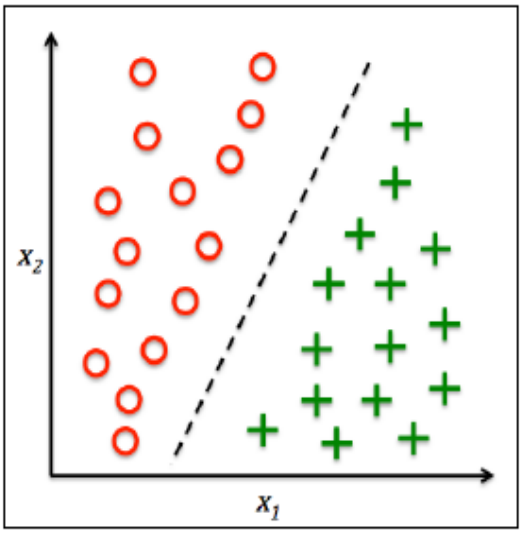
Nadgledano učenje obuhvata skup problema i tehnika za njihovo rešavanje u kojima se algoritmu, zajedno sa skupom podataka iz kojih uči, daju i željeni (tačni) izlazi, tj. vrednosti takozvane *ciljne promenljive*. Algoritam treba da nauči da za date podatke pruži odgovarajuće izlaze, ali i da novom, nepoznatom ulaznom podatku dodeli tačnu izlaznu vrednost. Pronalaženje modela u ovom slučaju možemo razumeti kao pretragu skupa mogućih modela koja je vođena podacima, a koju realizuje algoritam učenja.

Standardni problemi nadgledanog učenja su:

- *klasifikacija* - ciljna promenljiva je kategorička
- *regresija* - ciljna promenljiva je neprekidna

Klasifikacija (eng. classification) je oblast mašinskog učenja koja ima za cilj predviđanje klasnih obeležja nove instance na osnovu prethodnih opažanja. Kao primer binarne klasifikacije može se navesti detekcija neželjene pošte, zarad koje mašinski algoritam uči skup pravila kako bi razlikovao dve postojeće klase: željenu i neželjenu poštu. Naravno, skup mogućih klasa ne mora biti binaran. Tipičan primer za višeklasnu klasifikaciju bi bio prepoznavanje karaktera.

Na slici 2.1 je ilustrovan primer *binarne klasifikacije* sa skupom za obučavanje od 30 instanci. Polovina skupa za obučavanje je obeležena kao pozitivna klasa (znak plus), a polovina kao negativna klasa (krug). Isprekidana linija predstavlja granicu

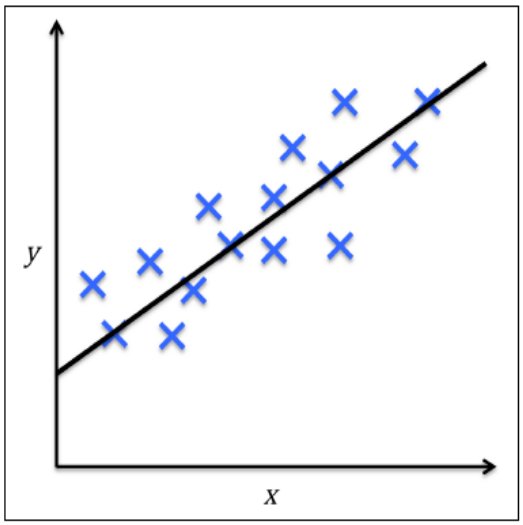


SLIKA 2.1: Primer binarne klasifikacije

odlučivanja koja deli dve pomenute klase i pomoću koje se vrši klasifikacija novog podatka.

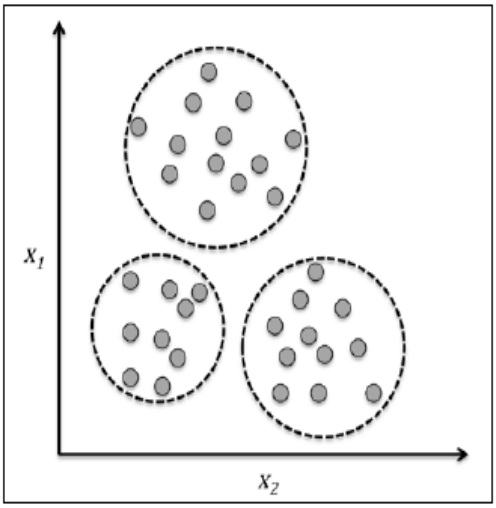
Regresija (eng. regression) je druga oblast mašinskog učenja koja se bavi predviđanjem neprekidne izlazne promenljive. Naziva se još i regresiona analiza, jer algoritam pokušava da nađe vezu između atributa i neprekidne izlazne promenljive. Na primer, pretpostavimo da želimo da predvidimo uspešnost studenata na sledećem ispitu. Ukoliko postoji relacija između utrošenog vremena za učenje i rezultata na ispitu, to možemo iskoristiti kao podatke za obučavanje modela koji bi na osnovu utrošenog vremena za učenje mogao da predvidi rezultat budućih studenata koji planiraju da polažu ispit.

U slučaju *linearne regresije*, za skup mogućih modela se uzimaju prave. Na slici 2.2 je prikazan izabrani model na osnovu podataka (atributa x i izlazne promenljive y), prava koja za dati skup podataka minimizuje sumu kvadrata odstupanja duž pravca y ose.



SLIKA 2.2: Primer linearne regresije

Primer linearne regresije bi bilo predviđanje cena nekretnina na osnovu njihove



SLIKA 2.3: Prikaz tehnike klasterovanja

površine. Podaci koji bi se koristili za učenje: površine nekretnina (x) i cene (y) u nekom gradu.

2.2.2 Nenadgledano učenje

Nenadgledano učenje obuhvata skup problema i tehnika za njihovo rešavanje u kojima se algoritmu koji uči pružaju samo ulazni podaci bez željenih izlaza. Od algoritma koji uči se očekuje da sam uoči neke zakonitosti u podacima koji su mu dati. U ovom slučaju ne postoji ciljna promenljiva, pa je potrebno naći model iz skupa dopustivih modela koji je najbolji u odnosu na neki unapred zadati kriterijum.

Klasterovanje je tehnika istraživanja podataka koja otkriva objekte podataka sličnih osobina i organizuje ih u grupe (klastere), bez ranijeg znanja o njihovom pripadanju nekoj grupi, čineći ih preglednijim i potencijalno korisnijim. Svaki klaster definiše grupu objekata koji dele među sobom određen stepen sličnosti, dok sa objektima iz ostalih klastera dele visok stepen različitosti.

Na slici 2.3 je ilustrovano kako klasterovanje može biti iskorišćeno za organizaciju neobeleženih podataka u tri različite grupe zasnovanom na sličnostima njihovih karakteristika x_1 i x_2.

Primer za koji bi mogla da se iskoristi tehnika klasterovanja je određivanje konfekcijske veličine na osnovu visine i težine ljudi. Klasteri bi predstavljali grupe ljudi sa istom konfekcijskom veličinom (S, M, L,...).

Problemi nenadgledanog učenja su:

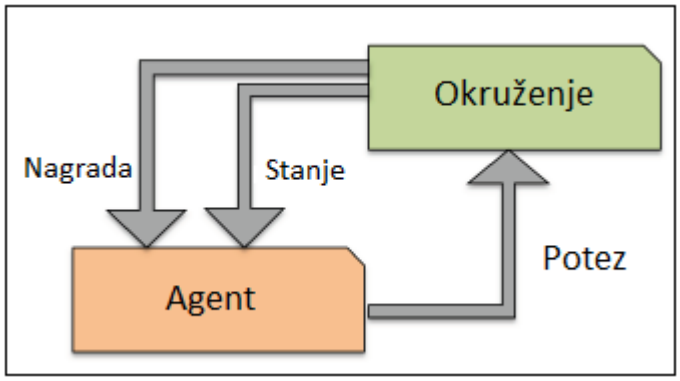
- *grupisanje* - problem učenja grupisanja, kada želimo da otkrijemo inherentne grupe u podacima (grupisanje kupaca prema ponašanju u kupovini)
- *asocijacija* - problem učenja pravila asocijacije, gde želimo da otkrijemo pravila koja opisuju velike delove podataka(npr. da kupac koji kupuje X takođe ima tendenciju da kupi Y)

2.2.3 Učenje uslovljavanjem

Učenje uslovljavanjem ili uz posticaj predstavlja vrstu algoritma mašinskog učenja koji omogućava softveru - agentu i mašinama da samostalno odrede optimalno ponašanje unutar određenog okruženja u cilju dostizanja maksimalnih performansi. Učenje

uslovljavanjem predstavlja interakciju između okruženja i agenta. Okruženje „nagrađuje" agenta za ispravnu radnju što je signal za podsticaj, u suprotnom okruženje „kažnjava" agenta. Kako agent ima više „nagrada" njegov podsticaj je veći, odnosno poboljšava znanje o okruženju i lakše bira ispravnu sledeću akciju (slika 2.4). Algoritmi ove vrste učenja ne daju eksplicitne izlaze za sve ulaze. Kroz iteraciju pokušaja i grešaka dolazi se do cilja, tj. politike ponašanja koja vodi do maksimalnih performansi sistema.

Popularan primer za učenje uslovljavanjem je šah. Igrač (agent) se odlučuje na seriju poteza u zavisnosti od stanja na šahovskoj tabli (okruženje), a nagrada može biti definisana kao pobeda ili poraz na kraju igre, ali nije za svaki potez poznato da li je dobar ili loš.



SLIKA 2.4: Šema za učenje uslovljavanjem

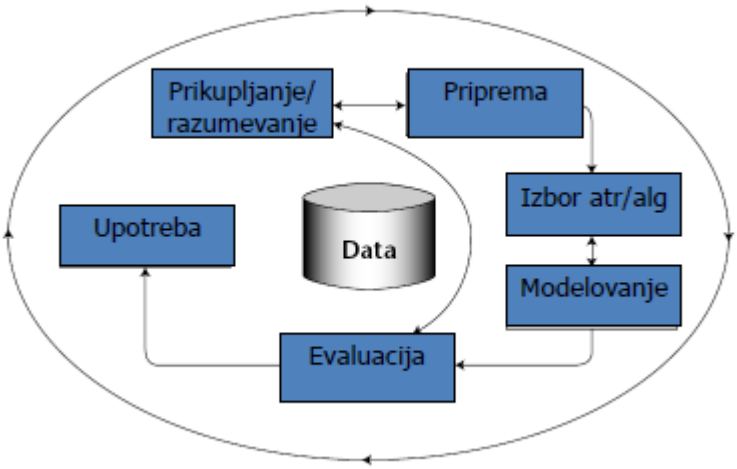
2.3 Osnovni koraci i elementi procesa mašinskog učenja

Osnovni koraci procesa mašinskog učenja (slika 2.5):

1. prikupljanje podataka potrebnih za obučavanje, validaciju i testiranje modela mašinskog učenja
2. eksplorativna analiza i vizualizacija
3. priprema podataka, „čišćenje" i „transformacija" podataka (vršiti skaliranje, redukovanje redundantnih podataka i sl. kako bi se proces učenja ubrzao)
4. izbor jedne ili više metoda mašinskog učenja
5. obuka, konfigurisanje i evaluacija kreiranih modela (npr. unakrsna validacija)
6. analiza modela

2.3.1 Reprezentacija podataka

Najčešće korišćen način predstavljanja objekata je pomoću nekih njihovih svojstava, odnosno atributa. Svojstva ili atributi predstavljaju karakteristike objekata kao što su težina, površina, oblik, boja i slično. Vrednosti atributa mogu biti numeričke, kao u slučaju frekvencije reči u nekom članku. Atributi takođe mogu predstavljati imena nekih kategorija kojima se ne mogu dodeliti numeričke vrednosti ili uređenje. Primer kategoričkog atributa može biti boja očiju osobe ili grad u kome je osoba rođena i slično.



SLIKA 2.5: Proces mašinskog učenja

Odabrati skup atributa u skladu sa time koje su karakteristike objekata bitne za dati problem učenja predstavlja pravi izazov i od presudnog je značaja za kvalitet učenja. Ako su atributi dobro izabrani, često i jednostavni algoritmi učenja mogu postići dobre rezultate. S druge strane, ukoliko atributi loše opisuju objekte, ne može se očekivati mnogo ni od najnaprednijih algoritama.

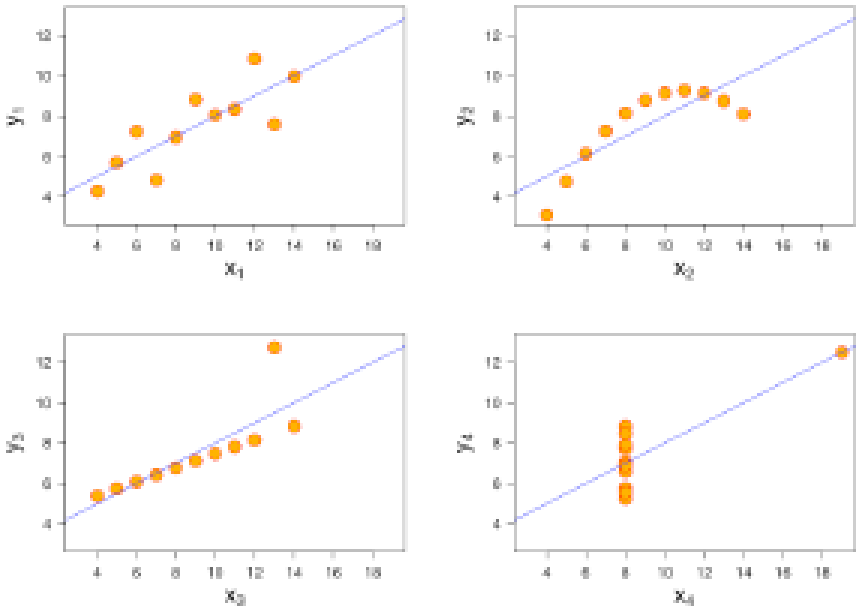
2.3.2 Podaci za obučavanje i testiranje

Podaci za obučavanje se nazivaju još i podacima za obučavanje, a njihov skup skup za obučavanje. Testiranje predstavlja procenu uspešnosti modela, tj. kvaliteta naučenog znanja. Prilikom testiranja se koriste podaci kojima model nije imao pristup u fazi učenja (test skup). Često se raspolaže ograničenom količinom podataka koja se mora upotrebiti u obe svrhe. Jedan od standardnih načina je da se jedna trećina, ili nešto približno tome, izdvoji unapred za testiranje, a da se obučavanje vrši na ostatku.

2.3.3 Izbor i evaluacija modela

Izbor modela uglavom zavisi od vrste problema koji se rešava, karakteristika skupa atributa (tip, stepen međuzavisnoti atributa, opseg vrednosti atributa,...), obima podataka koji su na raspolaganju, itd. Na slici 2.6 je prikazan pokušaj aproksimacije četiri različita skupa podataka primenom iste linearne funkcije, tj. linearne regresije. Očigledno je da različiti podaci zahtevaju različite funkcije, tj. vrste modela.

Pored izbora samog modela, bitno je izabrati i način na koji se on ocenjuje. Kao što je napomenuto u 2.3.2, praksa je da se model evaluira na odvojenom skupu podataka za testiranje. Pritom se podela raspoloživih podataka na podatke za obučavanje i podatke za testiranje vrši slučajnim izborom podataka za testiranje. Međutim, različite podele na skup za obučavanje i skup za testiranje mogu uroditi različitim rezultatima, pa slučajno deljenje nije najbolji način formiranja ovih skupova. Čest pristup za efikasno korišćenje raspoloživih podataka je takozvana *unakrsna validacija* (eng. cross-validation). Ceo skup podataka kojim se raspolaže se deli na n približno jednakih podskupova. Jedan podskup se izdvaja i obučavanje se vrši na ostalih $n - 1$ podskupova. Posle obučavanja, performanse modela se ocenjuju



SLIKA 2.6: Odabir modela

na izdvojenom podskupu. Ovaj postupak se ponavlja za sve ostale izdvojene podskupove i kao finalna ocena kvaliteta se uzima prosek dobijenih ocena za svaki od podskupova. Za vrednost n se obično uzima broj 5 ili 10 i ne preporučuju se mnogo manje ili veće vrednosti. Ovakav postupak daje stabilniju ocenu kvaliteta.

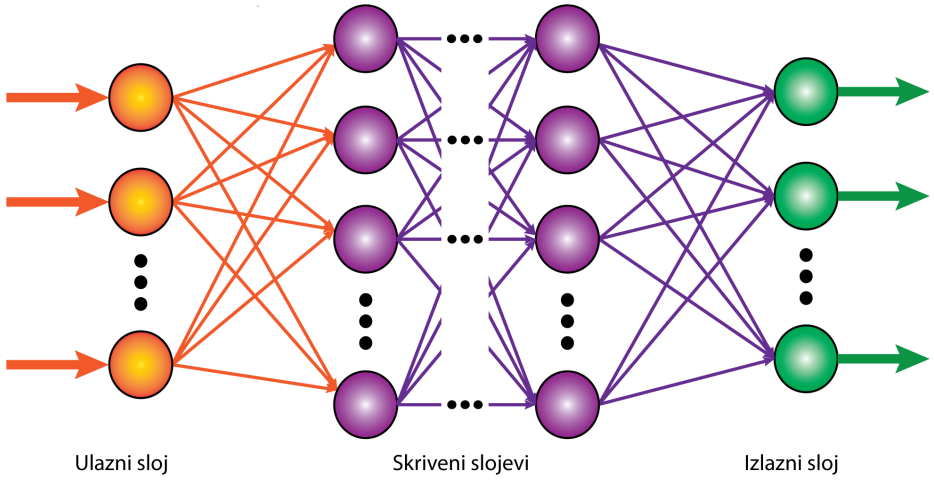
2.4 Veštačke neuronske mreže

Neuronske mreže su jedan od najpoznatijih i trenutno najaktuelnijih modela mašinskog učenja. Definišu se kao modeli za rezovanje zasnovani na analogiji sa mozgom, sa akcentom na naglašenu kognitivnu sposobnost učenja i generalizaciju stečenog znanja. Neke od njihovih primena su prepoznavanje oblika, rukopisa, govora, upravljanje robotima, predviđanje kretanja cena na tržištu, analiza medicinskih testova, kriminološka istraživanja i slično. Neuronske mreže zapravo predstavljaju univerzalni aproksimator neprekidnih funkcija. Među najznačajnijim su neuronske mreže sa propagacijom unapred (2.4.1).

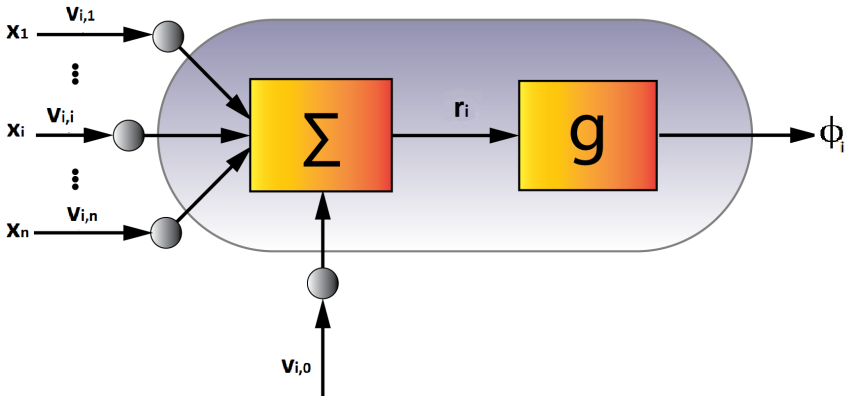
2.4.1 Neuronske mreže sa propagacijom unapred

Neuronska mreža sa propagacijom unapred (eng. feed forward neural networks) se sastoji od slojeva koji se sastoje od osnovnih jedinica - *neurona*. Način na koji se neuroni međusobno povezuju i organizuju u slojeve predstavlja arhitekturu mreže. Neuroni prvog sloja mreže primaju kao svoje argumente (*ulaze*) informacije od ulaza mreže, tj. signale iz okruženja, dok neuroni ostalih slojeva primaju kao svoje argumente vrednosti (*izlaze*) neurona prethodnog sloja. Svaki neuron, bez obzira na sloj na kom se nalazi, računa linearnu kombinaciju svojih argumenata i nad njom računa neku nelinearnu transformaciju (*aktivacionu funkciju*). Aktivacione funkcije neurona su potrebne da bi mreža imala veće mogućnosti i bila u stanju da aproksimira i nelinearne funkcije, jer bi se u suprotnom kombinovanjem linearnih funkcija ponovo dobijala linearna funkcija. Slojevi čiji neuroni prosleđuju svoje izlaze drugim neuronima se nazivaju *skrivenim slojevima*. Izlazi neurona poslednjeg sloja predstavljaju

konačne rezultate obrade i nazivaju se *izlazima mreže*. Neuronska mreža može imati više skrivenih slojeva i tada se naziva *dubokom neuronskom mrežom*. U tom slučaju, izlazi jedinica skrivenih slojeva mreže se mogu smatrati atributima dobijenim na osnovu vrednosti prethodnih slojeva. Kaže se da svaki sloj vrši *ekstrakciju atributa*, pri čemu su oni na višim slojevima kompleksniji od onih na nižim. Smatra se da je mogućnost konstrukcije novih atributa jedan od glavnih razloga za uspešnost dubokih neuronskih mreža.



SLIKA 2.7: Struktura neuronske mreže sa propagacijom unapred



SLIKA 2.8: Stuktura neurona

Model se može matematički formulisati na sledeći način:

$$\begin{aligned} \phi_0 &= x \\ r_i = V_i\phi_{i-1} + v_{i0} &\qquad i = 1, 2, \ldots, K \\ \phi_i = g(r_i) &\qquad i = 1, 2, \ldots, K \end{aligned}$$

gde je x vektor ulaznih promenljivih, K je broj slojeva mreže, V_i je matrica čija j-ta vrsta predstavlja vektor koeficijenata neurona j u sloju i, v_{i0} predstavlja vektor slobodnih članova linearnih kombinacija koje neuroni i-tog sloja izračunavaju, a g je nelinearna aktivaciona funkcija. Za vektor r_i, $g(r_i)$ predstavlja vektor $(g(r_{i1}), g(r_{i2}),$

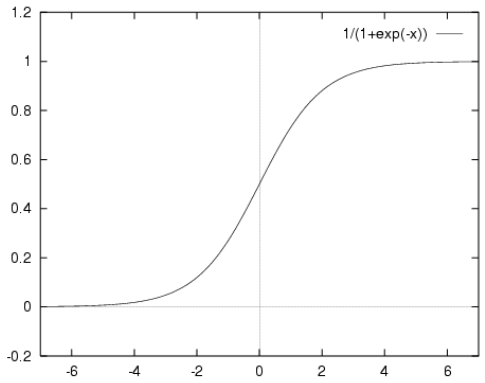
$\ldots, g(r_{in}))^T$, gde je n broj neurona u jednom sloju. Skup svih koeficijenata modela se obeležava sa v. Struktura neuronske mreže sa propagacijom unapred je prikazana na slici 2.7, a model jednog neurona na slici 2.8. Ukoliko izlaz mreže sa skupom koeficijenata v za ulaz x obeležimo sa $f_v(x)$, vaziće jednakost $f_v(x) = \phi_K$. Za aktivacionu funkciju g najčešće se koriste:

- *sigmoidna funkcija*
- *tangens hiperbolički*
- *ispravljačka linearna jedinica*

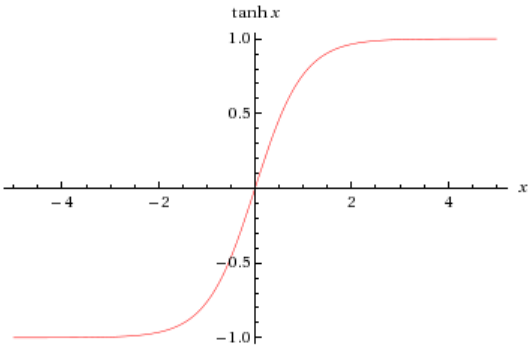
Opšti oblik *sigmoidne funkcije* ili *S-aktivacione funkcije* je:

$$g(x) = \frac{1}{1 + e^{-x}}$$

Sigmoidna funkcija je realno vrednosna, monotona i diferencijabilna, što je važno za njenu primenu u neuronskim mrežama. Grafik sigmoidne funkcije je prikazan na slici 2.9.



SLIKA 2.9: Grafik sigmoidne aktivacione funckije



SLIKA 2.10: Grafik funkcije tangens hiperbolički

Opšti oblik funkcije *tangens hiperbolički* je (grafik je prikazan na slici 2.10):

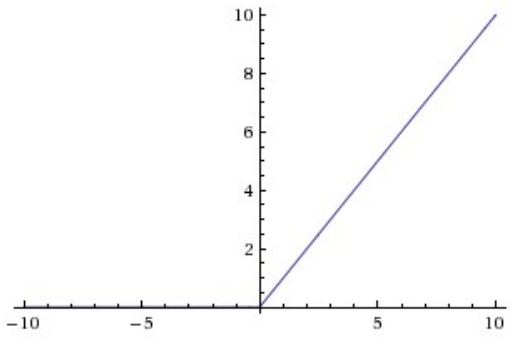
$$g(x) = \frac{e^{2x} - 1}{e^{2x} + 1}.$$

Ispravljačka linearna jedinica (eng. rectified linear unit (ReLU)) predstavlja trenutno najčešće korišćenu aktivacionu funkciju. Iako je nediferencijabilna, ima veoma pogodna

svojstva pri optimizaciji. Opšti oblik ove funkcije je:

$$g(x) = \max(0, x).$$

ReLU se najviše koristi u konvolutivnim mrežama čije je obučavanje i do nekoliko puta brže u poređenju sa drugim aktivacionim funkcijama. Grafik ReLU funkcije je prikazan na slici 2.11.



SLIKA 2.11: Ispravljačka linearna jedinica

Dve osnovne klase problema za koje se koriste neuronske mreže su:

- problem regresije - aproksimacija neprekidnih funkcija
- problem klasifikacije - aproksimacija funkcija koje uzimaju kategoričke vrednosti, ali za koje se pretpostavlja da ih ima konačno mnogo

Za rešavanje obe klase problema potrebno je koristiti metode matematičke optimizacije. Sve metode za optimizaciju neuronske mreže se oslanjaju na gradijent koji ukazuje na smer u kojem se greška modela smanjuje (videti 2.4.1).

Neuronske mreže, ipak, imaju i određene mane. Ne postoje jasne smernice kako izabrati pogodan model. Preterano složeni modeli mogu stvarati probleme pri učenju, odnosno, često ih je teško obučavati. Takođe, uglavnom su potrebne velike količine podataka. Tu je i problem izbora algoritma za optimizaciju, koji se najčešće svodi na veliki broj rešavanja optimizacionog problema dok se ne postignu zadovoljavajući rezultati, što može računski i vremenski biti vrlo zahtevno. Zbog svega navedenog upotreba neuronskih mreža zahteva dosta eksperimentisanja.

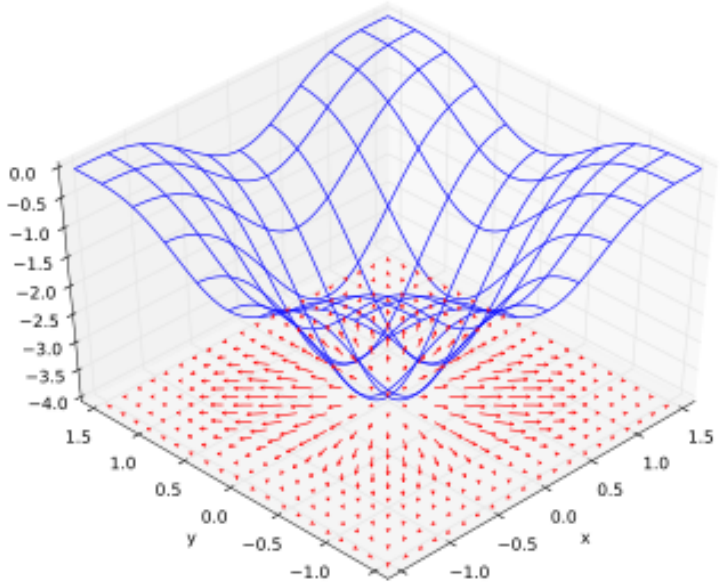
Metoda gradijentnog spusta

Gradijent funkcije f u tački $x = (x_1, \ldots, x_n)$, u oznaci $\nabla f(x)$, predstavlja vektor parcijalnih izvoda funkcije u toj tački:

$$\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \ldots, \frac{\partial f(x)}{\partial x_n} \right)$$

Gradijent $\nabla f(x)$ predstavlja pravac maksimalnog rasta funkcije f u tački x. Slično, $-\nabla f(x)$ predstavlja pravac maksimalnog opadanja funkcije f u tački x. Gradijent funkcije u različitim tačkama, prikazan je na slici 2.12.

Metoda gradijentnog spusta ili *gradijentna metoda* jedna je od najstarijih i najjednostavnijih iterativnih metoda za rešavanja problema konveksne optimizacije. Ona minimizira konveksnu diferencijabilnu funkciju. Ideja gradijentnog spusta se sastoji



SLIKA 2.12: Strelice u ravni argumenata funkcije predstavljaju gradijente funkcije u različitim tačkama.

u sledećem. Polazeći od neke nasumice izabrane tačke, nizom koraka doći vrlo blizu minimuma funkcije, tj. rešenja. Pritom, da bi se u svakom koraku dobilo poboljšanje, treba se kretati u pravcu maksimalnog opadanja funkcije, tj. u pravcu negativnog gradijenta. Tako dobijamo formulu za iterativni niz gradijentne metode:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) \quad k = 0, 1, 2, \ldots$$

gde je α_k pozitivan skalar koji nazivamo veličinom koraka. Izbor veličine koraka je veoma bitan jer za preveliku ili premalu veličinu metoda može oscilovati ili čak divergirati. Takođe, kriterijumi zaustavljanja mogu biti različiti. Najčešće se koristi zaustavljanje nakon unapred zadatog broja iteracija, nakon što razlika između susednih koraka postane manja od unapred zadate vrednosti ε ili nakon što razlika između vrednosti funkcije u susednim koracima postane manja od ε, a može se kombinovati i više kriterijuma. Ipak, jedna od mana ove metode je upravo spora konvergencija, kao i to što u slučaju postojanja više lokalnih minimuma ne garantuje pronalazak globalnog minimuma (što znači da troši puno vremena na računanje pravca koji nije optimalan).

Poboljšanje metode gradijenta je *stohastički gradijentni spust* (eng. stochastic gradient descent) koji se danas najčešće koristi za obučavanje neuronskih mreža. Osnovna ideja se zasniva na pretpostavci da se funkcija f koja se optimizuje može predstaviti kao prosek drugih funkcija f_i koje se lakše izračunavaju. Na primer, u slučaju mašinskog učenja, funkcija greške se može predstaviti kao zbir grešaka na pojedinačnim instancama. Tada se umesto gradijenta funkcije f, mogu koristiti gradijenti funkcija f_i koji su sa njim kolinearni i istog smera, a računski jednostavniji, čime se dobija novo pravilo za izračunavanje koraka:

$$x_{k+1} = x_k - \alpha_k \nabla f_i(x_k)$$

Obično se i bira tako da bude jednako $(k \ mod \ N) + 1$, odnosno tako da se u svakom koraku koristi naredna funkcija f_i, dok se ne dođe do poslednje, a onda se nastavlja

ponovo od prve.

Zahvaljujući prikazanoj aproksimaciji gradijenta, ova metoda može izbeći lokalne minimume. Takođe, računski je manje zahtevna od gradijentnog spusta, što vodi bržoj konvergenciji i čini je pogodnom za veće skupove podataka.

3 *Iskazna logika, SAT problem i SAT rešavači*

3.1 Uvod

SAT problem je problem ispitivanja iskazne zadovoljivosti formula u *konjunktivnoj normalnoj formi (KNF)* i jedan je od najznačajnijih problema teorijskog računarstva. Današnji *SAT* rešavači su u stanju da ispitaju zadovoljivost formula u KNF koje sadrže stotine hiljada promenljivih i milione klauza. Zahvaljujuci ovome, SAT rešavači se uspešno primenjuju u oblastima kao sto su raspoređivanje, planiranje, verifikacija hardvera, verifikacija softvera, itd.

3.2 Iskazna logika - osnovni pojmovi

Pretpostavimo da je dat prebrojiv skup *iskaznih promenljivih*. *Iskazne formule* su formule koje se grade nad iskaznim promenljivim (atomima ili iskaznim slovima) i logičkim konstantama ($\top, \bot$) pomoću logičkih veznika negacije ($\neg$), konjunkcije ($\wedge$), disjunkcije ($\vee$), implikacije ($\Rightarrow$) i ekvivalencije ($\Leftrightarrow$). *Literal* je ili promenljiva ili njena negacija. *Suprotan literal* literala l označavamo sa $\bar{l}$. Ukoliko je literal l promenljiva x_i, njemu suprotan literal je negacija promenljive $\neg x_i$, a ukoliko je literal l negacija promenljive $\neg x_i$, njemu suprotan literal je sama promenljiva x_i. Formula je u *negacionoj normalnoj formi (NNF)* akko je sastavljena od literala korišćenjem isključivo veznika $\wedge$ i $\vee$ ili je logička konstanta ($\top$ ili $\bot$). *Klauza* predstavlja disjunkciju literala. Formula je u *konjuktivnoj normalnoj formi (KNF)* ako je u obliku konjunkcije klauza. Na primer,

$$(x_1 \vee \neg x_2) \wedge (\neg x_3 \vee x_4).$$

Formula je u *disjunktivnoj normalnoj formi (DNF)* ako je u obliku disjunkcije konjukcija literala. Na primer,

$$(x_1 \wedge \neg x_2) \vee (\neg x_3 \wedge x_4).$$

Valuacija je funkcija koja iskaznim promenljivim pridružuje istinitosne vrednosti. Tehnički pogodna reprezentacija valuacije je u vidu liste (konačne sekvence) literala. Valuacija je *neprotivrečna* ukoliko ne sadrži dva međusobno suprotna literala. Svaka neprotivrečna valuacija jednoznačno određuje preslikavanje nekog skupa promenljivih u skup istinitosnih vrednosti (npr. u skup $\{0, 1\}$). Literal l je *tačan u valuaciji* v ukoliko je njen član, *netačan u valuaciji* v ukoliko je njemu suprotan literal član valuacije v, a nedefinisan inače. Klauza je *tačna u valuaciji* v ukoliko je bar jedan njen literal tačan u valuaciji v, a *netačna u valuaciji* v ukoliko su svi njeni literali netačni u v. Vrednost formule pri datoj valuaciji se može utvrditi na osnovu svojstava iskaznih veznika. Formula F u KNF je *tačna u valuaciji* v ukoliko su joj sve klauze tačne u valuaciji v, a *netačna u valuaciji* v ukoliko joj je bar jedna klauza netačna u valuaciji v. Za valuaciju kažemo da je *model* formule (klauze, literala) ukoliko je neprotivrečna i ukoliko je formula (klauza, literal) u njoj tačna. Formula je *zadovoljiva* akko ima model. Formula je *tautologija* ako joj je svaka neprotivrečna valuacija model.

3.3 SAT problem

Jedan od centralnih problema u iskaznoj logici je ispitivanje da li je data iskazna formula u KNF zadovoljiva. Ovaj problem poznat je kao *SAT problem (eng. satisfiability problem)*. *SAT* je prvi problem za koji je dokazano da je *NP-kompletan* [6] i za njega postoje ogromne praktične primene (raspored časova, rasporedi na aerodromima, dizajn elektronskih kola, verifikacija hardvera, verifikacija softvera, ...). S obzirom na to da se još uvek ne zna da li su klase *P* i *NP* problema jednake, još uvek se ne zna da li postoji algoritam za ispitivanje zadovoljivosti koji je polinomijalne složenosti. Kako je opšte uverenje da su klase ove klase problema različite, veruje se i da ne postoji polinomijalni algoritam za rešavanje *SAT* problema.

Najpoznatiji pristupi rešavanja SAT problema:

- naivni metodi (metod istinitosnih tablica [3.3.1])
- ukoliko je formula u DNF zadovoljivost se trivijalno ispituje, ali je postupak prevodenja u DNF netrivijalan [4.1.1]. Postoji efikasan postupak prevođenja formule u KNF linearne vremenske i prostorne složenosti koji čuva zadovoljivost [3.3.4]. Zadovoljivost formula u KNF se dalje ispituje nekim od sledećih algoritama:
  - *DP* procedura [7]. Ova procedura koristi iskaznu rezoluciju. Problem je bio u brzom popunjavanju memorije, zbog čega se brzo odustalo od nje.
  - *DPLL* procedura [3.3.6]. DPLL procedura ne koristi rezoluciju, već se zasniva na pretrazi u prostoru valuacija. Zauzima manje memorije od DP procedure, pa je bila glavno sredstvo do sredine 1990-ih godina.
  - *CDCL SAT* rešavači[3.5]. Predstavlja unapređenu DPLL proceduru koja ponovo u određenom obliku uključuje rezoluciju.
  - *Stohastički SAT* rešavači [3.5]. Predstavljaju vid stohastičke optimizacije. Ne mogu ustanoviti nezadovoljivost, ali u slučaju zadovoljivih formula mogu biti neuporedivo brži.

3.3.1 Istinitosne tablice

Istinitosne tablice predstavljaju metod koji se svodi na iscrpno ispitivanje valuacija. Vrednost formule je odredena vrednošću njenih iskaznih promenljivih. Za ispitivanje tautologičnosti (zadovoljivosti) formule dovoljno je ispitati konačno mnogo valuacija. Ukoliko formula sadrži n različitih iskaznih slova, onda je dovoljno ispitati 2^n različitih valuacija. Tabela 3.1 ilustruje metod ispitivanja tautologičnosti istinitosnom tablicom.

TABELA 3.1: Istinitosna tablica za formulu $x_1 \wedge (x_2 \vee \neg x_3)$

x_1	$\wedge$	$(x_2$	$\vee$	$\neg$	$x_3)$
0	0	0	1	1	0
0	0	0	0	0	1
0	0	1	1	1	0
0	0	1	1	0	1
1	1	0	1	1	0
1	0	0	0	0	1
1	1	1	1	1	0
1	1	1	1	0	1

3.3.2 Elementarne logičke ekvivalencije za pojednostavljivanje formula

Naredne logičke ekvivalencije se mogu upotrebiti za pojednostavljivanje formula.

$$\neg\bot \equiv \top \quad \neg\top \equiv \bot$$

$$x \wedge \bot \equiv \bot \quad \bot \wedge x \equiv \bot$$

$$x \wedge \top \equiv x \quad \top \wedge x \equiv x$$

$$x \vee \bot \equiv x \quad \bot \vee x \equiv x$$

$$x \vee \top \equiv \top \quad \top \vee x \equiv \top$$

$$x \Rightarrow \bot \equiv \neg x \quad \bot \Rightarrow x \equiv \top$$

$$x \Rightarrow \top \equiv \top \quad \top \Rightarrow x \equiv x$$

$$x \Leftrightarrow \bot \equiv \neg x \quad \bot \Leftrightarrow x \equiv \neg x$$

$$x \Leftrightarrow \top \equiv x \quad \top \Leftrightarrow x \equiv x$$

Pojednostavljivanje formule se često u literaturi naziva *simplifikacija*. Na primer, simplifikacijom formule

$$\neg(x_1 \wedge (x_2 \vee \top)) \Leftrightarrow (\top \Rightarrow x_3)$$

dobija se

$$\neg x_1 \Leftrightarrow x_3.$$

Često se pretpostavlja da je formula pojednostavljena prikazanim pravilima pre daljih transformacija u normalne forme.

3.3.3 Transformacije u NNF, DNF i KNF

Za transformaciju formule u NNF koristimo dva skupa pravila. Prvi skup pravila, uklanjanje implikacija i ekvivalencija, dato je sledećim pravilima:

$$x \Rightarrow y \equiv \neg x \vee y$$

$$\neg(x \Rightarrow y) \equiv x \wedge \neg y$$

$$x \Leftrightarrow y \equiv (x \vee \neg y) \wedge (\neg x \vee y)$$

$$\neg(x \Leftrightarrow y) \equiv (x \vee y) \wedge (\neg x \vee \neg y)$$

Drugi skup pravila, spuštanje negacije, dato je sledećim pravilima:

$$\neg(x \wedge y) \equiv \neg x \vee \neg y$$

$$\neg(x \vee y) \equiv \neg x \wedge \neg y$$

$$\neg\neg x \equiv x$$

Razmotrimo složenost NNF transformacije. NNF formula sa n veznika može da sadrži, u najgorem slučaju, više od 2^n veznika. Eksponencijalno uvećanje je posledica uklanjanja ekvivalencije.

Primenom sledećih logičkih ekvivalencija NNF formula se može prevesti u KNF:

$$x \vee (y \wedge z) \equiv (x \vee y) \wedge (x \vee z)$$

$$(y \wedge z) \vee x \equiv (y \vee x) \wedge (z \vee x)$$

Primenom sledećih logičkih ekvivalencija NNF formula se može prevesti u DNF:

$$x \wedge (y \vee z) \equiv (x \wedge y) \vee (x \wedge z)$$

$$(y \vee z) \wedge x \equiv (y \wedge x) \vee (z \wedge x)$$

Prikazana procedura se zove *prevođenje prezapisivanjem distributivnosti* i njena korektnost se zasniva na *teoremi o zameni* [9]. Ukoliko razmotrimo složenost procedure, primetićemo da pravila distributivnosti doprinose eksponencijalnom uvećavanju formule koja je bazirana na NNF, što proceduru čini praktično neupotrebljivom kada je reč o zadacima velike dimenzije.

3.3.4 Definiciona (Cajtinova) KNF

U slučaju da je dovoljno da formula u KNF bude ekvizadovoljiva polaznoj formuli, KNF se može izgraditi tako da veličina dobijene formule bude linearna u odnosu na veličinu polazne. Originalna ideja potiče od Cajtina [24], a naknadno je revidirana na mnoge načine od Vilsona [25]. Za potformule se uvode nova iskazna slova, odnosno definicije, od čega i potiče naziv definiciona KNF. Metod se najbolje može razumeti kroz primer. Pretpostavimo da želimo da transformišemo sledeću formulu u KNF:

$$(x \vee (y \wedge \neg z)) \wedge w.$$

Uvodimo novo iskazno slovo x_1, koje se ne pojavljuje u polaznoj formuli, kao skraćenicu za potformulu $y \wedge \neg z$ i definiciju novog iskaznog slova x_1 pridružujemo polaznoj formuli:

$$(x_1 \Leftrightarrow y \wedge \neg z) \wedge (x \vee x_1) \wedge w.$$

Nastavljamo sa primenom istog postupka, uvodeći novo iskazno slovo x_2, kao skraćenicu za potformulu $x \vee x_1$:

$$(x_1 \Leftrightarrow y \wedge \neg z) \wedge (x_2 \Leftrightarrow x \vee x_1) \wedge x_2 \wedge w$$

i dalje x_3, kao skraćenicu za potformulu $x_2 \wedge w$:

$$(x_1 \Leftrightarrow y \wedge \neg z) \wedge (x_2 \Leftrightarrow x \vee x_1) \wedge (x_3 \Leftrightarrow x_2 \wedge w) \wedge x_3.$$

Definicione ekvivalencije se na uobičajen način prevode u KNF:

$$\begin{aligned}(\neg x_1 \vee y) \wedge (\neg x_1 \vee \neg z) \wedge (x_1 \vee \neg y \vee z) \wedge \\ (\neg x_2 \vee x \vee x_1) \wedge (x_2 \vee \neg x) \wedge (x_2 \vee \neg x_1) \wedge \\ (\neg x_3 \vee x_2) \wedge (\neg x_3 \vee w) \wedge (x_3 \vee \neg x_2 \vee \neg w) \wedge \\ x_3.\end{aligned}$$

Može se primetiti da je ovom transformacijom izbegnuto eksponencijalno uvećanje formule. Broj definicionih ekvivalencija je ograničen brojem podformula polazne formule. S obzirom na jednostavnu formu dodatih ekvivalencija, njihovo klasično prevođenje u KNF dovodi do skromnog proširenja rezultujuće formule. Dokaz ekvizadovoljivosti definicione KNF se može naći u [9].

3.3.5 DIMACS CNF

DIMACS CNF je standardni format zapisa formula u KNF u tekstualne datoteke. Koristi se kao ulaz za SAT rešavače [3.5].

Ulazni fajl u DIMACS CNF formatu može početi proizvoljim brojem komentara. Linija koja predstavlja komentar počinje karakterom `c`. Nakon toga slede informacije o formuli. Broj iskaznih promenljivih (`variables`) i broj klauza (`clauses`) se definiše linijom `p cnf variables clauses`. Svaka od narednih linija predstavlja jednu klauzu: pozitivan literal se predstavlja odgovarajućim celim pozitivnim brojem iz segmenta $[1, variables]$, a negativan literal se predstavlja odgovarajućim negativnim brojem iz segmenta $[-variables, -1]$. Linija se uvek završava brojem 0.

Primer ulaznog fajla za formulu $(x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_1)$:

```
c Primer .cnf fajla
p cnf 3 2
1 -3 0
2 3 -1 0
```

3.3.6 DPLL procedura

DPLL procedura (eng.Davis-Putham-Loveland-Logemann procedure (DPLL)) predstavljena je 1962. [16] i bavi se ispitivanjem zadovoljivosti iskaznih formula u KNF. Predstavlja osnovu savremenih CDCL SAT rešavača [3.5].

DPLL pretraga

Osnovna komponenta DPLL procedure je pretraga za zadovoljavajućom valuacijom. Vrednost nekog literala l se „postavi na tačno", odnosno sva pojavljivanja literala l se zamene logičkom konstantom $\top$, a pojavljivanja njemu suprotnog literala $\bar{l}$ se zamene logičkom konstantom $\bot$ i ispita se zadovoljivost. Ako se ne dobije zadovoljivost, onda se vrednost njemu suprotnog literala $\bar{l}$ „postavi na tačno" i ispita se zadovoljivost. „Postavljanje literala na tačno" dovodi do pojednostavljivanja formule (dok se ne stigne do formule čija se zadovoljivost trivijalno očitava, npr. logičke konstante).

Neka $F[[l \to \top]]$ označava formulu F u kojoj je literal l „postavljen na tačno", a $F[[\bar{l} \to \top]]$ označava formulu F u kojoj je suprotan literal $\bar{l}$ „postavljen na tačno", nakon čega su izvršena elementarna pojednostavljivanja formule (pogledati 3.3.2). Umesto oznake $F[[\bar{l} \to \top]]$ se može koristiti i oznaka $F[[l \to \bot]]$, s obzirom da „postavljanje suprotnog literala $\bar{l}$ na tačno" između ostalog podrazumeva i zamenu svih pojavljivanja literala l logičkom konstantom $\bot$. Pretraga za zadovoljavajućom valuacijom je zasnovana na koraku split *(eng. splitting rule* koji ispituje zadovoljivost formula $F[[l \to \top]]$ i $F[[\bar{l} \to \top]]$.

Kada je reč o trivijalnim slučajevima, važi:

- DPLL pretraga prijavljuje zadovoljivost ukoliko je skup klauza prazan
- DPLL pretraga prijavljuje nezadovoljivost ukoliko skup klauza sadrži praznu klauzu.

Odabir literala za split pravilo je veoma važna odluka za efikasnost cele procedure. Neke varijante ovog pravila su da se bira literal sa najviše pojavljivanja u tekućoj formuli, da se bira neki literal iz najkraće klauze itd.

Elementi zaključivanja. DPLL procedura

Kada se opisana DPLL pretraga proširi sledećim elementima zaključivanja, dobija se klasična DPLL procedura:

- propagacija jediničnih klauza *(eng. unit propagation)*[3.3.6]
- eliminacija „čistih" literala *(eng. pure literal*[3.3.6]

Propagacija jediničnih klauza Ukoliko formula sadrži *jediničnu* klauzu, odnosno klauzu sa tačno jednim literalom $\{l\}$, zaključujemo da l mora biti tačan u zadovoljavajućoj valuaciji i tu informaciju koristimo da bismo uprostili polaznu formulu. Uklanjamo svaku pojavu literala $\bar{l}$ iz ostalih klauza kao i celokupne klauze koje sadrže l, uključujući i jediničnu klauzu.

Eliminacija „čistih" literala Ukoliko se u formuli javlja neki literal l, a ne javlja se njemu suprotan literal $\bar{l}$, onda taj literal može biti postavljen na tačno u zadovoljavajućoj valuaciji, što znači da se mogu ukloniti sve klauze koje sadrže literal l.

Algoritam 1 prikazuje osnovnu strukturu DPLL procedure. U najgorem slučaju, DPLL procedura zahteva eksponencijalni broj koraka u odnosu na veličinu ulazne formule (korak split generiše eksponencijalnost).

Osnovna, rekurzivna varijanta procedure je neefikasna jer zahteva kreiranje nove pojednostavljene formule i njeno prosleđivanje tokom svakog rekurzivnog poziva, što je jako zahtevno s obzirom na to da formule mogu biti veoma veliki objekti.

Algoritam 1: Osnovna struktura DPLL procedure

Ulaz: skup klauza F
Izlaz: $SAT/UNSAT$

```
1  function DPLL(F)
2  if F prazan skup klauza then
3  |  return SAT;
4  else if postoji prazna klauza u F then
5  |  return UNSAT;
6  else if postoji jedinična klauza [l] u F then
7  |  return DPLL(F[[l → ⊤]]);
8  else if postoji čist literal l u F then
9  |  return DPLL(F[[l → ⊤]]);
10 else
11 |  izabrati literal l iz F;
12 |  if DPLL(F[[l → ⊤]] = SAT then
13 |  |  return SAT;
14 |  else
15 |  |  return DPLL(F[[l → ⊥]] ;
```

3.3.7 Iterativna DPLL procedura

Opisanu DPLL proceduru je moguće modelovati korišćenjem skupa pravila prelaza koja opisuju kako se može preći iz stanja u stanje. Za razliku od rekurzivne implementacije, savremeni SAT rešavači tokom procesa rešavanja ne menjaju formulu F već eksplicitno čuvaju tekuću parcijalnu valuaciju M i koriste sledeći skup pravila prelaza:

Decide pravilo:

$$\frac{l \in F \quad l, \bar{l} \notin M}{M := M|l}$$

UnitPropagate pravilo:

$$\frac{l \vee l_1 \vee \ldots \vee l_k \in F \quad \bar{l}_1, \ldots, \bar{l}_k \in M \quad l, \bar{l} \notin M}{M := M\ l}$$

Backtrack pravilo:

$$\frac{M \models \neg F \quad M = M^{'} | l M^{''} \;\; decisions\; M^{''} = [\,]}{M := M^{'}\; \bar{l}}$$

Pravilo *Decide* dodaje novi literal l iz formule F u tekuću parcijalnu valuaciju M, odnosno dodaje literal l kao pretpostavljeni literal. Pretpostavljeni literali su označeni simbolom | sa svoje leve strane. Pravilo *UnitPropagate* koristi jedinične klauze kao mehanizam zaključivanja. Međutim, pošto se prilikom nerekurzivne implementacije procedure formula F i njene klauze ne menjaju, ovde se pojam jedinična klauza menja. Klauza je jedinična u odnosu na valuaciju v ako su joj svi literali netačni u valuaciji v, osim jednog literala koji je nedefinisan u valuaciji v. Da bi jedinična klauza bila tačna u valuaciji njen jedinični literal mora biti tačan i zato ga pravilo UnitPropagate dodaje u valuaciju kao izvedeni literal. Ukoliko formula F postane netačna u tekućoj valuaciji M, kažemo da je došlo do konflikta. Ako se lista pretpostavljenih literala iz valuacije M označi sa $decisions\; M$, literal l je poslednji pretpostavljen literal iz valuacije M, ukoliko važi da je valuacija oblika $M = M^{'} | l M^{''}$, za $decisions\; M^{''} = [\,]$. U slučaju konflikta, pravilo $Backtrack$ uklanja poslednji pretpostavljeni literal l i sve izvedene literale iza njega iz valuacije M i menja pretpostavku tako što u valuaciju dodaje kao izvedeni literal $\bar{l}$.

TABELA 3.2: Primer izvršavanja DPLL procedure

$F = \{\{-1,+2\},\{-3,+4\},\{5,-6\},\{+6,-5,-2\}\}$	
Primenjeno pravilo	M
	$[\,]$
Decide $(I = +1)$	$[\mid +1]$
UnitPropagate $(c = \{-1,+2\},\ I = +2)$	$[\mid +1, +2]$
Decide $(I = +3)$	$[\mid +1, +2, \mid +3]$
UnitPropagate $(c = \{-3,+4\},\ I = +4)$	$[\mid +1, +2, \mid +3, +4]$
Decide $(I = +5)$	$[\mid +1, +2, \mid +3, +4 \mid +5]$
UnitPropagate $(c = \{-5,-6\},\ I = -6)$	$[\mid +1, +2, \mid +3, +4 \mid +5, -6]$

U tabeli 3.2 je prikazan primer izvršavanja DPLL procedure za skup klauza F (literal x_i je predstavljen brojem $+i$, a njemu suprotan literal $\bar{x}_i$ brojem $-i$), gde M predstavlja tekuću parcijalnu valuaciju. Ukoliko analiziramo dati primer, uočićemo konflikt nakon poslednje primene pravila UnitPropagate. Klauza $\{-5,-6\}$ je postala tačna nakon pretpostavke -6, čime je u isto vreme klauza $\{+6,-5,-2\}$ postala netačna. Da bismo nastavili proceduru potrebno je poništiti neke odluke.

Rešenje ovakvih problema se ogleda u uvođenju *analize konflikata (eng. conflict analysis)* i *povratnih skokova (eng. backjumping)*. Analizom konflikta treba ustanoviti koje su odluke bile relevantne za konflikt, a zatim se povratnim skokom vratiti dublje u pretragu.

Izvršimo analizu konflikta na primeru prikazanom u tabeli 3.2. Prvo treba ustanoviti zbog čega je došlo do konflikta. Vidimo da je do konflikta došlo zbog literala $+2$, $+5$ i -6. Formiramo konfliktnu klauzu $c = \{+2,+5,-6\}$. Zatim krećemo otpozadi i zapitamo se zbog čega je prisutan literal -6. Na osnovu pravila UnitPropagate vidimo da je -6 prisutan zbog literala $+5$. Sada je $c = \{+2,+5\}$. Nema potrebe da se sada zapitamo za $+5$ jer je on dobijen pravilom Decide. Dakle, uz literal $+2$, ne ide literal $+5$. Zbog ovog saznanja možemo da dodamo novi korak Backjump na

kraj i nastaviti proceduru sa naučenom klauzom $\{-2, -5\}$, pomoću koje se dolazi do zadovoljavajuće valuacije $M = \{+1, +2, -5, +3, +4\}$ (tabela 3.3). Primena Backtrack pravila isto može razrešiti određene konflikte. Međutim, ovo pravilo uvek menja samo poslednju pretpostavku što ga čini manje efikasnim od pravila Backjump. Takođe, da bi se sprečilo eventualno javljanje iste vrste nepotrebnog ponavljanja postupka, ali u drugom kontekstu, uvodi se *učenje klauza iz ranijih konflikata (eng. clause learning)* .

TABELA 3.3: Primer nastavka izvršavanja DPLL procedure nakon učenja klauze

$F = \{\{-1, +2\}, \{-3, +4\}, \{5, -6\}, \{+6, -5, -2\}\}$	
Naučena klauza $c = \{-2, -5\}$	
Tekuća valuacija $M = [\| + 1, +2]$	
Primenjeno pravilo	M
UnitPropagate $(c = \{-2, -5\},\ I = -5)$	$[\| + 1, +2, -5]$
Decide $(I = +3)$	$[\| + 1, +2, -5, \| + 3]$
UnitPropagate $(c = \{-3, +4\},\ I = +4)$	$[\| + 1, +2, -5 \| + 3, +4]$

Sistem sa analizom konflikata predstavlja proširen sistem DPLL procedure koji se još naziva i *CDCL (eng. Conflict-Driven Clause Learning (CDCL))* sistem za SAT (njega koriste CDCL SAT rešavači [3.5]) i sastoji iz sledećih pravila.

Decide:

$$\frac{l \in F \quad l, \bar{l} \notin M}{M := M|l}$$

UnitPropagate:

$$\frac{l \vee l_1 \vee \ldots \vee l_k \in F \quad \bar{l}_1, \ldots, \bar{l}_k \in M \quad l, \bar{l} \notin M}{M := M\ l}$$

Conflict:

$$\frac{l_1 \vee \ldots \vee l_k \in F \quad \bar{l}_1, \ldots, \bar{l}_k \in M}{C := l_1 \vee \ldots \vee l_k}$$

Explain:

$$\frac{\bar{l} \in C \quad l \vee l_1 \vee \ldots \vee l_k \in F \quad \bar{l}_1, \ldots, \bar{l}_k \prec^M l}{C := (C \backslash \bar{l})\ l_1 \vee \ldots \vee l_k}$$

Backjump:

$$\frac{C = l \vee l_1 \vee \ldots \vee l_k \quad C \in F \quad level\ \bar{l} > m \geq level\ \bar{l}_i}{M := (prefixToLevel\ m\ M)\ l}$$

Analiza konflikta započinje konfliktnom klauzom C, odnosno klauzom koja je postala netačna u tekućoj valuaciji M. Literali iz C koji su netačni u M su dodati u M kao

pretpostavljeni ili izvedeni. Pravilo $Explain$ pronalazi klauzu iz F koja je bila razlog izvođenja nekog literala iz C i taj literal menja ostalim literalima iz klauze koja je bila razlog izvođenja. Izraz $\bar{l}_1, \ldots, \bar{l}_k \prec^M l$ označava da se literali $\bar{l}_1, \ldots, \bar{l}_k$ nalaze pre literala l u M. Nivo literala l u valuaciji M, u oznaci $level\ l\ M$ (nekada se M podrazumeva), je broj pretpostavljenih literala iz valuacije M koji prethode prvoj pojavi literala l, uključujući i l ukoliko je pretpostavljen. Prefiks do datog nivoa m valuacije M, u oznaci $prefixToLevel\ m\ M$, je prefiks valuacije M koji sadrži sve literale iz M čiji su nivoi manji ili jednaki od m. Primena pravila $Backjump$ zavisi od klauze povratnog skoka C. Mora biti ispunjeno da u M postoji literal $\bar{l}$ suprotan literalu l iz C, takav da se svi ostali literali $\bar{l}_i$ suprotni literalima iz C nalaze u M na nižem nivou od $\bar{l}$. Ukoliko se sa m označi nivo u M, takav da je $level\ \bar{l} > m \geq level\ \bar{l}_i$, pravilo $Backjump$ poništava sve literale iz M sem onih iz $prefixToLevel\ m\ M$ i u valuaciju dodaje l.

Učenje predstavlja dodavanje naučene klauze u početni skup klauza. Prethodni sistem se upotpunjuje pravilom *Learn* čime se dobija *sistem sa učenjem* u kome se početna formula F vremenom proširuje. Obično se uče isključivo klauze povratnog skoka.

Learn:

$$\frac{F \models c}{F := F \wedge c}$$

Zaboravljanje Ukoliko broj naučenih klauza postane preveliki, pronalaženje jediničnih i konfliktnih klauza se usporava. Neke naučene klauze se posle odredenog vremena uklanjaju. Zaboravljanje je kontrolisano heuristikama.

Otpočinjanje iznova Nekada klauze koje su u međuvremenu naučene mogu odvesti u lakšu granu stabla pretrage. Otpočinjanje iznova je takođe kontrolisano heuristikama.

Izbor promenljive za pravilo Decide Aktivnosti promenljivih u konfliktima, analizi konflikata i jediničnim propagacijama čine njihov skor. Skor promenljive se uvećava kad god ona učestvuje u nekom od pomenutih procesa. Prilikom izbora promenljive, bira se najaktivnija promenljiva.

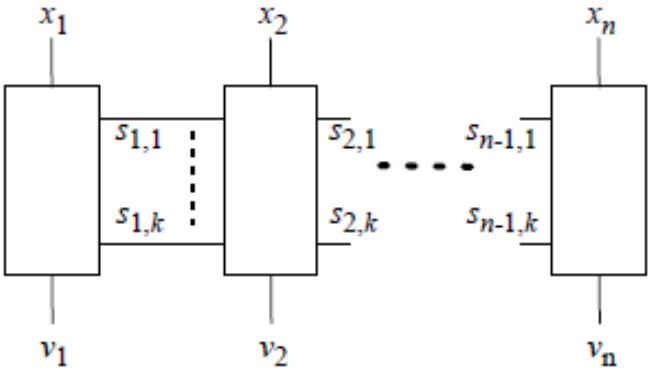
3.4 Ograničenja kardinalnosti

Ograničenja kardinalnosti (eng. cardinality constraints) predstavljaju broj tačnih promenljivih iz nekog skupa. Mnogi realni problemi uključuju različita ograničenja kardinalnosti. Rešavanje tih problema pomoću SAT rešavača, zahteva opis ovih ograničenja u obliku formule u KNF, tj. skupa iskaznih klauza. $atMost(k, x_1, x_2, \ldots, x_n)$ označava da najviše k od n promenljivih $x_1, x_2, \ldots, x_n$ istovremeno može biti *tačno*, dok $atLeast(k, x_1, x_2, \ldots, x_n)$ označava da najmanje k od n promenljivih $x_1, x_2, \ldots, x_n$ istovremeno može biti *tačno*. Potrebno je naći efikasan način da se ova ograničenja opišu formulama u KNF, tako da se koristi što manji broj klauza i dodatnih promenljivih. S obzirom da važi:

$$atLeast(k, x_1, x_2, \ldots, x_n) \equiv atMost(n - k, \neg x_1, \neg x_2, \ldots, \neg x_n). \tag{3.1}$$

dovoljno je prikazati prevođenje ograničenja $atMost$ u KNF.

Biće opisana dva načina za definisanje formule u KNF ograničenja kardinalnosti, koji su predstavljeni u [23].



SLIKA 3.1: Sekvencijalni brojač

3.4.1 Formula ograničenja kardinalnosti zasnovana na sekvencijalnom brojaču

Formula kojom se definiše ograničenje kardinalnosti oblika $atMost(k, x_1, x_2, \ldots, x_n)$ se može konstruisati na osnovu sekvencijalnog brojača. Sekvencijalni brojač je kombinatorno kolo (videti 4) koje za svoje ulazne promenljive $x_1, x_2, \ldots, x_n$, računa parcijalne sume oblika $s_i = \sum_{j=1}^{i} x_j$, gde i uzima vrednosti od 1 do n.

Vrednost parcijalne sume s_i se predstavlja pomoću iskaznih promenljivih $s_{i,j}$, $j = 1, \ldots, k$ koje se postavljaju na *tačno* ukoliko je $s_i \geq j$, za $j = 1, \ldots, k$, a odgovarajući bit prekoračenja v_i se postavlja na *tačno* ukoliko je parcijalna suma s_i veća od k (slika 3.1). Konstrukcijom odgovarajućih iskaznih formula za parcijalne sume i bitove prekoračenja, njihovim pojednostavljivanjem (primetimo da bitovi prekoračenja treba da budu postavljeni na *netačno* za svaku parcijalnu sumu) i konačno, prevodjenjem u KNF Cajtinovom tranformacijom, dobija se sledeći skup klauza:

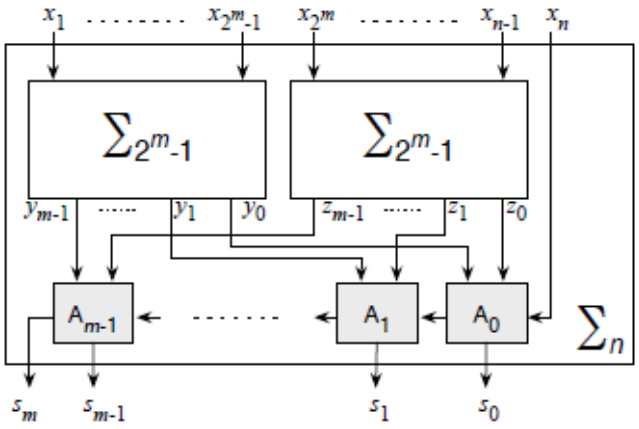
$$
\begin{aligned}
(\neg x_1 \vee s_{1,1}) & \\
(\neg s_{1,j}) & \quad za 1 < j \leq k \\
(\neg x_i \vee s_{i,1}) & \quad za 1 < i < n \\
(\neg s_{i-1,1} \vee s_{i,1}) & \quad za 1 < i < n \\
(\neg x_i \vee \neg s_{i-1,j-1} \vee s_{i,j}) & \quad za 1 < j \leq k, 1 < i < n \\
(\neg s_{i-1,j} \vee s_{i,j}) & \quad za 1 < j \leq k, 1 < i < n \\
(\neg x_i \vee \neg s_{i-1,k}) & \quad za 1 < i < n \\
(\neg x_n \vee \neg s_{n-1,k}) &
\end{aligned}
$$

koji je zasnovan na sekvencijalnom brojaču i definiše ograničenje $atMost(k, x_1, x_2, \ldots, x_n)$, za $k > 0, n > 1$ (detalji se mogu naći u [23]). Označimo dobijeni skup sa $atMost_{SEQ}^{n,k}$.

$atMost_{SEQ}^{n,k}$ sadrži $2nk + n - 3k - 1$, tj. $\mathcal{O}(nk)$ klauza i zahteva $(n-1)k$, odnosno $\mathcal{O}(nk)$ pomoćnih promenljivih. Ova forma se pokazala efikasnom za male vrednosti k jer se korišćenjem pravila jedinične propagacije (u linearnom vremenu) može zaključiti da neka valuacija ne zadovoljava ograničenje kardinalnosti, tj. postavlja više od k promenljivih x_i na tačno ili, u slučaju da delimična valuacija već postavlja k promenljivih x_i na *tačno*, istim pravilom se mogu dobiti vrednosti preostalih promenljivih x_i.

3.4.2 Formula ograničenja kardinalnosti zasnovana na paralelnom brojaču

Korišćenjem paralelnog brojača može se dobiti formula za ograničenje kardinalnosti oblika $atMost(k, x_1, x_2, \ldots, x_n)$ pogodnija za velike vrednosti k i n. Princip rada paralelnog brojača je sledeći [20]: ulazne promenljive x_i se dele na dva dela $x_1, \ldots, x_{2^m-1}$ i $x_{2^m}, \ldots, x_{n-1}$, za $m = \lfloor \log_2 n \rfloor$, zatim se rekurzivno za svaki deo računa broj promenljivih koje su postavljene na *tačno*, nakon čega se dobijeni rezultati, označeni sa $y_{m-1} \ldots y_0$ za prvi deo i sa $z_{m'-1} \ldots z_0$ za drugi deo, i poslednji ulaz x_n sabiraju korišćenjem m-bitnog sabirača (videti 4.1.4). Primetimo da je broj izlaza m' jednak $\lceil \log_2(n+1-2^m) \rceil$ od čega zavisi konfiguracija m-bitnog sabirača. Na slici 3.2 je prikazana shema paralelnog brojača za slučaj $m' = m$, kada se m-bitni sabirač gradi pomoću m punih sabirača $A_{m-1}, \ldots, A_0$. U slučaju da je $m' < m$, za izgradnju m-bitnog sabirača je dovoljno iskoristiti m' punih sabirača i $m - m'$ polusabirača.



SLIKA 3.2: Paralelni brojač

Uzimajući u obzir sve slučajeve, paralelni brojač se može implementirati pomoću $n - \lfloor \log n \rfloor - 1$ punih sabirača i najviše $\lfloor \log n \rfloor$ polusabirača. Na osnovu tabelarnih formi ovih kombinatornih kola prikazanih u 4.1.4, dobijamo odgovarajuće formule KNF:

- za polusabirač (tri klauze)

$$(a \vee \neg b \vee s_{out})(\neg a \vee \neg b \vee c_{out}) \wedge (\neg a \vee b \vee s_{out})$$

- za potpun sabirač (sedam klauza)

$$(a \vee b \vee \neg c \vee s_{out}) \wedge (\neg a \vee b \vee c \vee s_{out}) \wedge (\neg a \vee \neg b \vee c_{out}) \wedge$$
$$(a \vee \neg b \vee c \vee s_{out}) \wedge (\neg a \vee \neg b \vee \neg c \vee s_{out}) \wedge (\neg a \vee \neg c \vee c_{out}) \wedge$$
$$(\neg b \vee \neg cc_{out})$$

Ukupno, za predstavljanje paralelnog brojača, potrebno je najviše $7n - 4\lfloor \log_2 n \rfloor - 7$ klauza. Dodatno, za svaki potpun i polusabirač potrebne su pomoćne promenljive za sumu i prenos, što ukupno zahteva najviše $2(n-1)$ pomoćnu promenljivu.

Da bi paralelni brojač mogao da se iskoristi za kreiranje formule ograničenja $atMost$, njegov rezultat se prosleđuje komparatoru (kolu koje poredi dva niza bitova)

koji proverava da li je rezultat manji od zadatog k. S obzirom da je prilikom konstrukcije paralelnog brojača korišćen m-bitni sabirač, pretpostavimo da je rezultat u obliku m-bitnog binarnog broja $s_m s_{m-1} \dots s_0$. Takođe, neka je granica k zadata u obliku $(m+1)$-bitnog binarnog broja $k_m k_{m-1} \dots k_0$ (k se može predstaviti u ovom obliku jer je $m = \lfloor \log_2 n \rfloor$, a $k \leq n$). Tada, $(m+1)$-bitni komparator koji proverava da li je rezultat paralelnog brojača manji od zadatog k, možemo predstaviti skupom klauza, u oznaci $C(k_m k_{m-1} \dots k_0)$, koji se rekurzivno generiše na sledeći način:

$$C(k_0) = \begin{cases} \{\{\neg s_0\}\} & ako\ je\ k_0 = 0 \\ \emptyset & ako\ je\ k_0 = 1 \end{cases}$$

$$C(k_i \dots k_0) = \begin{cases} \{\{\neg s_0\}\} \cup C(k_{i-1} \dots k_0) & ako\ je\ k_i = 0 \\ \{\{\neg s_0\}\} \otimes C(k_{i-1} \dots k_0) & ako\ je\ k_0 = 1 \end{cases}$$

gde $\otimes$ predstavlja množenje klauza, tj. $M \otimes N = \{a \cup b \mid a \in M, b \in N\}$, za skupove klauza M i N. $C(k_m k_{m-1} \dots k_0)$ će imati najviše $m+1$ klauzu.

Označimo sa $atMost^{n,k}_{PAR}$ uniju skupa klauza potrebnih za definisanje paralelnog brojača i skupa klauza komparatora $C(k_m k_{m-1} \dots k_0)$. $atMost^{n,k}_{PAR}$ predstavlja skup klauza formule u KNF za ograničenje kardinalnosti $atMost(k, x_1, x_2, \dots, x_n)$ sa najviše $7n - 3\lfloor \log_2 n \rfloor - 6$ klauza i $2n - 2$ pomoćne promenljive.

Poređenjem broja klauza i pomoćnih promenljivih koje zahtevaju $atMost^{n,k}_{SEQ}$ (3.4.1) i $atMost^{n,k}_{PAR}$ zaključujemo da je $atMost^{n,k}_{PAR}$ efikasnije za veće vrednosti n i k, iako zahteva pretragu da bi proverio da li je ograničenje kardinalnosti zadovoljeno ili ne, za razliku od $atMost^{n,k}_{SEQ}$ koji to može proveriti jediničnom propagacijom. U radu [23] se mogu naći detaljnija poređenja sa još nekim do tada poznatim načinima predstavljanja ograničenja kardinalnosti. Daljim razmatranjem specijalnog slučaja formule $atMost^{n,k}_{SEQ}$ za $k = 1$ razvijene su tehnike za unapređenje SAT rešavača pomoću kojih rešavač može da ignoriše pomoćne promenljive uvedene u $atMost^{n,1}_{SEQ}$ i eliminiše ih iz naučenih klauza čime se smanjuje utrošak memorije i potrebno procesorsko vreme za rad [18]. U radu [22] se predlaže formula ograničenja kardinalnosti $atMost(k, x_1, x_2, \dots, x_n)$ zasnovana na tzv. mrežama kardinalnosti za koje je potrebno $\mathcal{O}(n \log^2 k)$ klauza i $\mathcal{O}(n \log^2 k)$ pomoćnih promenljivih. Poboljšanje ove ideje uvođenjem mreža za sortiranje i objedinjavanje je prikazano par godina kasnije u [10].

3.5 SAT rešavači

Postoje dve osnovne vrste SAT rešavača [21], potpuni i stohastički. Potpuni rešavači mogu da utvrde za bilo koju formulu da li je zadovoljiva ili ne, dok stohastički to mogu da utvrde samo za zadovoljivu, često brže nego što bi to utvrdili potpuni rešavači, ali ne mogu da utvrde da je nezadovoljiva.

CDCL SAT rešavači Većina potpunih SAT rešavača se zasniva na DPLL proceduri. Moderni SAT rešavači se nazivaju i *CDCL (eng. Conflict-Driven Clause Learning (CDCL))* rešavači jer se zasnivaju na sistemu za analizu konflikata koji u osnovi ima DPLL proceduru. Neki od najznačajnijih CDCL SAT rešavača su MiniSAT, Zchaff SAT, Z3 i drugi.

Stohastički SAT rešavači Jedan od načina rada stohastičkih rešavača je gramziva lokalna pretraga. Uzima se slučajna valuacija i razmatraju se sve valuacije koje se od izabrane razlikuju samo u vrednosti od najviše jedne promenljive. Od razmatranih valuacija bira se ona koja zadovoljava najveći broj klauza, a postupak se ponavlja

dok se ne nađe valuacija koja zadovoljava celu početnu formulu. Neki od najznačajnijih stohastičkih SAT rešavača su WalkSAT, GSAT i SAPS.

Promene u parametrima SAT rešavača, korišćcenim heuristikama ili u izboru samog SAT rešavača vode drastičnim razlikama u vremenu rešavanja iste formule. Zbog toga je korisno raspolagati većim brojem SAT rešavača i njihovih konfiguracija.

3.5.1 Portfolio sistemi za SAT

Portfolio sistemi za SAT su sistemi koji na neki način upošljavaju veći broj rešavača ili njihovih konfiguracija za rešavanje iskazne formule. Mogu se smatrati standardom u rešavanju praktičnih problema.

Pristupi:

- izbor jednog od nekoliko SAT rešavača
- pokretanje različitih SAT rešavača ili njihovih konfiguracija
- paralelno pokretanje uz razmenu naučenih klauza

SATzilla je najpoznatiji portfolio sistem za SAT [15]. Ovaj sistem bira SAT rešavač za ulaznu formulu pomoću procene vremena koje je potrebno za njeno rešavanje. Statistički model na osnovu kojeg se vrši procena je prethodno ocenjen metodama mašinskog učenja zbog čega se iskazne formule predstavljaju vektorima atributa. Sistem je vrlo komplikovan i mana mu je što nepotrebno rešava problem procene vremena umesto da direktno rešava problem izbora rešavača. Sistem koji se pokazao pogodnijim za izbor konfiguracije SAT rešavača je **ArgoSmart**, zasnovan na algoritmu k najbližih suseda [17].

3.5.2 MAX-SAT rešavači

MAX-SAT (od eng. maximum satisfiability problem) je problem pronalaženja valuacije koja zadovoljava maksimalan broj klauza date formule u KNF. Ovaj problem predstavlja uopštenje SAT problema. MAX-SAT rešavači su se pokazali pogodnim za mnoge realne probleme kod kojih su prihvatljiva rešenja sa određenim ograničenjima.

Postoje mnogobrojna istraživanja na temu transformacije tehnologija kreiranih za SAT u MAX-SAT. Jedan od najpoznatijih kreiranih MAX-SAT rešavača je MaxSatz [4]. U poglavlju 3.5.2 će biti prikazan najčešće korišćen algoritam za MAX-SAT, koji koristi i pomenuti MaxSatz rešavač.

Neka od značajnih proširenja MAX-SAT problema:

- *WMAX-SAT (od eng. Weighted MAX-SAT)* radi sa težinskim klauzama i pronalazi maksimalnu težinu koja može biti zadovoljena nekom valuacijom, primer rešavača je Max-DPLL [11] koji je rešio mnoge instance koje su bile nedostižne rešavačima koji su postojali pre njega.
- *PMAX-SAT (od eng. Partial MAX-SAT)* (videti poglavlje 3.5.3), primer rešavača je PMS [2].
- *Soft-SAT* radi sa skupom SAT problema i pronalazi maksimalni podskup koji može biti zadovoljen nekom valuacijom, primeri rešavača su Soft-SAT-S i Soft-SAT-D, predstavljeni u [1].

Algoritam grananja i ograničavanja za MAX-SAT

Jedna od osnovnih razlika SAT i MAX-SAT rešavača je ta što SAT rešavači intenzivno koriste pravilo propagacije jediničnih klauza kojim se polazna SAT instanca F (skup klauza) transformiše u njoj ekvizadovoljivu $F^{'}$, jednostavniju za rešavanje. Međutim, rešavanje MAX-SAT problema za F, u opštem slučaju nije ekvivalentno rešavanju istog problema za $F^{'}$, tj. broj nezadovoljenih klauza u F i $F^{'}$ nije isti za svaku valuaciju (ovo ćemo koristiti kao definiciju ekvivalentnosti u domenu MAX-SAT problema). Na primer, primenom pravila propagacije jedinične klauze nad skupom klauza $F = \{x_1, \bar{x}_1 \vee x_2, \bar{x}_1 \vee \neg x_2, \bar{x}_1 \vee x_3, \bar{x}_1 \vee \neg x_3\}$, dobija se skup klauza $F^{'} = \{[\,], [\,]\}$. Primetimo da svaka valuacija u kojoj je $\bar{x_1}$ „postavljeno na tačno" ne zadovoljava jednu klauzu iz F i dve klauze iz $F^{'}$, zbog čega primena pravila propagacije jedinične klauze u MAX-SAT rešavačima nije moguća na način na koji se koristi u SAT rešavačima. Da bi se rešio ovaj problem, uvode se takozvana *saglasna pravila zaključivanja* koja transformišu MAX-SAT instancu F u njoj ekvivalentu $F^{'}$, jednostavniju za rešavanje.

Pogledajmo primer jednog saglasnog pravila zaključivanja: Ukoliko je data MAX-SAT instanca F (skup klauza) koja sadrži tri klauze oblika $l_1, l_2, \bar{l}_1 \vee \bar{l}_2$, gde su l_1, l_2 literali, zamenjujemo je skupom klauza

$$F^{'} = (F - \{l_1, l_2, \bar{l}_1 \vee \bar{l}_2\}) \cup \{[\,], l_1 \vee l_2\}.$$

Prikazano pravilo detektuje kontradikciju iz skupa $l_1, l_2, \bar{l}_1 \vee \bar{l}_2$, menjajući ove tri klauze praznom klauzom. Pravilo dodaje i klauzu $l_1 \vee l_2$ da bi za proizvoljnu valuciju broj nezadovoljenih klauza u F i $F^{'}$ bio isti.

Skup svih mogućih valuacija za datu formulu F (skup klauza) najčešće se predstavlja u obliku *stabla pretrage*, u kom unutrašnji čvorovi predstavljaju delimične valuacije, a listovi potpune valuacije za F. Većina MAX-SAT rešavača je zasnovana na *algoritmu grananja i ograničavanja (eng. Branch and bound algorithm - B&B [5])* kojim se pretražuje stablo tehnikom grananja u dubinu. U svakom čvoru stabla pretrage, algoritam poredi broj nezadovoljenih klauza u do sada najboljoj pronađenoj potpunoj valuaciji, takozvanu *gornju granicu (eng. upper bound - UB)*, sa brojem nezadovoljenih klauza u tekućoj delimičnoj valuaciji (oznaka *#emptyClauses* jer su u formuli tekućeg čvora pretrage nezadovoljene klauze predstavljene praznim klauzama) na koji se dodaje procena minimalnog broja nepraznih klauza koje će postati nezadovoljene ukoliko se tekuća delimična valuacija upotpuni (oznaka *underestimation*). Suma *#emptyClauses* + *underestimation* predstavlja *donju granicu (eng. lower bound - LB)* minimalnog broja nezadovoljenih klauza za bilo koju potpunu valuaciju dobijenu od tekuće delimične valuacije. Ukoliko je $LB \geq UB$, ne može se naći bolje rešenje iz tekućeg čvora pretrage i tada algoritam odbacuje celo podstablo ispod tekućeg čvora i vraća se na prethodni nivo stabla. Ukoliko je $LB < UB$, algoritam pokušava da pronađe eventualno bolje rešenje proširivanjem tekuće delimične valuacije tako što joj dodeljuje novi literal. Time dolazi do grananja iz tekućeg čvora: leva grana odgovara postavljanju literala $\bar{x}$ na tačno, a desna postavljanju literala x na tačno. Formula pridružena levom (desnom) ogranku se dobija brisanjem svih klauza formule tekućeg čvora koje sadrže literal $\bar{x}$ (x) i brisanjem svih pojava literala x ($\bar{x}$), tj. algoritam primenjuje pravilo propagacije jedinične klauze.

Rešenje MAX-SAT problema je vrednost promenljive UB nakon pretrage celog stabla.

Algoritam 2 prikazuje osnovnu strukturu algoritma grananja i ograničavanja za MAX-SAT. Koristi se sledeća notacija:

- $simplifyFormula(F)$ je procedura za pojednostavljivanje formule F primenom saglasnih pravila zaključivanja
- $\#emptyClauses(F)$ je funkcija koja vraća broj praznih klauza u F
- LB je donja granica minimalnog broja nezadovoljenih klauza u F ukoliko se tekuća delimična valuacija upotpuni, pretpostavlja se da je početna vrednost 0
- $underestimation(F, UB)$ je funkcija koja vrši procenu minimalnog broja nepraznih klauza u F koje će postati nezadovoljene ukoliko se tekuća delimična valuacija upotpuni, algoritam 3 prikazaje implementaciju funkcije korišćenjem pravila jedinične propagacije (ideja predstavljena u [3])
- UB je gornja granica broja nezadovoljenih klauza u optimalnom rešenju. Pretpostavlja se da je početna vrednost broj klauza polazne formule F
- $selectVariable(F)$ je funkcija za izbor sledeće granajuće promenljive iz F (koriste se razne heuristike)
- $F_x(F_{\bar{x}})$ je formula dobijena primenom pravila propagacije jedinične klauze na F korišćenjem literala $x(\bar{x})$

Algoritam 2: Algoritam grananja i ograničavanja za MAX-SAT

Ulaz: formula F u KNF i gornja granica UB
Izlaz: minimalni broj nezadovoljivih klauza u F

```
1  function MaxSat(F, UB)
2  F ← simplifyFormula(F);
3  if F = ∅ ili F prazan skup klauza then
4  |  return #emptyClauses(F);
5  LB ← #emptyClauses(F) + underestimation(F, UB);
6  if LB ≥ UB then
7  |  return UB;
8  x ← selectVariable(F);
9  UB ← min(UB, MaxSat(F_x̄, UB));
10 return min(UB, MaxSat(F_x, UB));
```

Algoritam 3: Implementacija funkcije $underestimation$ korišćenjem pravila jedinične propagacije

Ulaz: formula F u KNF i gornja granica UB
Izlaz: procena donje granice za F $underestimation$

```
1  function underestimation(F, UB)
2  underestimation ← 0;
3  primenjivati pravilo jedinične propagacije nad jediničnim klauzama iz F sve
     dok se ne izvede prazna klauza;
4  if ne može se izvesti prazna klauza then
5  |  return underestimation;
6  F ← F bez klauza koje su se koristile prilikom izvođenja prazne klauze;
7  underestimation := underestimation + 1;
8  if underestimation + #emptyClauses(F) ≥ UB then
9  |  return underestimation;
10 go to 3;
```

Većina MAX-SAT rešavača implementira prikazani algoritam, a razlike se ogledaju

u mehanizmima zaključivanja, kao i u korišćenju različitih heuristika za izbor promenljivih. Kao primer, u nastavku ćemo prikazati pravila koja koristi MaxSatz rešavač.

Pregled pravila u MaxSatz rešavaču

MaxSatz koristi sledeća saglasna pravila zaključivanja u proceduri za pojednostavljivanje formule (F_1 je polazni skup klauza, $F^{'}$ je podskup od F_1, a F_2 rezultujući skup klauza):

1. Ukoliko je $F_1 = \{l_1 \vee l_2 \vee \ldots l_k, \bar{l}_1 \vee l_2 \vee \ldots l_k\} \cup F^{'}$, tada je $F_2 = \{l_2 \vee \ldots l_k\} \cup F^{'}$ ekvivalento sa F_1.
2. Ukoliko je $F_1 = \{l, \bar{l}\} \cup F^{'}$, tada je $F_2 = \{[\,]\} \cup F^{'}$ ekvivalentno sa F_1.
3. Ukoliko je $F_1 = \{l_1, \bar{l}_1 \vee \bar{l}_2, l_2\} \cup F^{'}$, tada je $F_2 = \{[\,], l_1 \vee l_2\} \cup F^{'}$ ekvivalentno sa F_1.
4. Ukoliko je $F_1 = \{l_1, \bar{l}_1 \vee l_2, \bar{l}_2 \vee l_3, \ldots, \bar{l}_k \vee l_{k+1}, \bar{l}_{k+1}\} \cup F^{'}$, tada je $F_2 = \{[\,], l_1 \vee \bar{l}_2, l_2 \vee \bar{l}_3, \ldots, l_k \vee \bar{l}_{k+1}\} \cup F^{'}$ ekvivalentno sa F_1.
5. Ukoliko je $F_1 = \{l_1, \bar{l}_1 \vee l_2, \bar{l}_1 \vee l_3, \bar{l}_2 \vee \bar{l}_3\} \cup F^{'}$, tada je $F_2 = \{[\,], l_1 \vee \bar{l}_2 \vee \bar{l}_3, \bar{l}_1 \vee l_2 \vee l_3\} \cup F^{'}$ ekvivalentno sa F_1.
6. Ukoliko je $F_1 = \{l_1, \bar{l}_1 \vee l_2, \bar{l}_2 \vee l_3, \ldots, \bar{l}_k \vee l_{k+1}, \bar{l}_{k+1} \vee l_{k+2}, \bar{l}_{k+1} \vee l_{k+3}, \bar{l}_{k+2} \vee \bar{l}_{k+3}\} \cup F^{'}$, tada je $F_2 = \{[\,], l_1 \vee \bar{l}_2, l_2 \vee \bar{l}_3, \ldots, l_k \vee \bar{l}_{k+1}, l_{k+1} \vee \bar{l}_{k+2} \vee \bar{l}_{k+3}, \bar{l}_{k+1} \vee l_{k+2} \vee l_{k+3}\} \cup F^{'}$ ekvivalentno sa F_1.

Dokazi da su navedena pravila saglasna, tj. da su formule F_1 i F_2 ekvivalente, mogu se naći u [4].

Dodatno, MaxSatz primenjuje i sledeća pravila:

- Pravilo eliminacije čistih literala (videti 3.3.6).
- Pravilo za jedinične klauze: Označimo sa $neg1(x)$ $(pos1(x))$ broj jediničnih klauza u kojima je x negativno (pozitivno). Ukoliko je $\#emptyClauses(F) + neg1(x) \geq UB$, postaviti x na netačno. Ukoliko je $\#emptyClauses(F) + pos1(x) \geq UB$, postaviti x na tačno.
- Pravilo dominantne jedinične klauze: Ukoliko broj klauza u kojima se pojavljuje literal x $(\bar{x})$ nije veći od $neg1(x)$ $(pos1(x))$, postaviti x na netačno (tačno).
- Heuristika za izbor promenljive: Označimo sa $neg2(x)(pos2(x))$ broj binarnih klauza u kojima je x negativno (pozitivno), a sa $neg3(x)$ $(pos3(x))$ broj klauza sa tri ili više literala u kojima je x negativno (pozitivno). Biramo promenljivu za koju je izraz $(neg1(x)+4*neg2(x)+neg3(x))*(pos1(x)+4*pos2(x)+pos3(x))$ maksimalan.
- Heuristika za izbor vrednosti promenljive: Neka je x izabrana granajuća promenljiva. Ukoliko je $neg1(x) + 4 * neg2(x) + neg3(x) < pos1(x) + 4 * pos2(x) + pos3(x)$, postaviti x na tačno, inače, postaviti x na netačno.

3.5.3 PMAX-SAT rešavači

PMAX-SAT (eng. partial maximum satisfiability problem) predstavlja kombinaciju SAT i MAX-SAT problema. Razmatraju se dva tipa klauza:

- *tvrde (eng. hard)* ili nerelaksirajuće klauze koje se označavaju uglastim zagradama $[x \vee y]$
- *meke (eng. soft)* ili relaksirajuće klauze koje se označavaju običnim zagradama $(x \vee y)$

Pretpostavlja se da je data formula F u KNF u kojoj su neke klauze tvrde, a neke meke. PMAX-SAT je problem pronalaženja valuacije koja zadovoljava sve tvrde klauze i maksimalan broj mekih klauza formule F.

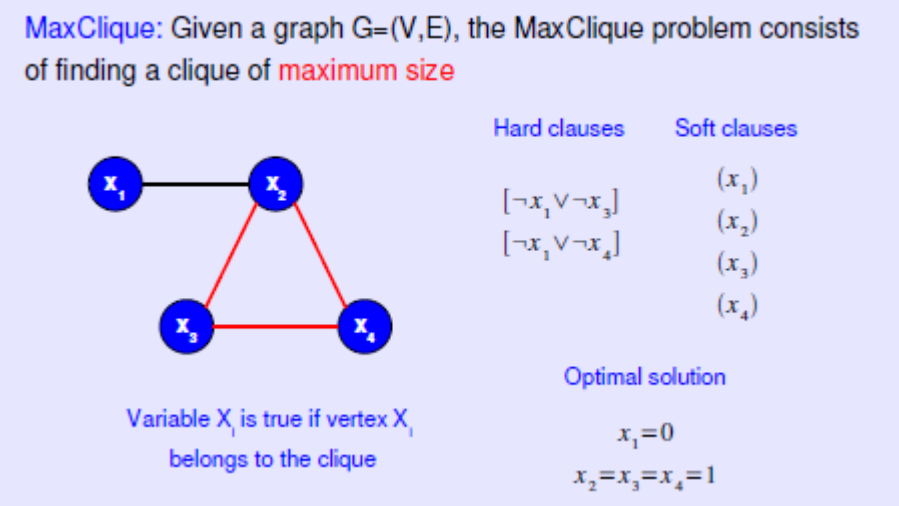
Format podataka za PMAX-SAT Kao ulaz za PMAX-SAT rešavače se koristi DIMACS format (pogledati 3.3.5) sa malim izmenama za tvrde i meke klaze formule u KNF. Nakon proizvoljnog broja linija komentara, slede informacije o formuli. Pravilo je da se svakoj klauzi dodeli težina (ceo broj veći ili jednak 1). Sve meke klauze imaju težinu 1, a suma težina svih mekih klauza mora biti manja od 2^{63}. Za svaku formulu se uvodi donje ograničenje za težinu tvrdih klauza, `top`. `Top` mora biti uvek veći od sume težina zadovoljenih mekih klauza u rešenju. Broj iskaznih promenljivih (`variables`), broj klauza (`clauses`) i ograničenje `top` se definišu linijom `p wcnf variables clauses top`. Svaka od narednih linija predstavlja jednu klauzu, koja kao dodatak u odnosu na klasičan DIMACS format ima na početku dodatu težinu klauze.

Primer ulaznog fajla za formulu $[x_1 \vee \neg x_2 \vee x_4] \wedge [\neg x_1 \vee \neg x_2 \vee x_3] \wedge (\neg x_2 \vee \neg x_4) \wedge (\neg x_3 \vee x_2) \wedge (x_1 \vee x_3)$:

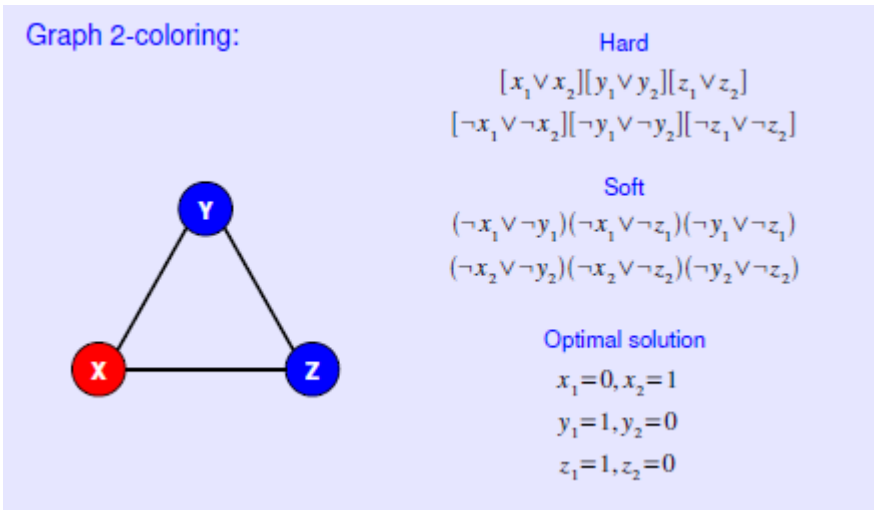
```
c Primer .wcnf fajla
p wcnf 4 5 15
15 1 -2 4 0
15 -1 -2 3 0
1 -2 -4 0
1 -3 2 0
1 1 3 0
```

Reprezentacija sa tvrdim i mekim klauzama i način rešavanja su se pokazali pogodni za mnoge realne probleme. Neki od primera su prikazani na slikama 3.3 i 3.4.

Različite modifikacije za PMAX-SAT rešavače i njihova primena na neke od poznatijih realnih problema su prikazane u [2].



SLIKA 3.3: Rešavanje problema maksimalne klike u grafu.



SLIKA 3.4: Rešavanje problema bojenja grafa sa 2 boje

4 *Sinteza kombinatornih kola*

4.1 Uvod

Logička (istinitosna) funkcija nad n argumenata je funkcija sa domenom $S = \{0,1\}^n$ i kodomenom S, tj. preslikavanje:

$$f : S \times S \times \ldots \times S \to S$$

Nad n argumenata ima 2^{2^n} različitih logičkih funkcija (jer skup $\{0,1\}^n$ ima 2^n elemenata i svaki od njih se može preslikati u 0 ili 1). Svaka iskazna formula koja ima n iskaznih slova generiše neku logičku funkciju nad n argumenata. Iskazne formule sa istim brojem iskaznih slova generišu identične logičke funkcije. Kažemo da je skup iskaznih veznika *potpun* ako je svaka iskazna formula logički ekvivalentna nekoj iskaznoj formuli samo nad tim skupom veznika. Primeri potpunih skupova veznika su $\{\wedge, \vee, \neg\}$, $\{\vee, \neg\}$, $\{\wedge, \neg\}$.

Veznici $\uparrow$ i $\downarrow$ se definišu na sledeći način: $A \uparrow B$ je jednako $\neg(A \wedge B)$, a $A \downarrow B$ je jednako $\neg(A \vee B)$. Veznik $\downarrow$ zovemo $nili$ ili Pirsova funkcija, veznik $\uparrow$ zovemo ni ili Šeferova funkcija. Na osnovu definicija mogu se izvesti istinitosne tablice za ove veznike (tabela 4.1). Pomenuti skupovi veznika kao i $\downarrow$ i $\uparrow$ su potpuni [12].

TABELA 4.1: Istinitosne tablice za $\downarrow$ i $\uparrow$

A	B	$A \downarrow B$	$A \uparrow B$
0	1	1	1
0	1	0	1
1	0	0	1
1	1	0	0

Moderni digitalni sistemi su zasnovani na implementaciji logičkih funkcija. *Logički elementi* predstavljaju elektronske implementacije logičkih veznika (najčešće onih koji predstavljaju potpun skup). Nazivi logičkih elemenata dobijeni na osnovu funkcije koju implementiraju, prikazani su u tabeli 4.2.

TABELA 4.2: Osnovni logički elementi

Funkcija	Naziv logičkog elementa	Naziv na engleskom
Negacija	NE-element	NOT-element
Konjukcija	I-element	AND-element
Disjunkcija	ILI-element	OR-element
Šeferova funkcija	NI-element	NAND-element
Pirsova funkcija	NILI-element	NOR-element

Mreža logičkih elemenata međusobno povezanih u cilju realizovanja logičkih funkcija zove se *kombinatorno kolo*. Informacije dobijene pomoću kombinatornog

kola, tj. rezultat obrade zavisi isključivo od ulaznih vrednosti. Posledica toga je da za svaku moguću kombinaciju ulaznih vrednosti postoji jedinstveni izlaz. Takođe, navedeno svojstvo kombinatornih kola važi i za istinitosne vrednosti iskaznih formula — za datu formulu, one zavise isključivo od vrednosti koje valuacija dodeljuje iskaznim promenljivim u toj formuli.

Sinteza kombinatornih kola je postupak kojim se na osnovu zadatog zakona funkcionisanja dolazi do strukturne sheme kola. Zakon funkcionisanja je dat funkcijama izlaza. To su izrazi koji daju zavisnost svakog izlaznog signala od ulaznih signala. Zadatak sinteze je da na osnovu datog zakona funkcionisanja prvo generiše odgovarajuće Bulove funkcije, zatim izvrši njihovu minimizaciju (uprošćavanje) i na kraju realizuje minimizovane Bulove funkcije pomoću raspoloživih osnovnih kola.

Postoji više različitih formi za predstavljanje kombinatornih kola koje se koriste u procesu sinteze.

4.1.1 Tabelarna forma

Kombinatorno kolo od n binarnih ulaza i m binarnih izlaza se predstavlja u obliku tabele u kojoj jedan red odgovara jednoj od 2^n mogućih ulaznih kombinacija, a kolone predstavljaju odgovarajuće ulazne/izlazne vrednosti. Primetimo da ovaj način reprezentacije kola odgovara istinitosnim tablicama za iskazne formule (videti 3.3.1). Primer kombinatornog kola od 3 ulaza i 1 izlaza je dat u tabeli 4.3.

TABELA 4.3: Primer tabelarne forme za kolo sa tri ulaza A, B i C i izlazom F

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Poseban oblik tabelarne forme su *Karnoove mape* koje predstavljaju jedan od metoda minimizacije kola i biće predstavljene u odeljku 4.2.2.

4.1.2 Algebarska forma

Algebra logike je struktura $\{\{\top, \bot\}, \wedge, \vee, \neg\}$. Najčešće se koristi sledeća računarska notacija:

- konstante: 0 za $\bot$, 1 za $\top$
- operacije: $'$ ili $^-$ za $\neg$ (oznake A', $\bar{A}$), $+$ za $\vee$, $*$ za $\wedge$ (nekada, umesto $A * B$ se koristi i skraćena oznaka AB).

Zakonitosti koje važe u pomenutoj strukturi su prikazane u tabeli 4.4.

U skladu sa prikazanom notacijom algebre logike, konjukcija se često posmatra kao proizvod svojih argumenata, a disjunkcija kao suma svojih argumenata. Zbog toga ćemo uvesti i nove oznake za već definisane normalne forme, KNF i DNF (videti uvodno poglavlje 3.2). Za KNF sada možemo reći da predstavlja proizvod

TABELA 4.4: Zakonitosti

Zakon komutacije	$AB = BA$	$A + B = B + A$
Zakon asocijacije	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Zakon distribucije	$A(B + C) = AB + AC$	$A + (BC) = (A + B)(A + C)$
Neutralni element	$1 * A = A$	$0 + A = A$
Inverzni element	$AA' = 0$	$A + A\prime = 1$
Nula	$A * 0 = 0$	$A + 1 = 1$
Idempotencija	$A * A = A$	$A + A = A$
Apsorpcija	$A(A + B) = A$	$A + (AB) = A$
Dvostruka negacija	$A'' = A$	
De Morganova pravila	$(AB)' = A' + B\prime$	$(A + B)\prime = A'B'$

suma literala, pa u skladu sa tim uvesti oznaku *POS* (od eng. product of sums). Slično, DNF bi predstavljala sumu proizvoda literala, čemu bi odgovarajuća oznaka bila *SOP* (od eng. sum of products).

Prikazaćemo kako se dobija POS i SOP zapis iz tabelarne forme na primeru kola iz tabele 4.3: SOP - za svaku ulaznu kombinaciju koja daje na izlazu vrednost 1 napraviti odgovarajući proizvod, ukoliko je vrednost promenljive u kombinaciji 1, piše se oznaka promenljive, u suprotnom, piše se njena negacija (komplement), SOP je suma pomenutih proizvoda:

$$A'B'C' + A'B'C + AB'C' \tag{4.1}$$

POS - za svaku ulaznu kombinaciju koja daje na izlazu vrednost 0 napraviti odgovarajuću sumu, ukoliko je vrednost promenljive u kombinaciji 0, piše se oznaka promenljive, u suprotnom, piše se njena negacija (komplement), POS je proizvod pomenutih suma:

$$(A + B' + C)(A + B' + C')(A' + B + C')(A' + B' + C)(A' + B' + C') \tag{4.2}$$

Alternativni zapis SOP Za svaki proizvod u SOP zapisu, uvodi se odgovarajući broj n_i čiji binarni zapis kodira polaritet svakog od literala iz toga proizvoda, tj. konjukcije. Tada se SOP može zapisati u obliku $\sum(n_1, n_2, \ldots, n_k)$, gde $n_1, n_2, \ldots, n_k$ predstavljaju opisane brojeve. Alternativna SOP forma za primer 4.1 dobija se na sledeći način :

$$\begin{aligned} A'B'C' &= 000_2 = 0 = n_1 \\ A'B'C &= 001_2 = 1 = n_2 \\ AB'C' &= 100_2 = 4 = n_3 \\ A'B'C' + A'B'C + AB'C' &= \sum(n_1, n_2, n_3) = \sum(0, 1, 4) \end{aligned}$$

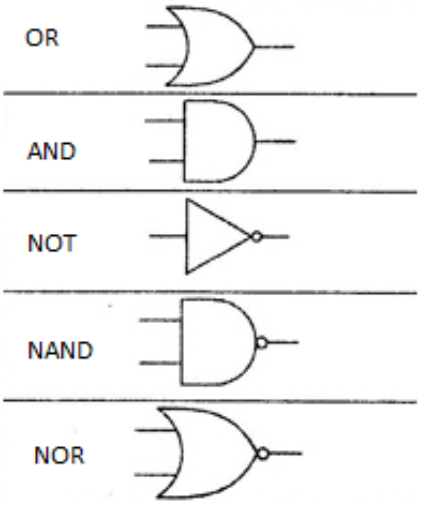
Alternativni zapis POS Za svaku sumu u POS zapisu, uvodi se odgovarajući broj n_i čiji binarni zapis kodira polaritet svakog od literala iz te sume, tj. disjunkcije. POS se tada zapisuje u obliku $\prod(n_1, n_2, \ldots, n_k)$, gde $n_1, n_2, \ldots, n_k$ predstavljaju opisane

brojeve. Za primer 4.2, alternativna POS forma se dobija na sledeći način:

$$A + B' + C = 101_2 = 5 = n_1$$
$$A + B' + C' = 100_2 = 4 = n_2$$
$$A' + B + C' = 010_2 = 2 = n_3$$
$$A' + B' + C = 001_2 = 1 = n_4$$
$$A' + B' + C' = 000_2 = 0 = n_5$$
$$(A + B' + C)(A + B' + C')(A' + B + C')(A' + B' + C)(A' + B' + C') = \prod(5, 4, 2, 1, 0)$$

4.1.3 Grafička forma

Skup najčešće korišćenih grafičkih simbola za osnovne logičke elemente, definisan ANSI/IEEE standardom iz 1984. je prikazan na slici 4.1.



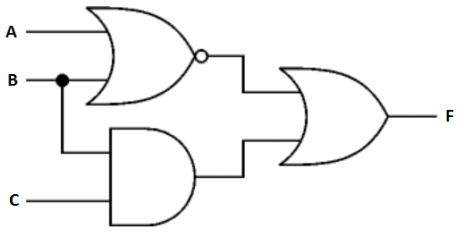
SLIKA 4.1: Grafički simboli za osnovne logičke elemente

Svako kombinatorno kolo može se grafički predstaviti povezivanjem prikazanih simbola za osnovne logičke elemente. Postoji dosta programa za projektovanje logičkih kola i simulaciju njihovog rada, CEDAR Logic, Logisim, Logic Circuit Designer,...

Na primer, grafička reprezentacija kombinatornog kola F koje je zadato u algebarskoj formi:

$$F = (A + B)' + BC$$

prikazana je na slici 4.2.



SLIKA 4.2: Primer grafičke reprezentacije kombinatornog kola

4.1.4 Vrste kombinatornih kola

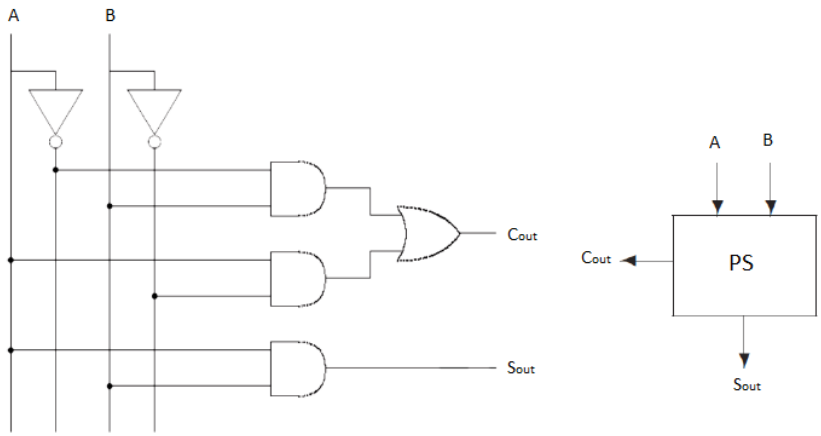
Sabirači

Među najčešće korišćenim kombinatornim kolima su *sabirači*. Sabirači su kombinatorna kola koja se koriste pri implementaciji operacije sabiranja. Razlikujemo polusabirače, pune sabirače i složene (višebitne) sabirače.

Polusabirač (oznaka PS) je kombinatorno kolo koje sabira svoje ulaze, dva jednocifrena binarna broja A i B. Izlazi kola su jedan bit rezultata S_{out} i jedan bit prenosa C_{out}. Tabelarna forma ovog kola je prikazana u tabeli 4.5, a grafička reprezentacija kao i najčešće korišćen simbol za polusabirač mogu se videti na slici 4.3.

TABELA 4.5: Tabelarna forma za polusabirač

A	B	S_{out}	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



SLIKA 4.3: Polusabirač: a) grafička reprezentacija, b) grafički simbol

Potpun sabirač (najčešće se pod sabirač misli na potpun, oznaka S) je kombinatorno kolo koje sabira svoje ulaze, dva jednocifrena binarna broja A, B i jednocifren binarni prenos C. Izlazi kola su jedan bit rezultata S_{out} i jedan bit prenosa C_{out}. Tabelarna forma ovog kola je prikazana u tabeli 4.6, a grafička reprezentacija kao i najčešće korišćen simbol za sabirač mogu se videti na slici 4.4.

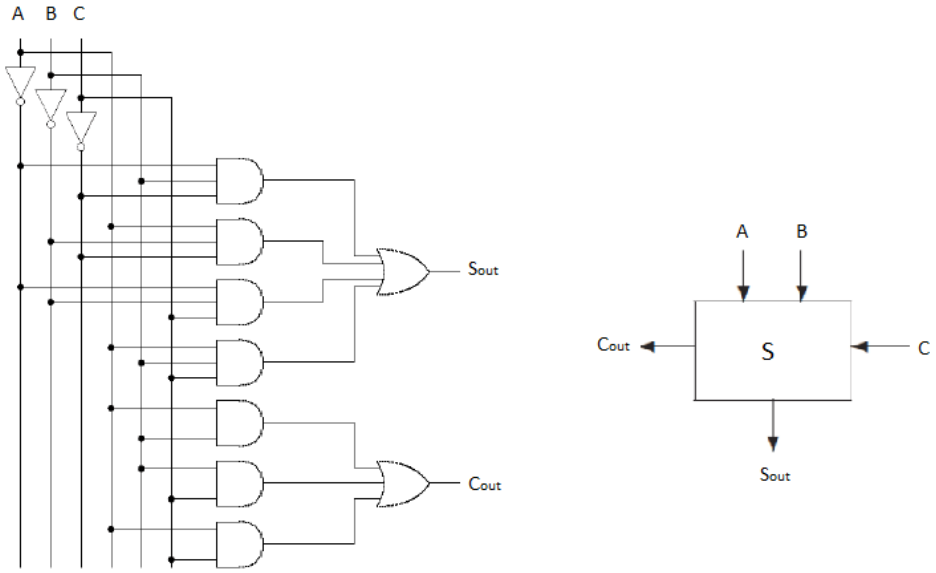
Složeni ili višebitni sabirač je kombinatorno kolo koje sabira dva višecifrena binarna broja A, B i dati prenos C. Implementira se pomoću više binarnih sabirača. Primer 4-bitnog sabirača je dat na slici 4.5.

Multiplekseri

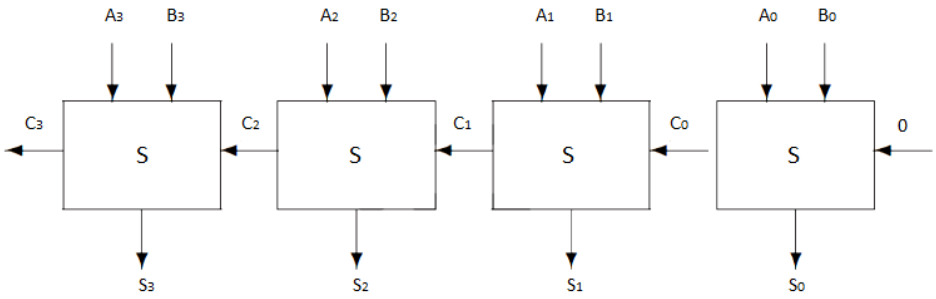
Multiplekseri su kombinatorna kola koja imaju 2^n ulaza, dodatnih n selektorskih ulaza i 1 izlaz. Vrednost izlaza odgovara vrednosti ulaza koji je određen vrednošću selektorskih ulaza. Biranjem ulaznih vrednosti, multiplekseri se mogu koristiti za

TABELA 4.6: Tabelarna forma za polusabirač

C	A	B	S_{out}	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



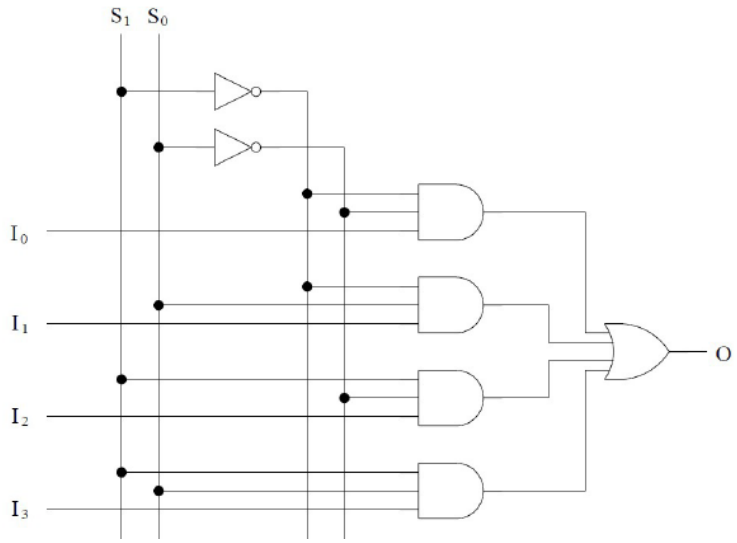
SLIKA 4.4: Sabirač: a) grafička reprezentacija, b) grafički simbol



SLIKA 4.5: 4-bitni sabirač

implementiranje funkcija od selektorskih ulaza, kao što su izračunavanje parnosti za selektorske ulaze, izračunavanje zastupljenijeg bita na selektorskim ulazima, itd.

Grafička forma multipleksera 4-1 prikazana je na slici 4.6, gde su sa I_0, I_1, I_2, I_3 označeni ulazi, sa S_0 i S_1 označeni selektorski ulazi, a sa O izlaz.

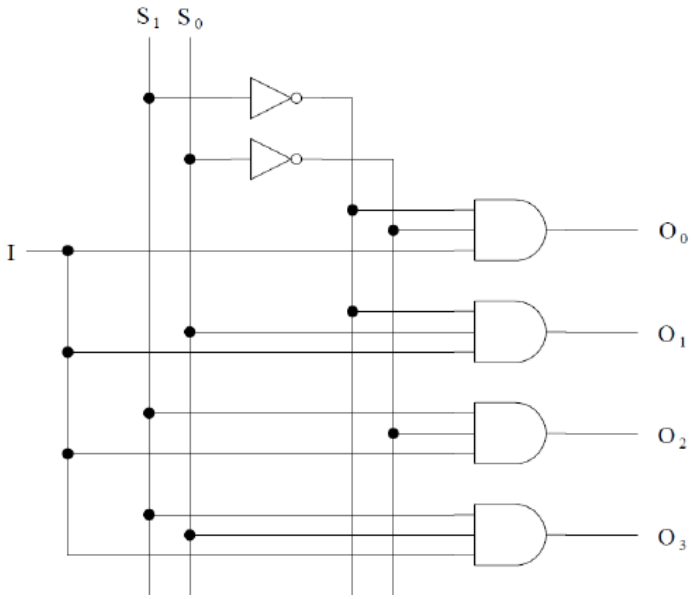


SLIKA 4.6: Multiplekser 4-1

Demultiplekseri

Demultiplekseri su kombinatorna kola koja imaju $n + 1$ ulaz, dodatnih n selektorskih ulaza i 2^n izlaza. Predstavljaju inverzno kolo multipleksora. Vrednost ulaza se preslikava na tačno jedan izlaz, dok svi ostali izlazi imaju vrednost 0.

Grafička forma demultipleksera 1-4 data je na slici 4.7, gde je sa O označen ulaz, sa S_0 i S_1 su označeni selektorski ulazi, a sa O_0, O_1, O_2, O_3 izlazi.

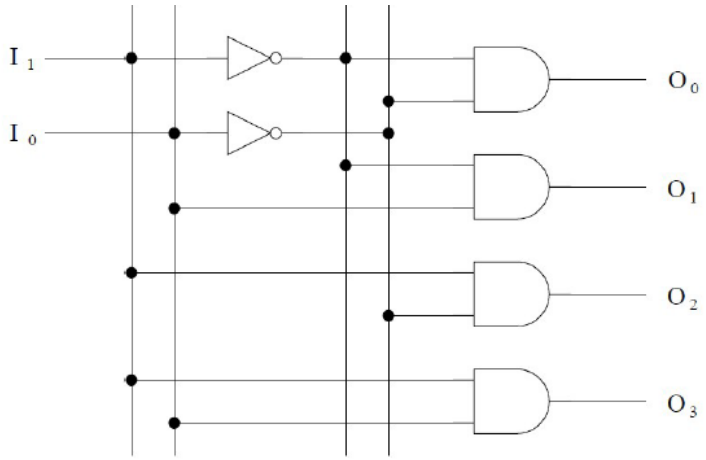


SLIKA 4.7: Demultiplekser 1-4

Dekoderi

Dekoderi su kombinatorna kola koja imaju n ulaza i najviše 2^n izlaza. U zavisnosti od ulaza, uvek je aktivan najviše jedan izlaz. Koriste je najviše za dekodiranje memorijskih adresa.

Grafička forma dekodera 2-4 prikazana je na slici 4.8, gde su sa I_0, I_1 označeni ulazi, a sa O_0, O_1, O_2, O_3 izlazi. Na osnovu 2-bitnog broja bira se odgovarajući izlaz.



SLIKA 4.8: Dekoder 2-4

Enkoderi

Dekoderi su kombinatorna kola koja imaju 2^n ulaza i n izlaza. Predstavljaju inverzna kola dekoderima. U svakom trenutku je aktivan najviše jedan ulaz kojim je određen izlaz. Najčešće se dodaju kontrolni ulaz koji uključuje enkoder i kontrolni izlaz koji je aktivan ukoliko je aktivan kontrolni ulaz i bar jedan ulazni bit.

Tablica enkodera 4-2 sa kontrolnim ulazom In i izlazom Out data je u tabeli 4.8, a grafička forma je prikazana na slici 4.9 gde su sa I_0, I_1, I_2, I_3 označeni ulazi, a sa O_0, O_1 izlazi. Oznaka x u tabeli znači da izlaz kola ne zavisi od ulaza (samo u situaciji kada kontrolni signal nije aktivan).

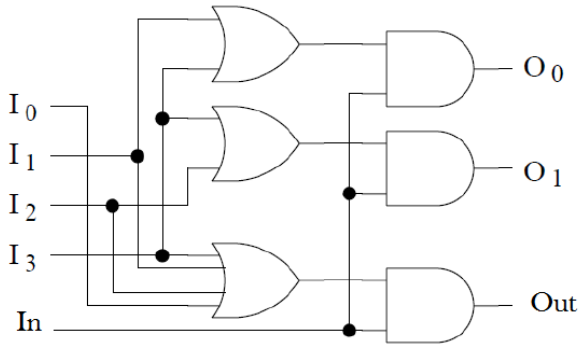
TABELA 4.7: Enkoder 4-2

In	I_3	I_2	I_1	I_0	O_1	O_0	Out
0	x	x	x	x	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	1	1	0	1
1	0	0	1	0	0	1	1
1	0	1	0	0	1	0	1
1	1	0	0	0	1	1	1

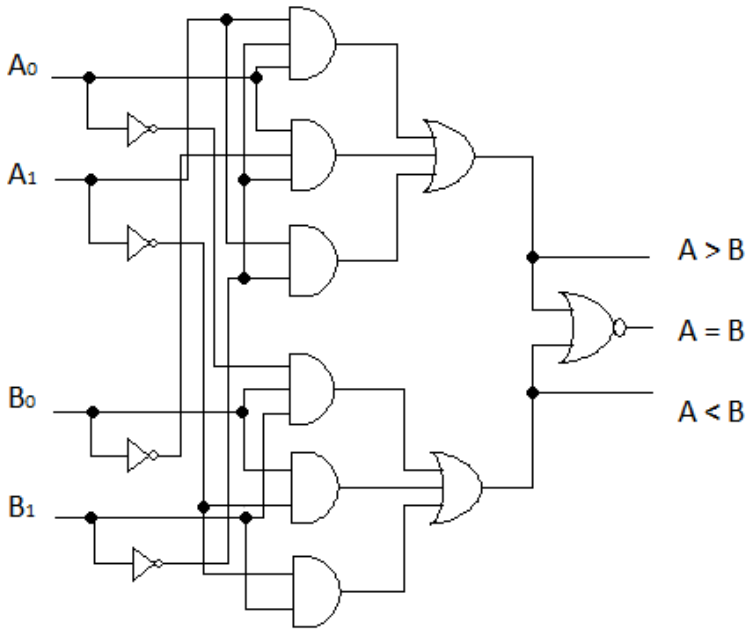
Komparatori

Komparatori su kombinatorna kola koja porede dva niza od po n bitova na odgovarajući način. Imaju $2n$ ulaza i u zavisnosti od vrste poređenja, odgovarajući broj izlaza. Nizovi bitova se mogu porediti samo na jednakost i u tom slučaju je dovoljan jedan izlaz, a postoje varijante koje porede veličine brojeva, u kojima su dovoljna dva, a najčešće se koriste tri izlaza. Ukoliko se porede brojevi čiji binarni zapisi imaju mn bitova, za $m > 2$, ulazna informacija obuhvata i rezultat poređenja prethodne grupe od n bitova.

Grafički forma komparatora dva 2-bitna broja A i B prikazana je na slici 4.10.



SLIKA 4.9: Enkodera 4-2 sa kontrolnim ulazom In i izlazom Out



SLIKA 4.10: Komparator dva 2-bitna broja

4.2 Metode minimizacije logičkih funkcija

Kombinatorna kola mogu biti vrlo složena. Iz tog razloga, pre nego što se na osnovu kreiranog dizajna pristupi fizičkoj implementaciji, dizajn prolazi kroz niz transformacija kojima se vrše optimizacije kola kako bi se uštedelo na njegovoj proizvodnji, doprinelo brzini i slično. Da bi se zapisivanje i implementacija logičkih funkcija učinili jednostavnijim, potrebno ih je minimizovati. Minimizovanje logičke funkcije je pronalaženje njenog najjednostavnijeg zapisa. U nastavku će biti prikazane neke od najpoznatijih metoda minimizacije.

4.2.1 Algebarske transformacije

Osnovni metod za pojednostavljivanje kombinatornog kola je metod zasnovan na algebarskim transformacijama. Algebarski pristup minimizaciji logičkih funkcija

se zasniva na primeni zakonitosti prikazanih u tabeli 4.4, čime se složenije podformule zamenjuju jednostavnijim, logički ekvivalentnim formulama. Prikazaćemo primer minimizacije funkcije $F = A'BC' + A'BC + ABC'$ korišćenjem algebarskih transformacija po koracima:

1. primena idempotencije (dodavanje $A'BC'$) i komutacije
   - $F = A'BC' + A'BC + ABC' + A'BC'$
2. primena zakona distribucije
   - $F = A'B(C' + C) + (A + A')BC'$
3. primena zakona o inverznim elementima
   - $F = A'B + BC'$
4. primena zakona komutacije i distibucije
   - $F = B(A' + C')$

Mana ovog načina pojednostavljivanja funkcija je što pre svega ne postoji opšti algoritam za primenu zakonitosti i primena postaje vrlo teška povećanjem stepena složenosti polaznog izraza. Zbog toga, nema veliku praktičnu primenu.

4.2.2 Karnoove mape

Karnoova mape ili skraćeno *K-mape* predstavljaju tablični metod minimizacije logičkih funkcija. Metod je prvi put predstavljen 1953. u radu [14].

Opis Karnoove mape:

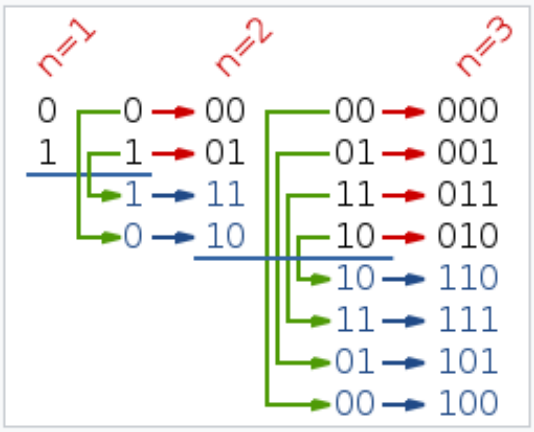
- ako je n broj promenljivih funkcije, mapa se sastoji od 2^n kvadrata
- kolone i vrste mape se označavaju kombinacijama vrednosti promenljivih
- ako je širina (odnosno visina) mape n kvadrata, po širini (odnosno visini) se zadaju vrednosti za $\log_2 n$ promenljivih
- oznake kolona, odnosno vrsta (kombinacije vrednosti promenljivih) su poređane tako da čine *Grejov kod* (sistem binarnih cifara u kojem se dve uzastopne vrednosti razlikuju u samo jednom bitu, detaljnije u [8])

Konstukcija n-bitnog Grejovog koda primenom refleksije: n-bitna lista Grejovog koda može biti generisana rekurzivno iz liste za $n - 1$ bitova tako što se na nju nadovežu elementi u obrnutom redosledu (izvrši refleksija liste), a nakon toga se na elemente originalne liste doda prefiks 0, a na elemente reflektujuće liste prefiks 1. Liste Grejovog koda za $n = 1, 2, 3$ su prikazane na slici 4.11.

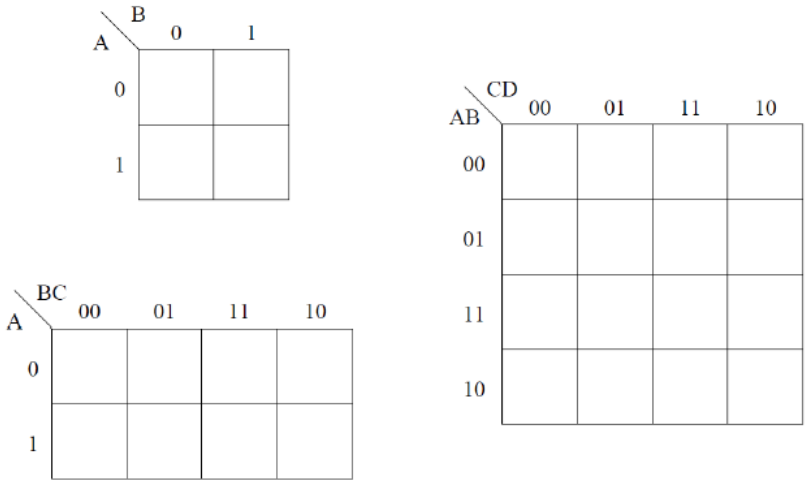
Nepopunjene Karnoove mape za $n = 2, 3, 4$ su prikazane na slici 4.12.

Konstrukcija Karnoove mape u zavisnosti od date reprezentacije:

- ukoliko je logička funkcija zapisana u SOP, može se predstaviti pomoću Karnoove mape tako što se u svako polje mape upiše 1 ukoliko postoji konjukcija u SOP takva da je njena vrednost 1 za vrednosti promenljivih koje odgovaraju tom polju.



SLIKA 4.11: Primeri Grejovog koda



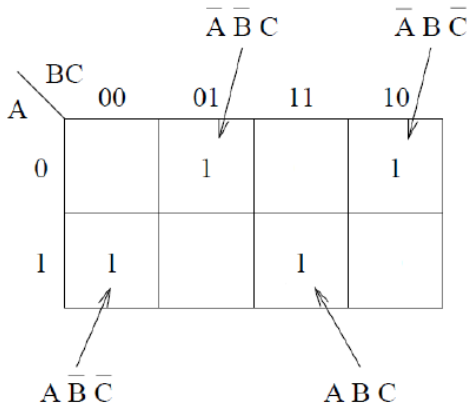
SLIKA 4.12: Primeri Karnoovih mapa

- ukoliko je data tabelarna reprezentacija funkcije, Karnoova mapa se dobija jednostavim upisivanjem jedinica u polja koja odgovaraju vrstama tabele za koje je vrednost funkcije 1. Ukoliko tabela nije potpuna (nedostaju neke vrste), u polja mape koja odgovaraju tim vrstama možemo upisati neki specijalni simbol, npr. n, ?,$*$,... Pri minimizaciji se takva polja interpretiraju onako kako nam odgovara u cilju što bolje minimizacije.

Na primer, Karnoova mapa za funkciju $F = A'BC' + A'B'C + ABC + AB'C'$, prikazana je na slici 4.13.

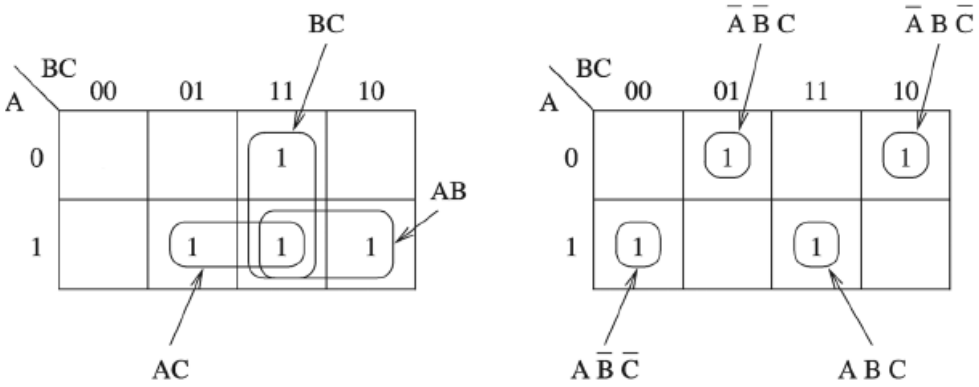
Minimizacija se zasniva na postupku uočavanja grupa od po 2^k jedinica kojima se konjukcija može dodeliti kao grupi, umesto da se to radi pojedinačno. Kod formiranja grupa jedinica, važe sledeća pravila:

- grupe se sastoje samo od jedinica
- broj jedinica u grupi mora biti stepen dvojke
- jedinice moraju biti raspoređe u susednim poljima u obliku pravougaonika
- svaka jedinica mora biti u nekoj grupi
- grupe se mogu preklapati



SLIKA 4.13: Karnoova mapa

- grupe čija su polja u potpunosti sadržana u nekim drugim grupama treba zanemariti
- smatra se da mapa ima oblik torusa, odnosno da se mogu grupisati i jedinice koje postaju susedne kada se spoje naspramne ivice mape

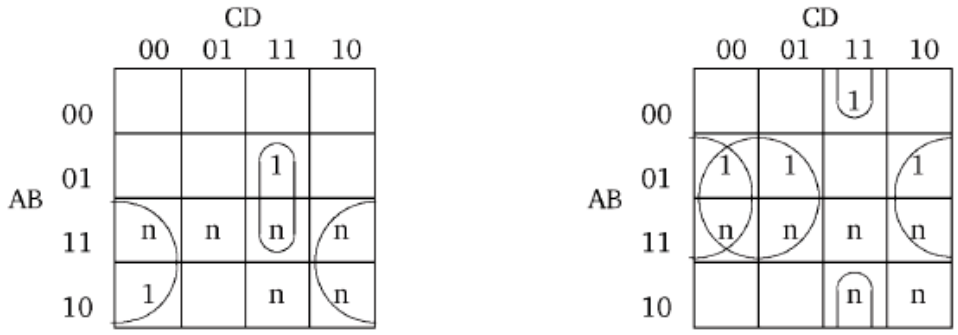


SLIKA 4.14: Primeri grupisanja u K-mapama

Data pravila ne određuju jednoznačno grupisanje jedinica. Osnovni princip koji garantuje miniminalnost je: vršiti grupisanje tako da se sa što manje što većih grupa obuhvate sve jedinice. Primeri grupisanja u Karnoovim mapama su prikazani na slici 4.14. Specijalno, ukoliko mapa sadrži neodređena polja, njih tumačimo na način da grupišemo jedinice u što manje što većih grupa (slika 4.15).

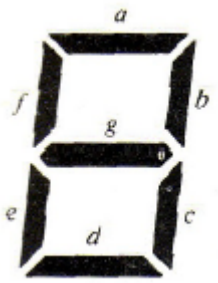
Određivanje analitičkog oblika minimizovane forme, tj. čitanje mape se sastoji u sledećem:

- za svaku formiranu grupu definiše se konjukcija, promenljiva X koja je konstantna na svim poljima grupe učestvuje kao literal X ako je njena vrednost 1 ili kao X' ako je njena vrednost 0
- kada su formirane konjukcije za sve grupe, konačna funkcija se dobija kao disjunkcija ovih pojedinačnih članova



SLIKA 4.15: Primeri grupisanja u nepotpunim K-mapama

Pogledajmo i konstrukciju Karnoove mape za reprezentaciju logičke forme veoma korišćenog elektronskog uređaja, *sedmodelnog displeja*. *Sedmodelni displej* (eng. seven-segment display, SSD) je elektronski uređaj za prikaz decimalnih cifara. Koristi se u digitalnim časovnicima, digitronima i mnogim složenijim elektronskim uređajima koji prikazuju digitalne informacije. Za svaki od sedam segmenata displeja je zaduženo po jedno kombinatorno kolo sa četiri ulaza i jednim izlazom, koji odgovara signalu da li je odgovarajući segment uključen ili isključen u prikazu neke cifre (slika 4.16). Ulazi tih kola predstavljaju bitove jedne BCD cifre (BCD je klasa binarnih kodiranja decimalnih brojeva gde se svaka decimalna cifra predstavlja sa fiksnim brojem bitova, najčešće sa četiri kao u ovom slučaju). Sedmodelni displej ćemo predstaviti kao skup kombinatornih kola čiji je zakon funkcionisanja dat u obliku istinitosne tablice (4.8).

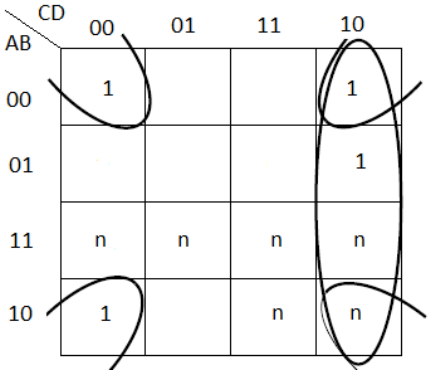


SLIKA 4.16: Sedmodelni displej

TABELA 4.8: Istinitosna tablica za sedmodelni displej

	Ulaz BCD kod				Izlaz Segmenti displeja						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Napravićemo Karnoovu mapu za jedan segment, tj. jedno kombinatorno kolo. Na prime, za segment e. Primetimo da istinitosna tablica nije potpuna, tj. nisu dati ulazi koji ne predstavljaju cifre. Karnoova mapa u ovom slučaju ima nedefinisana stanja, koja ćemo tumačiti kao jedinice ukoliko time možemo napraviti veću grupu (slika 4.17). Čitanjem mape, dobijamo funkciju za segment $e = B'D' + CD'$ u SOP,



SLIKA 4.17: Karnoova mapa za e segment sedmodelnog displeja

koja se može zapisati i u POS kao $e = (B' + C)D'$ za koju bi se u dizajnu koristilo jedno kolo manje.

Karnoove mape se koriste za funkcije do šest promenljivih. Za veće brojeve promenljivih postaju nepregledne i previše složene. Zbog toga je ubrzo nakon ovog metoda, predstavljena tablična metoda Kvin-MekKlaskog, koja je funkcionalno zasnovana na Karnoovim mapama i alternativnom zapisu za SOP, ali se pokazala efikasnijom i jeftinijoj za računarsku implementaciju [19]. Međutim, još uvek je broj promenljivih bio dosta ograničen (oko desetine). Potreba za sintezom funkcija koje imaju ogroman broj ulaza i izlaza je dovela do pravljenja efikasnih optimizatora, koji korišćenjem određenih heuristika mogu da vrše minimizaciju kola sa preko 100 ulaza/izlaza i predstavljaju osnovu mnogih drugih optimizacionih alata.

4.2.3 Heurističke metode za minimizaciju

Heurističke metode ne mogu garantovati optimalno rešenje, ali za relativno kratko vreme daju rešenje vrlo blizu optimalnog, koje je zadovoljavajuće za praktičnu primenu. Do tada poznate Karnoove mape i metoda Kvin-MekKlaskog bile su praktično neupotrebljive za složenija kola (preko desetine ulaza), tako da je korišćenje različitih heuristika dovelo do značajnih pomaka u oblasti sinteze funkcija. Najpoznatiji heuristički optimizatori među kojima su Espresso, Presto, Mini i Pop, mogu da rade sa formulama kombinatornih kola koje imaju preko 10000 klauza i 100 ulaza/izlaza.

Osnovu heurističkih metoda predstavljaju sledeći koraci:

1. spajanje, tj. objedinjavanje proizvoda
2. eliminisanje literala iz proizvoda
3. brisanje redundantnih proizvoda

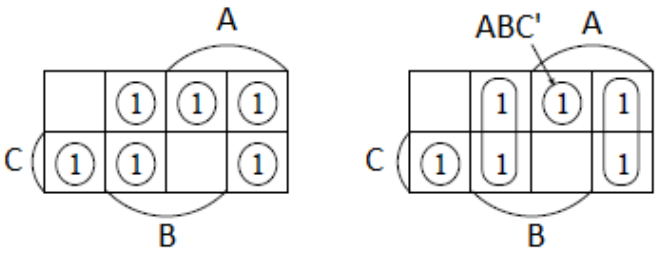
Operacija spajanja proizvoda se zasniva na identitetu $A + A' = 1$. Ukoliko se dva proizvoda razlikuju samo u suprotnim literalima jedne promenljive, ti proizvodi se mogu spojiti u jedan bez pomenutih literala. Na primer, za datu formulu u SOP:

$$F_1 = ABC' + AB'C + AB'C' + A'BC + A'BC' + A'B'C \tag{4.3}$$

2. i 3. proizvod, kao i 4. i 5. proizvod zadovoljavaju uslove za spajanje na osnovu C, čime se dobija pojednostavljena formula:

$$F_2 = ABC' + AB' + A'B + A'B'C. \tag{4.4}$$

Na slici 4.18 levo je prikazana Karnoova mapa formule F_1, a desno rezultat spajanja proizvoda, tj. dve grupe na Karnoovoj mapi čime se dobija formula F_2.

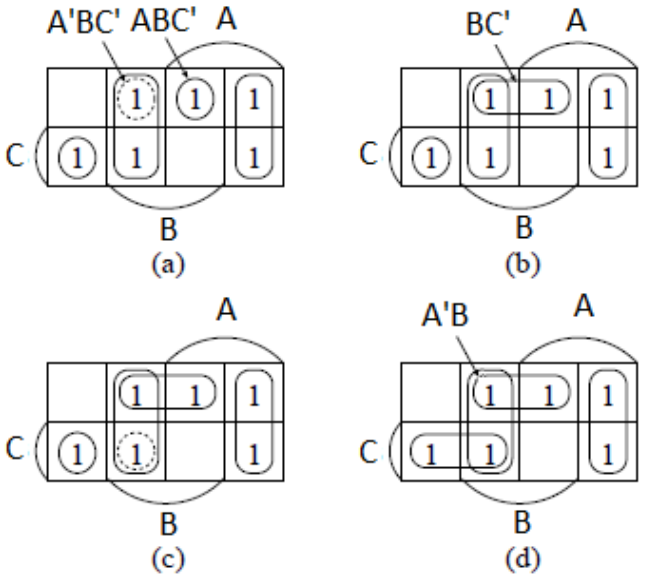


SLIKA 4.18: Operacija spajanja proizvoda

Operacija eliminisanja literala iz proizvoda se zasniva na sledećoj ideji. Pretpostavimo da želimo eliminisati literal A iz proizvoda AC koji je deo formule F. Zapišimo polaznu formulu u obliku $AC + G$, gde je G ostatak formule F koji ne sadrži AC. Ukoliko je proizvod $A'C$ sadržan u polaznoj formuli F, tada na osnovu zakonitosti iz 4.4 važi:

$$F = A'C + F = A'C + (AC + G) = (A' + A)C + G = C + G$$

čime smo eliminisali literal A iz proizvoda AC. Prikazani postupak se ponavlja i za ostale literale proizvoda. Kada više nijedan literal ne može biti eliminisan iz proizvoda, kažemo da je rezultujući proizvod nedeljiv. U kontekstu Karnoove mape, to znači da je grupa koja odgovara tom proizvodu maksimalno proširena. Na slici



SLIKA 4.19: Operacija eliminisanja literala iz proizvoda

4.19 je prikazana primena ovog postupka nad svim proizvodima formule 4.4. Na osnovu Karnoove mape formule F_2, zaključujemo da se iz prvog proizvoda ABC' može eliminisati A jer je proizvod $A'BC'$ sadržan u formuli F_2 (slika 4.19 deo pod

(a)). Time se dobija formula:

$$F_2 = BC' + AB' + A'B + A'B'C$$

Proizvod BC' je postao nedeljiv, jer ni $B'C'$ ni BC nisu sadržani u formuli F_2 da bi se mogao eliminisati literal B, odnosno literal C iz proizvoda (slika 4.19 deo pod (b)). Na sličan način možemo zaključiti da su i proizvodi AB', $A'B$ nedeljivi. Na kraju, iz proizvoda $A'B'C$ možemo eliminisati literal B' (slika 4.19 deo pod (d)) čime se dobija nedeljivi proizvod $A'C$ (slika 4.19 deo pod (d)) i finalna forma SOP:

$$F_2 = BC' + AB' + A'B + A'C. \tag{4.5}$$

Brisanjem redundantnih proizvoda se dobija minimalna (bar lokalno) forma SOP. Iako se primenom prethodnih koraka dolazi do forme SOP koja je izgrađena samo od nedeljivih proizvoda, moguće je da sadrži redundantne (nepotrebne) proizvode. Ovakvi proizvodi se jednostavno mogu uočiti na Karnoovoj mapi, jer je grupa koja njima odgovara u potpunosti prekrivena nekom drugom grupom ili delovima više drugih grupa. Za formulu 4.5, brisanjem redundantnog proizvoda $A'B$ (slika 4.19 deo pod (d)), dobija se finalna forma SOP:

$$F_3 = BC' + AB' + A'C.$$

Prikazan je najčešće korišćen redosled primene osnovnih koraka, ali se oni u opštem slučaju mogu proizvoljno kombinovati. U tu svrhu optimizatori koriste različite heuristike kako bi se došlo do kvalitetnijeg rešenja.

5 *Iskazne neuronske mreže*

U ovoj glavi opisuje se novi, diskretni model učenja, nalik neuronskim mrežama sa regularizacijom, kojim se mogu aproksimirati kombinatorna kola na osnovu uzorka ulaza i izlaza, a čije obučavanje, umesto na neprekidnoj optimizaciji, počiva na pretrazi koju vrše SAT i PMAX-SAT rešavači. Kao što je prikazano u glavi 2, neuronskim mrežama je moguće proizvoljno dobro aproksimirati proizvoljnu neprekidnu funkciju, pomoću dovoljno velikog uzorka njenih vrednosti. Međutim, mnogi sistemi, poput kombinatornih kola, ne predstavljaju neprekidne funkcije i primena neuronskih mreža za njihovo modelovanje nije adekvatna. S druge strane, postoji potreba za pronalaženjem kombinatornih kola koja za odredene ulaze daju odredene izlaze (videli smo primer sinteze sedmodelnog displeja u 4.2.2). Zato je poželjno formulisati diskretne modele koji su u stanju da na osnovu uzorka ulaza i izlaza sintetišu kola koja su saglasna sa njima.

5.1 Model iskazne neuronske mreže

Iskazni neuron predstavlja sledeću formulu:

$$\phi(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{w}') = \bigvee_{k=1}^{n} w_k \wedge (w'_k \oplus x_k) \tag{5.1}$$

u kojoj je sa $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ označen vektor argumenata (ulaza) neurona dužine n, $\boldsymbol{w}$ predstavlja vektor koeficijenata neurona, čiji element w_k određuje da li se ulaz x_k uključuje u sumu (disjunkciju) ili ne. Dodatno, $\boldsymbol{w}'$ je pomoćni vektor koeficijenata, čiji element w'_k određuje polaritet odgovarajućeg ulaza x_k. Elementi vektora $\boldsymbol{x}$, $\boldsymbol{w}$ i $\boldsymbol{w}'$ su iskazna slova. Vrednost argumenata obe vrste neurona će zavisiti od ulaza mreže i načina povezivanja neurona u mreži, dok će vrednost koeficijenata biti određena pretragom koju će vršiti SAT i PMAX-SAT rešavači.

Iskazna neuronska mreža sa M slojeva i N_i iskaznih neurona po sloju gde $i = 1, \ldots, M$ se definiše za skup instanci $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) \mid i = 1, \ldots, T\}$, pri čemu $\boldsymbol{x}_i$ označava vektor dimenzionalnosti N_0, a $\boldsymbol{y}_i$ vektor dimenzionalnosti N_M i predstavlja konjunkciju sledećih formula:

- formule koje definišu arhitekturu mreže:

  $$\phi_{ijk} \triangleq \phi(\boldsymbol{\phi}_{ij-1}, \mathbf{w}_{jk}, \mathbf{w'}_{jk})$$

  za $i = 1, \ldots, T$, $j = 1, \ldots, M$ i $k = 1, \ldots, N_j$. Pritom, simboli ϕ_{0jk} za sve j i k predstavljaju iskazna slova, a simboli ϕ_{ijk} za $i \geq 1$ i sve j i k predstavljaju pomoćne metasimbole koji služe za definisanje iskazne formule, ali se u njoj ne pojavljuju. Drugim rečima, svako pojavljivanje simbola ϕ_{ijk} za $i \geq 1$, označava pojavljivanje odgovarajuće formule. Oznaka $\triangleq$ znači da simbol sa leve

strane predstavlja formulu sa desne. Oznaka ϕ_{ij} predstavlja vektor simbola $(\phi_{ij1}, \ldots, \phi_{ijN_j})$.

- formule nadgledanog obučavanja:

$$\bigwedge_{k=1}^{N_0} \phi_{i0k} \Leftrightarrow x_{ik}$$

$$\psi_i \Leftrightarrow \bigwedge_{k=1}^{N_M} \phi_{iMk} \Leftrightarrow y_{ik}$$

gde ψ_i predstavljaju iskazna slova za $i = 1, \ldots, T$.

- [opciono] *binarnost neurona* - za svaki neuron se dodaje ograničenje kardinalnosti za broj nenula koeficijenata:

$$\bigwedge_{j=1}^{M} \bigwedge_{k=1}^{N_j} atMost(2, w_{jk1}, \ldots, w_{jkN_{j-1}})$$

gde w_{jkl} označava l-ti koeficijent k-tog neurona u j-tom sloju mreže ($l = 1, \ldots, N_{j-1}$). Ograničenje kardinalnosti $atMost(2, w_{jk1}, \ldots, w_{jkN_{j-1}})$ se predstavlja formulom u KNF (detaljnije u 3.4).

- [opciono] *regularizacija* - ograničenje kardinalnosti za broj nenula koeficijenata u celoj mreži:

$$atMost(S, w_{111}, \ldots, w_{MN_MN_{M-1}})$$

gde je S parametar modela koji predstavlja gornju granicu za broj nenula koeficijenata u celoj mreži. Ograničenje kardinalnosti $atMost(S, w_{111}, \ldots, w_{MN_MN_{M-1}})$ se predstavlja formulom u KNF (detaljnije u 3.4). Cilj dodavanja ovog ograničenja je sprečiti preprilagođavanje modela skupu za obučavanje.

Uslovi binarnosti nisu neophodni i u kontekstu sinteze kombinatornih kola verovatno nisu ni poželjni. Zato neće biti korišćeni u eksperimentima, iako su podržani implementacijom. Ipak, moguće su i druge primene iskaznih neuronskih mreža. Na primer, u oblasti istraživanja podataka (eng. data mining) nad binarnim podacima. U istraživanju podataka poželjno je analizirati dobijeni model, a nekada je na taj način moguće doći i do novog znanja. Iskazne neuronskre mreže, kao i obične neuronske mreže, se ne mogu lako interpretirati, ali je moguće da korišćenje binarnih veznika u tome može pomoći.

Za neuronske mreže je poznato da mogu proizvoljno dobro da aproksimiraju bilo koju neprekidnu funkciju. Prirodno je postaviti pitanje da li iskazne neuronske mreže mogu da predstave bilo koju logičku funkciju. Iako to nije dokazano, osnov za takvo uverenje je sledeći. Iskazni neuron može da predstavi iskazni veznik NAND, koji predstavlja potpun sistem veznika. Iako se svaka logička funkcija može predstaviti iskaznom formulom pomoću ovog veznika, nije sigurno da je to moguće pri uslovima koje nameće arhitektura iskazne neuronske mreže, koja uslovljava da izlazi svih nižih slojeva budu prosleđeni neuronima na višim slojevima (koji ih, ako predstavljaju NAND veznik moraju invertovati ili zanemariti, a ne mogu ih samo propustiti ka izlazu). Ipak, iskazni neuron može da predstavi i projekciju - funkciju koja na izlaz propušta jedan svoj ulaz, pa deluje da je na taj način za dovoljno veliku mrežu komponovanjem NAND veznika i projekcija moguće predstaviti bilo koju iskaznu formulu i bilo koju logičku funkciju.

5.2 Obučavanje mreže

Obučavanje mreže se zasniva na pronalaženju zadovoljavajuće valuacije formule koja predstavlja iskaznu neuronsku mrežu korišćenjem SAT i PMAX-SAT rešavača. Zbog toga je neophodno izvršiti transformaciju formule u KNF. S obzirom da nije neophodno da formula u KNF bude logički ekvivalentna polaznoj, koristiće se Cajtinova transformacija (videti 3.3.4) nad formulama za nadgledano obučavanje mreže. Formule koje izrazavaju binarnost neurona i formule za regularizaciju su već u KNF i nad njima nije potrebna dodatna transformacija.

Postavlja se pitanje da li je uvek moguće obučiti mrežu tako da bude saglasna sa svim instancama datog skupa za obučavanje i u slučaju da nije, koji je broj grešaka prihvatljiv. U zavisnosti od vrste rešavača koja se koristi za obučavanje mreže, polaznoj formuli mreže se dodaje i ograničenje za broj grešaka koje mreža sme da pravi nad skupom za obučavanje.

U slučaju obučavanja mreže pomoću SAT rešavača, minimalan broj instanci skupa za obučavanje sa kojima mreža mora da bude saglasna, zadaje se sledećim uslovom kardinalnosti:

$$\bigwedge_{ij} atLeast(L, \psi_1, \ldots, \psi_l)$$

gde je $l = |D|$, tj. ukupan broj instanci u skupu za obučavanje, L parametar modela koji zajedno sa parametrom S treba birati na način da broj grešaka bude najmanji moguć, ali da model ne postane preprilagođen skupu za obučavanje. Ograničenje kardinalnosti $atLeast(L, \psi_1, \ldots, \psi_T)$ se transformiše u formulu u KNF koristeći ekvivalenciju 3.1 i prikazane transformacije za uslov $atMost$ (detaljnije u 3.4).

Alternativno, može se postaviti i odgovarajući PMAX-SAT problem. U polaznoj formuli sve klauze se označe kao tvrde i doda se skup mekih klauza $\{\psi_1, \ldots, \psi_T\}$. PMAX-SAT rešavač će tragati za valuacijom koja zadovoljava sve klauze polazne formule i maksimalan broj dodatih mekih klauza, odnosno valuacijom za koju mreža najmanje greši na instancama skupa za obučavanje. Dobro svojstvo PMAX-SAT pristupa je to da ukoliko istekne predviđeno vreme za rešavanje, taj pristup daje model saglasan bar se nekim instancama, dok SAT rešavač ne daje nikakvo rešenje.

6 *Implementacija i eksperimentalni rezultati*

U ovoj glavi će biti predstavljeni relevantni detalji implementacije, detalji sprovedenih eksperimenata i rezultati koji su njima dobijeni.

6.1 Implementacija

U ovom poglavlju će biti prikazana implementacija modela iskazne neuronske mreže koji je opisan u glavi 5. Celokupna implementacija je zasnovana na pojmovima iskazne logike i zahtevanom formatu instanci za SAT i PMAX-SAT rešavače koji su opisani u glavi 3. Predloženi model iskazne neuronske mreže je implementiran u okviru konzolne aplikacije u programskom jeziku C++.

Osnova implementacije je hijerarhija klasa kojom su predstavljene formule iskazne logike neophodne za definisanje iskazne neuronske mreze i njenih osnovnih jedinica - iskaznih neurona. Iskazni neuroni i iskazna neuronska mreža su predstavljeni klasama cije su clanice podaci i metodi definisani u skladu sa opisom modela iz glave 5.

Prilikom implementacije objekata kojima su predstavljene formule iskazne logike korišćena je biblioteka `<memory>` koja implementira mehanizme za upravljanje dinamičkom memorijom među kojima su i pametni pokazivači. Pametni pokazivači omogućavaju automatsko i bezbedno upravljanje životnim vekom objekata. Konkretno, korišćen je pametni pokazivač `std::shared_ptr` koji omogućava deljenje vlasništva objekta na koji pokazuje, tako da više objekata može bezbedno imati pokazivač ka istom objektu.

Osnovna klasa je apstraktna bazna klasa `BaseFormula`, čije su potklase klase kojima su zasebno predstavljeni veznici iskazne logike. Radi efikasnijeg upravljanja objektima ove klase, uveden je novi tip podataka `Formula`, pametni pokazivač za objekte potklasa klase `BaseFormula`. Takođe, neprotivrečna valuacija je predstavljena novim tipom podataka `Valuation`, odnosno mapom `std::map <int, bool>` koja sadrži parove oblika (identifikator iskaznog slova tipa `int`, istinitosna vrednost iskaznog slova u datoj valuaciji).

Klasa `BaseFormula` ima sledeću strukturu:

```
1 class BaseFormula : public enable_shared_from_this<BaseFormula> {

3 public:

5   enum Type { T_TRUE, T_FALSE, T_ATOM, T_NOT, T_AND, T_OR, T_IMP, T_IFF
      };

7   virtual void printFormula() const = 0;
    virtual Type getType() const = 0;
9   virtual bool equalTo(const Formula & f) const = 0;
    virtual bool eval(const Valuation & v, const vector <int> & x) const
      = 0;
```

```
11    virtual Formula substitute(const Formula & a, const Formula & b) = 0;
      virtual Formula simplify() = 0;
13    virtual Formula nnf() = 0;
      virtual Formula cnf(vector <Formula> & def) = 0;
15    virtual ~BaseFormula() {}

17  };
```

Ova klasa sadrži nabrojivi tip podataka `enum Type` koji predstavlja tip formule, tj. vodećeg veznika formule i skup apstraktnih metoda među kojima su najbitniji sledeći:

- `virtual bool eval(const Valuation & v, const vector <int> & x) const` - izračunava interpretaciju formule u datoj valuaciji
- `virtual Formula simplify()` - primenjuje elementarne logičke ekvivalencije za pojednostavljivanje formule (3.3.2)
- `virtual Formula nnf()` - transformiše formulu u NNF pravilima prikazanim u 4.1.1, ali bez uklanjanja ekvivalencije da bi se izbeglo eksponencijalno uvećanje formule
- `virtual Formula cnf(vector <Formula> & def)` - transformiše formulu u KNF Cajtinovom transformacijom (3.3.4), pri čemu dobijene klauze za definicione ekvivalencije izdvaja u vektor `def`, dok je povratna vrednost novo iskazno slovo kojim se predstavlja polazna formula

Potklase klase `BaseFormula` su:

- `AtomicFormula` - apstraktna klasa za predstavljanje logičkih konstanti i iskaznih slova, odgovarajuće potklase su `True`, `False` i `Atom`
- `UnaryConjective` - apstraktna klasa za predstavljanje unarnog iskaznog veznika, sa potklasom `Not` kojim se predstavlja formula čiji je vodeći veznik negacija
- `BinaryConjective` - apstraktna klasa za predstavljanje formula čiji je vodeći veznik binaran, sadrži dve formule, operande koji su povezani binarnim veznikom. Potklase ove klase su klase `Imp` i `Iff` za formule čiji su vodeći veznici implikacija, odnosno ekvivalencija
- s obzirom da formule mreže zahtevaju i proizvoljne n-arne veznike konjukcije i disjunkcije, definisane su i odgovarajuće klase `And` i `Or` koje sadrže n-arni vektor formula operanada

Iskazni neuron se predstavlja objektima klase `PropositionalNeuron`, za koju se takođe uvodi novi tip podataka `Neuron`, koji predstavlja pametni pokazivač za objekte ove klase.

Klasa `PropositionalNeuron` ima sledeću strukturu:

```
1  class PropositionalNeuron : public enable_shared_from_this<
       PropositionalNeuron> {

3  private:
     int _m;
5    int _n;
     int _size;
7    vector<Formula> _x;
```

```
  vector<Formula> _w;
  vector<Formula> _v;
  Formula _neuron;

public:
  PropositionalNeuron(const int & n);
  PropositionalNeuron(const int & t, const int & m, const int & n,
    const vector<Formula> & x);
  const int & getM() const;
  const int & getN() const;
  const int & getSize() const;
  const vector<Formula> & getW() const;
  const Formula & getFormula() const;
  bool eval(const Valuation & v, const vector <int> & x) const;
  void printNeuron() const;
};
```

Članice podaci ove klase su:

- `int _m` - sloj mreže u kome se nalazi neuron
- `int _n` - redni broj neurona u `_m`-tom sloju mreže
- `int _size` - veličina neurona, odnosno, broj ulaza neurona
- `vector<Formula> _x` - vektor ulaznih podataka, ulazi mreže ukoliko je u pitanju prvi sloj, a inače formule neurona prethodnog sloja
- `vector<Formula> _w` - vektor $\boldsymbol{w}$ koeficijenata neurona u formuli 5.1 - svaki koeficijent je formula, tačnije iskazno slovo predstavljeno objektom klase `Atom`
- `vector<Formula> _v` -vektor $\boldsymbol{w}'$ koeficijenata neurona u formuli 5.1
- `Formula _neuron` - formula koja predstavlja neuron

Radi lakse implementacije mreže, ulazi mreže ϕ_{i0k} se takođe predstavljaju neuronima koji predstavljaju jedno iskazno slovo. Za njihovo instaciranje se koristi konstruktor sa jednim argumentom koji predstavlja redni broj ulaza mreže, a za instaciranje neurona mreže se koristi konstruktor sa četiri argumenta, koji redom označavaju tip neurona, sloj mreže, redni broj neurona u zadatom sloju i vektor ulaza neurona. Funkcija `eval` određuje interpretaciju formule neurona u datoj valuaciji koeficijenata mreže $\boldsymbol{v}$ kojoj pripada neuron i konkretnih ulaza mreže $\boldsymbol{x}$ za testiranje.

Iskazna neuronska mreža je predstavljena klasom `NeuralNetwork` koja ima sledeću strukturu:

```
class NeuralNetwork : public enable_shared_from_this<NeuralNetwork> {

private:
  int _M;
  vector<int> _N;
  vector< vector<Neuron> > _conections;
  char _use;
  vector<Formula> _learning_constraintsOp1;
  vector<Formula> _learning_constraintsOp2;
  vector<Formula> _learning_constraints;
  Formula _binary_connectives;
  bool _id_b;
  int _S;
  Formula _regularization;
  bool _id_r;
```

```
16    char _test;
      int _L;
18    Formula _error_constraint;
      void makeConstraints();
20    bool makeConnectivesSEQ();
      bool makeConnectivesPAR();
22    bool makeRegularizationSEQ();
      bool makeRegularizationPAR();
24    void makeErrorConstraintSEQ();
      void makeErrorConstraintPAR();
26    void getValuation();
      void testNetwork();
28  public:
      NeuralNetwork(const int & type, const int & M, const vector<int> & N,
         const char & use);
30    const int & getM() const;
      const vector<int> & getN() const;
32    const int & getS() const;
      const bool & getIDR() const;
34    const bool & getIDB() const;
      void printNetwork() const;
36    Formula makeCNF() const;
      void printDIMACS() const;
38
  };
```

Članice podaci ove klase su:

- `int _M` - broj slojeva mreže
- `vector<int> _N` - vektor dimenzija slojeva mreže dužine `_M`
- `vector< vector<Neuron> > _conections` - arhitektura mreže je predstavljena ovim vektorom, element `_conections[i]` predstavlja vektor neurona i-tog sloja mreže
- `char _use` - način korišćenja mreže, u slučaju da se mreža obučava zadaje se karakter T, a u slučaju da se testira obučena mreža zadaje se karakter S.
- `vector<Formula> _learning_constraintsOp1`, `vector<Formula> _learning_constraintsOp2`, `vector<Formula> _learning_constraints` - vektori formula koje se kreiraju na osnovu zadatog skupa za obučavanje za svaku njegovu instancu
- `Formula _binary_connectives` - u slučaju da se koristi opcija za binarnost neurona koja se zadaje preko podatka `_id_b`, kreira se formula koja predstavlja konjukciju svih ograničenja za binarnost
- `Formula _regularization` - u slučaju da se koristi opcija za regularizaciju mreže koja se zadaje preko podatka `_id_r`, kreira se odgovarajuća formula na osnovu zadate gornje granice za broj nenula koeficijenata `_S`
- `char _test` - ovim podatkom se zadaje način obučavanja mreže (M za korišćenje SAT rešavača, a P za korišćenje PMAX-SAT rešavača)
- `Formula _error_constraint` - ukoliko se koristi SAT rešavač za obučavanje mreže dodaje se formula kojom se zadaje minimalan broj instanci skupa za obučavanje sa kojima mreža mora biti saglasna, na osnovu zadatog parametra `_L`

Privatne metode ove klase su zadužene za pravljenje odgovarajućih formula mreže:

- `void makeConstraints()` - kreira formule za obučavanje mreže, tj. vektor `_learning_constraints` na osnovu zadatog skupa za obučavanje
- `bool makeConnectivesSEQ(), bool makeConnectivesPAR()` - kreiraju konjukciju formula koje izražavaju binarnost neurona, verzija SEQ implementira formulu korišćenjem sekvencijalnog brojača opisanu u 3.4.1, dok verzija PAR implementira formulu korišćenjem paralelnog brojača i komparatora opisanu u 3.4.2
- `bool makeRegularizationSEQ(), bool makeRegularizationPAR()` - kreiraju formulu za regularizaciju mreže, verzija SEQ implementira formulu korišćenjem sekvencijalnog brojača opisanu u 3.4.1, dok verzija PAR implementira formulu korišćenjem paralelnog brojača i komparatora opisanu u 3.4.2
- `void makeErrorConstraintSEQ(), void makeErrorConstraintPAR()` - slično kao u prethodnim primerima, dve verzije za kreiranje formule za ograničenje broja grešaka koje su dozvoljene mreži
- `void getValuation()` - učitava obučenu mrežu, tj. dobijenu zadovoljavajuću valuaciju koeficijenata mreže
- `void testNetwork()` - testira obučenu mrežu tako što učitava ulaze za koje ispisuje rezultate mreže

Za potrebe pravljenja formula za ograničenje kardinalnosti pomoću paralelnog brojača i komparatora, implementirane su pomoćne funkcije:

- `vector<Formula> halfAdder(const Formula & a, const Formula & b, vector<Formula> & clauses)` - određuje skup klauza, tj. kreira vektor formula `clauses` koji predstavlja polusabirač
- `vector<Formula> fullAdder(const Formula & a, const Formula & b, const Formula & c, vector<Formula> & clauses)` - određuje skup klauza, tj. kreira vektor formula `clauses` koji predstavlja potpuni sabirač
- `vector<Formula> parallelCounter(const vector<Formula> & x, int begin, int end, vector<Formula> & clauses)` - određuje skup klauza koji predstavlja paralelni brojač
- `void comparator(int k, const vector<Formula> & s, vector<Formula> & clauses)` - određuje skup klauza koji predstavlja komparator

Iskazna neuronska mreža se instancira pozivanjem konstruktora sa četiri argumenta koji redom označavaju tip neurona koji se koristi u mreži, dimenzije mreže, tj. broj slojeva i broj neurona u slojevima i koja je namena mreže, tj. da li želimo da obučimo mrežu ili testiramo gotovu mrežu. Dodatno, moguće je ispisati formulu mreže na standardni izlaz pomoću metoda `void printNetwork() const`.

Arhitektura mreže se zadaje fajlom u kom se prvo navodi broj slojeva mreže uključujući i sloj ulaza mreže, a zatim niz brojeva koji predstavljaju dimenziju sloja ulaza mreže (mora se poklapati sa dimenzijom ulaznih instanci iz skupa za obučavanje), dimenzije skrivenih slojeva mreže i dimenziju izlaznog sloja mreže (mora

se poklapati sa dimenzijom izlaznih instanci iz skupa za obučavanje). Na primer, arhitektura mreže koja ima dva skrivena sloja sa po 10 neurona i obučava se za kolo sa 3 ulaza i 5 izlaza predstavljena u opisanom formatu je sledećeg oblika:

```
4
3 10 10 5
```

Skup za obučavanje mreže zadaje se fajlom u kome su instance zadate u zasebnim linijama i predstavljaju odgovarajuće binarne kombinacije ulaza i izlaza za određeno kombinatorno kolo. Instance su generisane u programskom alatu Logisim izgradnjom istinitosnih tablica za određeno kolo. Na primer, ukoliko bi obučavali mrežu za polusabirač, skup za obučavanje bi bio zadat fajlom sledećeg formata:

```
0 0 0 0
0 1 1 0
1 0 1 0
1 1 0 1
```

Za potrebe obučavanja mreže je implementirana metoda `Formula makeCNF() const` za transformaciju formule mreže u KNF, kao i metoda `void printDIMACS() const` koja predstavlja formulu mreže u obliku ulazne instance za odgovarajući rešavač , tj. ispisuje formulu u DIMACS formatu za SAT (3.3.5) ili PMAX-SAT (3.5.3), u zadati izlazni fajl. Kreiran fajl sa ekstenzijom $.cnf$, odnosno $.wcnf$ se prosleđuje kao ulaz za SAT, odnosno PMAX-SAT rešavač, čime se vrši pretraga za zadovoljavajućom valuacijom koeficijenata mreže. Izlaz SAT, tj. PMAX-SAT rešavača, u slučaju da je pronađena zadovoljavajuća valuacija se koristi kao ulazni fajl za testiranje obučene mreže.

6.2 Eksperimentalni rezultati

U ovom poglavlju će biti prikazan niz eksperimenata nad različitim arhitekturama iskazne neuronske mreže, čiji je model opisan u glavi 5. Model je testiran nad kombinatornim kolima prikazanim u 4.1.4. Za svako kombinatorno kolo su generisane odgovarajuće SAT i PMAX-SAT instance u programu čija je implementacija prikazana u 6.1. Za rešavanje su korišćeni MiniSat kao predstavnik SAT rešavača i QMaxSat kao predstavnik PMAX-SAT rešavača. Eksperimenti su dizajnirani po jedinstvenom šablonu za sva kombinatorna kola i zasnivaju se na postepenom povećavanju dimenzije mreže u širinu (povećavanje broja neurona po sloju) i u dubinu (povećavanje broja slojeva) sve dok rezultat nije zadovoljavajući, tj. dok nije pronađena arhitektura mreže koja može da se obuči za dato kombinatorno kolo. Svi eksperimenti su izvršeni na Intel(R) Core(TM) i3 procesoru pod Windows 7 operativnim sistemom. Vremensko ograničenje izvršavanja SAT i PMAX-SAT rešavača postavljeno je na 30 minuta.

6.2.1 Sinteza kombinatornih kola na osnovu potpunog uzorka ulaza i izlaza

U ovom delu će biti opisani eksperimentalni rezultati u slučaju kada su u skupu za obučavanje date sve kombinacije ulaza sa odgovarajućim izlazima kola. Prvo su prikazani rezultati eksperimenata sa mrežama sa jednim skrivenim slojem neurona. Ti eksperimenti su izvršeni na svim kolima. Potom su prikazani rezultati eksperimenata sa mrežama sa dva ili tri skrivena sloja. Ti eksperimenti su izvršeni na nekim od složenijih kola.

Rezultati testiranja su navedeni u tabelama 6.1-6.16 na sledeći način. Prve vrste tabele sadrže sledeće podatke o kombinatornom kolu za koje se obučava mreža: naziv, veličinu kola (broj ulaza i izlaza) i veličinu odgovarajućeg skupa za obučavanje. Prva kolona sadrži podatke o arhitekturi mreže koja se obučava. Podaci su zadati u formatu $m \times n$, gde n predstavlja broj skrivenih slojeva u mreži, a m broj iskaznih neurona po skrivenom sloju. Broj neurona u izlaznom sloju je određen brojem izlaza kola. Polazna arhitektura mreže je 1×2. Korišćeno je najviše 24 neurona po skrivenom sloju i najviše 3 skrivena sloja. Broj ulaza mreže je jednak broju ulaza kombinatornog kola, a broj izlaza mreže odgovara broju izlaza kombinatornog kola.

Naredne dve kolone se odnose na dimenziju problema koji se rešava, odnosno dimenziju formule mreže u KNF, broj promenljivih (iskaznih slova) i broj klauza. Broj promenljivih u pojedinim slučajevima ide i preko 100 hiljada, a broj klauza i preko 200 hiljada.

Preostale kolone se odnose na dobijene rezultate i vreme izvršavanja SAT i PMAX-SAT rešavača (izraženo u sekundama).

Rezultat pokretanja MiniSat rešavača je predstavljen sledećim oznakama:

- sat, ukoliko je pronađena zadovoljavajuća valuacije formule koja predstavlja mrežu, tj. moguće je obučiti mrežu datim skupom instanci
- $unsat$, ukoliko je data formula mreže nezadovoljiva, tj. mrežu nije moguće obučiti tako da bude saglasna sa svim instancama iz zadatog skupa za obučavanje
- $-$, ukoliko rešavač nije uspeo da ustanovi zadovoljivost/nezadovoljivost formule za zadato ograničenje od 30 minuta, tj. 1800 sekundi

Rezultat pokretanja QMaxSat rešavača je predstavljen sledećim oznakama:

- opt, ukoliko je pronađenja valuacija koja zadovoljava sve tvrde klauze i maksimalan broj mekih klauza formule mreže, što znači da je pronađeno optimalno rešenje
- $unsat$, ukoliko je skup tvrdih klauza formule mreže nezadovoljiv
- $-$, ukoliko rešavač nije uspeo da pronađe optimalno rešenje niti da dokaže nezadovoljivost skupa tvrdih klauza formule za zadato ograničenje od 30 minuta, tj. 1800 sekundi

Dodatno, za QMaxSat je u poslednjoj koloni prikazan broj zadovoljenih mekih klauza (što predstavlja broj instanci iz skupa za obučavanje sa kojima je mreža saglasna) u pronađenom optimalnom rešenju ili do isteka vremenskog ograničenja rada rešavača. Ovaj podatak je veoma koristan jer možemo pratiti napredak u obučavanju mreže, odnosno kako promena arhitekture dovodi do promene kvaliteta rezultata.

U tabeli 6.1 su prikazani rezultati dobijeni za polusabirač. Za ovo kolo je bila dovoljna mreža sa jednim skrivenim slojem sa tri neurona.

U tabeli 6.2 su prikazani rezultati dobijeni za potpun 1-bitni sabirač. Od početne arhitekture 1×2, dodavanjem neurona u skrivenom sloju, za koje je QMaxSat pokazao napredak, dolazi se do arhitekture 1×7 za koju je mreža saglasna sa svim instancama.

Povećanjem dimenzije kola se povećava veličina skupa za obučavanje, a samim tim i formula mreže postaje zahtevnija za rešavanje. Na primeru potpunog 2-bitnog sabirača (tabela 6.3) se najbolje ogleda značaj korišćenja PMAX-SAT rešavača. Već u

slučaju arhitekture 1×6, QMaxSat nije u mogućnosti da da konačan odgovor u roku od 30 minuta, ali na osnovu broja mekih klauza koje je uspeo u tom periodu da zadovolji, zaključujemo da postoji napredak sa povećanjem broja neurona u skrivenom sloju, iako na osnovu ishoda u slučaju MiniSat rešavača znamo da za pomenutu konfiguraciju nije moguće obučiti mrežu. S obzirom na veličinu formule, za dvocifren broj neurona po sloju, od arhitekture 1×10 se dodaju po dva neurona. Praćenjem trenda rasta zadovoljenih mekih klauza, dolazi se do konfiguracije 1×24 za koju je mreža saglasna sa svim instancama.

TABELA 6.1: Eksperimentalni rezultati za polusabirač

Naziv kola: **polusabirač**							
Broj ulaza: 2 Broj izlaza: 2 Veličina skupa za obučavanje: 4							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
1×2	159	466	unsat	0.001	opt	0.002	3
1×3	231	682	**sat**	0.001	opt	0.002	4

TABELA 6.2: Eksperimentalni rezultati za potpun 1-bitni sabirač

Naziv kola: **potpun 1-bitni sabirač**							
Broj ulaza: 3 Broj izlaza: 2 Veličina skupa za obučavanje: 8							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
1×2	372	1192	unsat	0.002	opt	0.016	5
1×3	542	1752	unsat	0.007	opt	0.027	6
1×4	712	2312	unsat	0.008	opt	0.077	6
1×5	882	2872	unsat	0.008	opt	0.062	7
1×6	1052	3432	unsat	0.007	opt	0.076	7
1×7	1222	3992	**sat**	0.022	opt	0.06	8

U tabelama 6.4 i 6.5 su prikazani rezultati dobijeni za multipleksere, a u tabelama 6.6, 6.7 i 6.8 rezultati za demultipleksere.

U slučaju dekodera, pokazalo se da broj neurona u skrivenom sloju odgovara veličini skupa za obučavanje (tabele 6.9, 6.10 i 6.11). Ušteda na eksperimentima je načinjena u slučaju dekodera $4 - 16$ jer je krenulo od arhitekture 1×8 sa pretpostavkom da ukoliko je to najmanja arhitektura mreže za dekoder $3 - 8$, da jednostavnija mreža ne može biti obučena za istu vrstu kola sa većom dimenzijom.

Prikazani su i rezultati testiranja za komparator dva 2-bitna, tj. 3-bitna broja, čiji izlaz govori da li je prvi broj manji, jednak ili veći od drugog (tabele 6.13 i 6.14). Koristi se ista pretpostavka za arhitekturu mreže kao u slučaju dekodera, zbog čega testiranje za 3-bitni komparator počinje od arhitekture 1×10.

U slučaju sedmodelnog displeja, dovoljna je mreža arhitekture 1×9 (6.15).

Primećujemo da praćenjem napretka obučavanja pomoću podataka dobijenih od PMAX-SAT rešavača, intuitivno možemo oceniti koja bi arhitektura mreže bila zadovoljavajuća, tj. da li je potrebno nastaviti proširenje mreže u širinu ili u dubinu.

TABELA 6.3: Eksperimentalni rezultati za potpun 2-bitni sabirač

Naziv kola: **potpun 2-bitni sabirač**							
Broj ulaza: 5 Broj izlaza: 3 Veličina skupa za obučavanje: 32							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
1×2	2864	10112	unsat	0.013	opt	3.614	13
1×3	4224	15008	unsat	0.052	opt	17.766	16
1×4	5584	19904	unsat	0.121	opt	238.677	17
1×5	6944	24800	unsat	0.186	opt	1270.11	18
1×6	8304	29696	unsat	0.272	-	1800.00	20
1×7	9664	34592	unsat	2.211	-	1800.00	20
1×8	11024	39488	unsat	14.09	-	1800.00	22
1×9	12384	44384	unsat	61.372	-	1800.00	22
1×10	13744	49280	unsat	625.066	-	1800.00	24
1×12	16464	59072	unsat	875.072	-	1800.00	24
1×14	19184	68864	unsat	1601.3011	-	1800.00	25
1×16	21904	78656	-	1800.00	-	1800.00	26
1×18	24624	88448	-	1800.00	-	1800.00	27
1×20	27344	98240	-	1800.00	-	1800.00	28
1×22	30064	108032	-	1800.00	-	1800.00	29
1×24	32784	117824	**sat**	41.89	-	1800.00	30

TABELA 6.4: Eksperimentalni rezultati za multiplekser 2-1

Naziv kola: **multiplekser 2-1**							
Broj ulaza: 3 (1 selektorski) Broj izlaza: 1 Veličina skupa za obučavanje: 8							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
1×2	200	616	**sat**	0.001	opt	0.003	8

TABELA 6.5: Eksperimentalni rezultati za multiplekser 4-1

Naziv kola: **multiplekser 4-1**							
Broj ulaza: 6 (2 selektorska) Broj izlaza: 1 Veličina skupa za obučavanje: 64							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
1×2	2268	8000	unsat	0.013	opt	1.291	48
1×3	3306	11776	unsat	0.049	opt	6.218	56
1×4	4344	15552	**sat**	0.073	opt	19.424	64

Pokazalo se da je mreža sa jednim skrivenim slojem vrlo stabilna, tj. dodavanje neurona dovodi do napretka ili eventualne stagnacije, ali ne i do smanjenja efikasnosti mreže.

U slučaju dubokih mreža, tj. mreža sa više skrivenih slojeva, već sa par desetina instanci u skupu za obučavanje generisanje odgovarajuće instance za rešavač postaje

TABELA 6.6: Eksperimentalni rezultati za demultiplekser 1-2

Naziv kola: **demultiplekser 1-2**							
Broj ulaza: 2 (1 selektorski) Broj izlaza: 2 Veličina skupa za obučavanje: 4							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
1×2	158	464	**sat**	0.001	opt	0.001	4

TABELA 6.7: Eksperimentalni rezultati za demultiplekser 1-4

Naziv kola: **demultiplekser 1-4**							
Broj ulaza: 3 (2 selektorska) Broj izlaza: 4 Veličina skupa za obučavanje: 8							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
1×2	696	2304	unsat	0.002	opt	0.010	6
1×3	1030	3424	unsat	0.007	opt	0.021	7
1×4	1364	4544	**sat**	0.015	opt	0.029	8

TABELA 6.8: Eksperimentalni rezultati za demultiplekser 1-8

Naziv kola: **demultiplekser 1-8**							
Broj ulaza: 4 (3 selektorska) Broj izlaza: 8 Veličina skupa za obučavanje: 16							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
1×2	3176	11136	unsat	0.016	opt	0.423	10
1×3	4736	16640	unsat	0.035	opt	1.52	11
1×4	6296	22144	unsat	0.085	opt	3.993	12
1×5	7856	27648	unsat	0.149	opt	8.999	13
1×6	9416	33152	unsat	0.235	opt	19.855	14
1×7	10976	38656	unsat	0.668	opt	35.778	15
1×8	12536	44160	**sat**	0.423	opt	1.023	16

TABELA 6.9: Eksperimentalni rezultati za dekoder 2-4

Naziv kola: **dekoder 2-4**							
Broj ulaza: 2 Broj izlaza: 4 Veličina skupa za obučavanje: 4							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
1×2	296	900	unsat	0.002	opt	0.006	2
1×3	436	1332	unsat	0.003	opt	0.006	3
1×4	576	1764	**sat**	0.006	opt	0.01	4

TABELA 6.10: Eksperimentalni rezultati za dekoder 3-8

Naziv kola: **dekoder 3-8**							
Broj ulaza: 3 Broj izlaza: 8 Veličina skupa za obučavanje: 8							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
1×2	1356	4552	unsat	0.005	opt	0.339	2
1×3	2018	6792	unsat	0.007	opt	1.36	3
1×4	2680	9032	unsat	0.01	opt	2.525	4
1×5	3343	11272	unsat	0.01	opt	2.364	5
1×6	4004	13512	unsat	0.008	opt	1.726	6
1×7	4666	15752	unsat	0.006	opt	0.62	7
1×8	5328	17992	**sat**	0.113	opt	0.285	8

TABELA 6.11: Eksperimentalni rezultati za dekoder 4-16

Naziv kola: **dekoder 4-16**							
Broj ulaza: 4 Broj izlaza: 16 Veličina skupa za obučavanje: 16							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
1×8	24960	88208	unsat	7.212	opt	725.254	8
1×9	28072	99216	unsat	7.212	opt	725.254	9
1×10	31184	110224	unsat	7.212	opt	725.254	10
1×12	37408	132240	unsat	7.212	opt	725.254	12
1×14	43632	154256	unsat	7.212	opt	725.254	14
1×16	49856	176272	**sat**	6.256	opt	123.328	16

TABELA 6.12: Eksperimentalni rezultati za enkoder 4-2 sa kontrolnim ulazom i izlazom

Naziv kola: **enkoder 4-2 sa kontrolnim ulazom i izlazom**							
Broj ulaza: 5 (1 kontrolni ulaz) Broj izlaza: 3 (1 kontrolni izlaz) Veličina skupa za obučavanje: 32							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
1×2	2879	10142	unsat	0.01	opt	0.653	24
1×3	4239	15038	unsat	0.03	opt	0.331	31
1×4	5599	19934	unsat	0.07	opt	0.396	31
1×5	6959	24830	unsat	0.184	opt	0.911	31
1×6	8319	29726	**sat**	0.109	opt	0.62	32

veoma zahtevno. Za arhitekture 3×3 i 3×4 koje su korišćene za potpun 2-bitni sabirač, generisanje formule je trajalo od 15 do 30 minuta. Takođe, ove formule su izrazito teške za rešavanje, što se može zaključiti na osnovu prikazanih rezultata u tabeli 6.16.

Na ranijim primerima smo mogli primetiti da povećanje broja neurona po sloju

TABELA 6.13: Eksperimentalni rezultati za komparator dva 2-bitna broja

Naziv kola: **komparator dva 2-bitna broja**							
Broj ulaza: 4 Broj izlaza: 3 (<, =, >) Veličina skupa za obučavanje: 16							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
1×2	1244	4272	unsat	0.006	opt	0.159	10
1×3	1834	6336	unsat	0.016	opt	0.586	11
1×4	2424	8400	unsat	0.046	opt	2.229	12
1×5	3014	10464	unsat	0.073	opt	8.226	12
1×6	3604	12528	unsat	0.189	opt	17.744	13
1×7	4194	14592	unsat	0.586	opt	23.425	14
1×8	4784	16656	unsat	4.879	opt	26.638	14
1×9	5374	18720	unsat	18.796	opt	188.648	14
1×10	5964	20784	**sat**	0.282	opt	0.721	16

TABELA 6.14: Eksperimentalni rezultati za komparator dva 3-bitna broja

Naziv kola: **komparator dva 3-bitna broja**							
Broj ulaza: 6 Broj izlaza: 3 (<, =, >) Veličina skupa za obučavanje: 64							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
1×10	31157	113858	-	1800.00	-	1800.00	10
1×12	37337	136514	-	1800.00	-	1800.00	11
1×14	43517	159170	-	1800.00	-	1800.00	12
1×16	49697	181826	-	1800.00	-	1800.00	12
1×18	3604	12528	-	1800.00	-	1800.00	13
1×20	4194	14592	-	1800.00	-	1800.00	14
1×22	68237	249794	**sat**	182.719	-	1800.00	16

dovodi do rasta broja naučenih mekih klauza. Da to ipak nije opšte pravilo, i da višeslojni modeli mogu biti u tom smislu nepredvidljivi, pokazuju eksperimenti prikazani u tabeli 6.16. Dodavanjem samo jednog neurona po sloju, mreža može poprilično izgubiti na kvalitetu, tj. broj instanci sa kojima je saglasna može se naglo smanjiti. Pretpostavka je da povećavanje složenosti mreže otežava rešavanje formule i da je rešavaču potrebno dosta više vremena da nađe rešenje koje zadovoljava isti broj instanci.

6.2.2 Sinteza kombinatornih kola na osnovu nepotpunog uzorka ulaza i izlaza

Prethodni eksperimenti su već pokazali koliki su vreme izvršavanja i veličine arhitektura koje su potrebne za sintezu osnovnih komibinatornih kola date veličine. Pored toga, potrebno je proceniti i koliko je uspešno učenje na osnovu nepotpunog uzorka. Ova procena je izvršena na potpunom 2-bitnom sabiraču, korišćenjem unakrsne validacije. Ukupan skup ima 32 instance, pa je zbog deljivosti pri unakrsnoj

TABELA 6.15: Eksperimentalni rezultati za sedmodelni displej

Naziv kola: **sedmodelni displej**							
Broj ulaza: 4 Broj izlaza: 7 Veličina skupa za obučavanje: 10							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
1×2	1803	6188	unsat	0.011	opt	0.479	4
1×3	2665	9198	unsat	0.056	opt	0.51	6
1×4	3527	12208	unsat	0.072	opt	0.587	6
1×5	4389	15218	unsat	0.12	opt	0.61	7
1×6	5251	18228	unsat	0.43	opt	1.18	8
1×7	6113	21238	unsat	0.662	opt	3.61	9
1×8	6975	24248	unsat	16.157	opt	21.616	9
1×9	7837	27258	**sat**	0.506	opt	1.055	10

TABELA 6.16: Eksperimenti sa više skrivenih slojeva za potpun 2-bitni sabirač

Naziv kola: **potpun 2-bitni sabirač**							
Broj ulaza: 5 Broj izlaza: 3 Veličina skupa za obučavanje: 32							
	formula		**SAT**		**PMAX-SAT**		
arhitektura mreže	broj promenljivih	broj klaza	rešenje	vreme	rešenje	vreme	naučenih instanci
2x2	11704	41600	unsat	2.433	opt	618.542	3
2x3	25554	91616	unsat	104.816	-	1800.00	16
2x4	44784	161216	unsat	632.078	-	1800.00	14
2x5	69394	250400	-	1800.00	-	1800.00	**3**
3x2	47040	167552	unsat	54.419	-	1800.00	12
3x3	153444	551264	-	1800.00	-	1800.00	**4**
3x4	289844	989664	-	1800.00	-	1800.00	**1**

validaciji skup deljen na 8 delova. Preciznost predviđanja izlaza je 44% ukoliko se tačnim smatraju samo ishodi u kojima su oba bita zbira i prenos tačno izračunati. Ukoliko se prihvati delimična tačnost rezultata pri kojoj se smatra da je predviđanje tačno sa težinom proporcionalnom broju bitova koji su tačno izračunati, preciznost predviđanja je 64%. Odavde se vidi da se može pronaći mreža koja je saglasna sa datim instancama, ali da to ne znači da će ona i na drugim instancama odgovarati funkciji koju je potrebno implementirati.

6.2.3 Sinteza kombinatornih kola sa regularizacijom

Postavlja se pitanje da li se ograničavanjem broja nenula koeficijenata mreže za arhitekturu koja se pokazala zadovoljavajućom, može dobiti formula pogodnija za rešavanje i eventualno mreža sa manjim brojem aktivnih neurona. Aktivni neuron je neuron koji u zadovoljavajućoj valuaciji ima bar jedan nenula koeficijent w_k iz formule 5.1, u suprotnom, neuron će nezavisno od ulaza mreže uvek imati vrednost 0, tj. biti neaktivan. Eksperimenti sa regularizacijom su izvršeni na potpunom 2-bitnom sabiraču za arhitekturu mreže 1×26. U slučaju kada nema regularizacije,

obučena mreža ima 107 nenula koeficijenata od ukupno 200 i svi neuroni su aktivni. Može se pretpostaviti da je moguće dobiti mrežu sa neaktivnim neuronima za ovu arhitekturu, s obzirom da je eksperimentima utvrđeno da se i sa 24 neurona u skrivenom sloju može obučiti mreža. U daljim eksperimentima je uključena regularizacija, a dobijeni rezultati su prikazani u tabeli 6.17. Prva kolona se odnosi na parametar regularizacije S koji se bira na osnovu do tada najboljeg poznatog rezultata za broj nenula koeficijenata u mreži, što je u početnom slučaju 107. Ideja je postepeno smanjivati parametar S sve dok SAT rešavač ne utvrdi nezadovoljivost formule sa regularizacijom. Formule za regularizaciju koje se dodaju polaznoj formuli mreže su implementirane na dva načina, pomoću sekvencijalnog brojača (oznaka SEQ) i paralelnog brojača (oznaka PAR). Naredne kolone u tabeli se odnose na dimenzije dobijenih formula mreže u zavisnosti od implementacije koja je korišćena kao i na dobijene rezultate testiranja ovih formula pomoću SAT rešavača. Dodatno, kolone S_{rez} se odnose na broj nenula koeficijenata mreže sa regularizacijom. Smanjivanjem parametra S za očekivati je da u jednom trenutku formula ne može biti zadovoljena, što je u ovom slučaju za $S = 90$. Da bi se potvrdilo da je do tada poznat najbolji rezultat ujedno i donja granica za broj nenula koeficijenata mreže u slučaju arhitekture 1×26, testirana je i mreža sa parametrom regularizacije $S = 91$. S obzirom da je za nju SAT rešavač prijavio nezadovoljivost, zaključujemo da je 92 donja granica za broj nenula koeficijenata mreže koja ima jedan neaktivan neuron. Iz dobijenih rezultata se može zaključiti da je korišćenje paralelnog brojača za implementaciju formule regularizacije značajno efikasnije i da SAT rešavač vrlo brzo može savladati datu formulu u slučaju da je zadovoljiva. Očigledno je da se korišćenjem parametra regularizacije može uštedeti na broju neurona.

TABELA 6.17: Primer regularizacije za potpun 2-bitni sabirač

Naziv kola: **potpun 2-bitni sabirač**
Broj ulaza: 5
Broj izlaza: 3
Veličina skupa za obučavanje: 32
Arhitektura mreže: 1×26
Ukupan broj koeficijenata mreže: 200
Broj nenula koeficijenata obučene mreže: **107**
Vreme izvršavanja bez regularizacije: 31.086

	SEQ		**SAT**			**PAR**		**SAT**		
S	broj promenljivih	broj klaza	rešenje	S_{rez}	vreme	broj promenljivih	broj klauza	rešenje	S_{rez}	vreme
105	55039	164604	sat	103	1601.644	34538	124083	sat	103	28.691
100	54044	162619	sat	99	1245.073	34538	124084	sat	97	28.522
95	53049	160634	sat	94	817.224	34538	124083	sat	**92**	**28.522**
90	52054	158649	-	-	1800.00	34538	124084	unsat	-	104.618
91*	52253	159046	-	-	1800.00	34538	124082	unsat	-	103.824

7 Zaključci i pravci daljeg rada

U ovom radu predložen je jedan diskretan model učenja inspirisan neuronskim mrežama. Razvijena je njegova implementacija i primenjen je na problem sinteze kombinatornih kola na osnovu uzoraka njihovih ulaza i izlaza. Eksperimenti su sprovedeni nad osnovnim kombinatornim kolima: sabiračima, multiplekserima, demultiplekserima, enkoderima, dekoderima i komparatorima. Korišćena je SAT formulacija i PMAX-SAT formulacija problema obučavanja. Eksperimenti pokazuju da se obe mogu koristiti, ali da korišćenje SAT formulacije obično vodi većoj efikasnosti. Prednost korišćenja PMAX-SAT rešavača je u tome što u slučajevima kada se dostigne vremensko ograničenje, SAT ne daje nikakav koristan rezultat, dok PMAX-SAT obično uspe da nađe model koji je saglasan sa delom instanci za obučavanje.

Rezultati pokazuju da se osnovna kombinatorna kola često mogu tačno sintetisati u izrazito kratkom vremenu iz potpunog uzorka ulaza i izlaza, ali i da postoji nezanemarljiv problem u primeni predloženog modela. Ukoliko je broj neurona u mreži manji nego što je dovoljno za uspešno učenje, nekada je potrebno nezanemarljivo vreme da se ustanovi nezadovoljivost formule. U ponavljanju eksperimenata se zbog toga može izgubiti dosta vremena, pre nego što se pronađe adekvatna arhitektura. S druge strane, ukoliko je broj neurona dovoljno veliki i formula može biti vrlo velika po broju promenljivih i klauza. Dodatno, u prikazanom modelu sve klauze koje opisuju arhitekturu mreže se umnožavaju sa svakom instancom. Kako je za komplikovanija kola potrebno imati veliki broj instanci koje se koriste u obučavanju i generisana iskazna formula postaje vrlo velika. U slučaju sinteze iz nepotpunog uzorka, moguće je naći neku mrežu koje je sazlasna sa instancama za obučavanje, ali to ne znači da će ta mreža ispravno naučiti funkciju koju mreža treba da izračunava.

Postoji mogućnost da predloženi pristup nije primenljiv na problem sinteze većih kola u praktičnom kontekstu. Ipak, navedeni problem ukazuje i na pravac u kojem je potrebno uložiti dalje napore kako bi se to eventualno postiglo. Bilo bi poželjno pronaći formulaciju diskretne neuronskre mreže koja ne zahteva ponavljanje svih klauza koje definišu arhitekturu za svaku instancu. Ukoliko se to može postići u okvirima iskazne logike, to bi omogućilo da formule budu manje. Pored toga, postoji mogućnost da korišćeni SAT i PMAX-SAT rešavači ne predstavljaju najbolji izbor i da je moguće naći rešavače koji efikasnije vrše obučavanje. Za to bi bilo moguće konstruisati i portfolio ukoliko bi se prethodno konstruisao adekvatan korpus za njegovo obučavanje.

Literatura

[1] J. Argelich and F. Many$\grave{a}$. "Exact Max-SAT solvers for over-constrained problems". In: *Journal of Heuristics, Volume 12* (2006).

[2] J. Argelich and F. Many$\grave{a}$. "Partial Max-SAT solvers with clause learning". In: *Proceedings of SAT-2007* (2007), pp. 28–40.

[3] F. Many$\grave{a}$ C. M. Li and J. Planes. "Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers". In: *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming* (2005).

[4] F. Many$\grave{a}$ C. M. Li and J. Planes. "New inference rules for Max-SAT". In: *Journal of Artificial Intelligence Research* (2007), pp. 321–359.

[5] J. Clausen. "Branch and Bound Algorithms—Principles and Examples". In: *University of Copenhagen* (1999).

[6] S. A. Cook. "The complexity of theorem-proving procedures". In: *Proceedings of the third annual ACM symposium on Theory of computing* (1971), pp. 151–158.

[7] M. Davis and H. Putnam. "A Computing Procedure for Quantification Theory". In: *Journal of the Association for Computing Machinery, 7* (1960), pp. 201–215.

[8] R. W. Doran. "The Gray Code". In: *University od Auckland* (2007).

[9] J. Harrison. "Handbook of Practical Logic and Automated Reasoning". In: *The United States of America by Cambridge University Press* (2009).

[10] A. Oliveras I. Abio R. Nieuwenhuis and E. Rodriguez-Carbonell. "Bounds to complexities of networks for sorting and for switching". In: *Technical University of Catalonia, Barcelona* (2013).

[11] F. Heras J. Larrosa and S. de Givry. "A logical approach to eficient max-sat solving". In: *Artificial Intelligence* (2008), pp. 204–233.

[12] P. Janičić. "Matematička logika u računarstvu". In: *Matematički fakultet, Univerzitet u Beogradu* (2005).

[13] P. Janičić and M. Nikolić. "Veštačka inteligencija". In: *Matematički fakultet, Univerzitet u Beogradu* (2017).

[14] M. Karnaugh. "The map method for synthesis of combinational logic circuits". In: *Transactions of the Communication and Electronics, American Institute of Electrical Engineers* (1953), pp. 593–599.

[15] H. H. Hoos L. Xu F. Hutter and K. Leyton-Brown. "SATzilla: Portfolio-based Algorithm Selection for SAT". In: *Journal Of Artificial Intelligence Research, Volume 32* (2011), pp. 565–606.

[16] G. Logemann M. Davis and D. Loveland. "A Machine Program for Theorem-proving". In: *Communications of the ACM, 5* (1962), pp. 394–397.

[17] F. Marić M. Nikolić and P. Janičić. "Simple algorithm portfolio for SAT". In: *Artificial Intelligence Review, Volume 40* (2011), pp. 457–465.

[18] J. Marques-Silva and I. Lynce. "Towards Robust CNF Encodings of Cardinality Constraints". In: $CP'07$ (2007), pp. 483–497.

[19] E. J. McCluskey. "Minimization of Boolean Functions". In: *Bell system technical Journal* (1956), pp. 1417–1444.

[20] D. E. Muller and F. P. Preparata. "Bounds to complexities of networks for sorting and for switching". In: *J. AMC 22* (1975), pp. 195–201.

[21] M. Nikolić. "Metodologija izbora pogodnih vrednosti parametara SAT rešavača, Magistarska teza". In: *Matematički fakultet, Univerzitet u Beogradu* (2008).

[22] A. Oliveras R. Asin R. Nieuwenhuis and E. Rodriguez-Carbonell. "Cardinality Networks: a theoretical and empirical study". In: *Constraints, Volume 16* (2011), pp. 195–221.

[23] C. Sinz. "Towards an optimal CNF encoding of boolean cardinality constraints". In: $CP'05$ (2005), pp. 827–831.

[24] G. S. Tseitin. "On the complexity of derivation in propositional calculus". In: *Studies in Constructive Mathematics and Mathematical Logic* (1968), 115—125.

[25] J. M. Wilson. "Compact Normal Forms in Propositional Logic and Integer Programming Formulations". In: *Computers and Operations Research* (1990), pp. 309–314.