

Predrag Janičić

Mladen Nikolić

**VEŠTAČKA INTELIGENCIJA**

Beograd  
2020.

Autori:

dr Predrag Janičić, redovni profesor na Matematičkom fakultetu u Beogradu

dr Mladen Nikolić, docent na Matematičkom fakultetu u Beogradu

## VEŠTAČKA INTELIGENCIJA

...

...

Obrada teksta, crteži i korice: *autori*

©2020. Predrag Janičić i Mladen Nikolić

Ovo delo zašticeno je licencom Creative Commons CC BY-NC-ND 4.0 (Attribution-NonCommercial-NoDerivatives 4.0 International License). Detalji licence mogu se videti na veb-adresi <http://creativecommons.org/licenses/by-nc-nd/4.0/>. Dozvoljeno je umnožavanje, distribucija i javno saopštavanje dela, pod uslovom da se navedu imena autora. Upotreba dela u komercijalne svrhe nije dozvoljena. Prerada, preoblikovanje i upotreba dela u sklopu nekog drugog nije dozvoljena.



---

# Sadržaj

---

<b>Sadržaj</b>	<b>3</b>
<b>1 Uvod</b>	<b>7</b>
1.1 Kratka istorija veštačke inteligencije . . . . .	7
1.2 Znanje i zaključivanje . . . . .	8
1.3 Simbolički i statistički zasnovana veštačka inteligencija . . . . .	8
1.4 Uska i opšta veštačka inteligencija . . . . .	8
1.5 Filozofski i etički aspekti veštačke inteligencije . . . . .	8
<b>I Pretraga</b>	<b>11</b>
<b>2 Rešavanje problema korišćenjem pretrage</b>	<b>13</b>
2.1 Modelovanje problema . . . . .	15
2.2 Rešavanje problema opisanog u matematičkim terminima . . . . .	17
2.3 Interpretiranje i analiza rešenja . . . . .	18
<b>3 Neinformisana pretraga</b>	<b>21</b>
3.1 Obilazak grafa u dubinu i širinu . . . . .	21
3.2 Dejkstrin algoritam . . . . .	26
<b>4 Informisana pretraga</b>	<b>29</b>
4.1 Pohlepna pretraga . . . . .	29
4.2 Pretraga Prvo najbolji . . . . .	35
4.3 Algoritam A* . . . . .	36
<b>5 Igranje strateških igara</b>	<b>49</b>
5.1 Opšte strategije za igranje igara . . . . .	49
5.2 Legalni potezi i stablo igre . . . . .	50
5.3 Otvaranje . . . . .	50
5.4 Središnjica . . . . .	51
5.5 Završnica . . . . .	60
5.6 Monte Karlo pretraga stabla igre . . . . .	61
<b>6 Genetski algoritmi</b>	<b>65</b>
6.1 Opšti genetski algoritam . . . . .	65
6.2 Komponente genetskog algoritma . . . . .	67
6.3 Svojstva genetskih algoritama . . . . .	72
6.4 Primeri primena genetskih algoritama . . . . .	72
<b>II Automatsko rasuđivanje</b>	<b>77</b>
<b>7 Rešavanje problema korišćenjem automatskog rasuđivanja</b>	<b>79</b>
7.1 Modelovanje problema . . . . .	80

7.2 Rešavanje problema opisanog u matematičkim terminima . . . . .	82
7.3 Interpretiranje i analiza rešenja . . . . .	82
<b>8 Automatsko rasudivanje u iskaznoj logici</b>	<b>85</b>
8.1 Sintaksa i semantika iskazne logike . . . . .	87
8.2 Logičke posledice i logički ekvivalentne formule . . . . .	91
8.3 Ispitivanje zadovoljivosti i valjanosti u iskaznoj logici . . . . .	93
8.4 Rešavanje problema svođenjem na SAT . . . . .	99
<b>9 Automatsko rasudivanje u logici prvog reda</b>	<b>111</b>
9.1 Sintaksa i semantika logike prvog reda . . . . .	112
9.2 Logičke posledice i logički ekvivalentne formule . . . . .	119
9.3 Ispitivanje zadovoljivosti i valjanosti u logici prvog reda . . . . .	121
9.4 Rešavanje problema svođenjem na problem valjanosti . . . . .	137
<b>III Mašinsko učenje</b>	<b>145</b>
<b>10 Rešavanje problema korišćenjem mašinskog učenja</b>	<b>147</b>
10.1 Modelovanje . . . . .	149
10.2 Rešavanje . . . . .	152
10.3 Evaluacija . . . . .	153
<b>11 Nadgledano mašinsko učenje</b>	<b>155</b>
11.1 Klasifikacija i regresija . . . . .	155
11.2 Minimizacija greške . . . . .	155
11.3 Preprilagodavanje i regularizacija . . . . .	156
11.4 Linearni modeli . . . . .	162
11.5 Neuronske mreže . . . . .	167
11.6 Modeli zasnovani na instancama: metoda $k$ najbližih suseda . . . . .	175
11.7 Stabla odlučivanja . . . . .	177
11.8 Evaluacija modela i konfigurisanje algoritama učenja . . . . .	182
11.9 Primeri primena nadgledanog učenja . . . . .	187
<b>12 Nenadgledano mašinsko učenje</b>	<b>199</b>
12.1 Klasterovanje . . . . .	199
12.2 Primeri primena klasterovanja . . . . .	204
<b>13 Učenje potkrepljivanjem</b>	<b>207</b>
13.1 Markovljevi procesi odlučivanja i njihovo rešavanje . . . . .	208
13.2 Učenje u nepoznatom okruženju . . . . .	212
13.3 Primene učenja potkrepljivanjem . . . . .	215

---

# Predgovor

---

Ova knjiga nastala je na osnovu beleški koje prate predavanja za predmet *Veštačka inteligencija* koje smo držali akademskih godina od 2007/08 do 2018/19 (prvi autor) i 2019/20, kao i vežbe od 2007/08 do 2011/12 (drugi autor). Dugi niz godina tražili smo, zajedno sa svojim studentima, najbolje načine da predstavimo ključna polja veštačke inteligencije. Nadamo da izbor sadržaja i način prezentovanja mogu da budu zanimljivi ne samo studentima, već i svima drugima koje interesuje ova oblast računarstva.

Zahvaljujemo kolegama koje su nam dale dragocene komentare i savete, pre svega Vesni Marinković, Filipu Mariću, Miroslavu Mariću i Dušku Vitasu, kao i Mirjani Đorić. Zahvaljujemo na veoma pažljivom čitanju, ispravkama i komentarima i studentima koji su slušali predmet „Veštačka inteligencija“, pre svega Nemanji Mićoviću, Mladenu Canoviću, Jani Protić, Vojislavu Stankoviću, Nikoli Premčevskom, Nikoli Ajzenhameru, Neveni Marković, Uni Stanković, Jeleni Simović, Nikoli Dimitrijeviću, Aleksandru Mladenoviću, Vladimиру Simonovskom, Petru Vukmiroviću, Danielu Doži, Ivanu Baleviću, Milanu Kovačeviću, Nebojši Ložnjakoviću, Milošu Samardžiji, Vojislavu Gruiću, Nemanji Antiću, Denisu Aličiću, Miodragu Radojeviću, Dalmi Beara, Mateji Marjanoviću, Đordu Nemetu, Ognjenu Milinkoviću, Milošu Petričkoviću i Maji Gavriloviću. Zahvaljujemo se studentu Kosti Grujčiću na nekim od slika koje smo upotrebili u delu o mašinskom učenju.

U pripremi predavanja i ove knjige koristili smo nekoliko znamenitih knjiga o veštačkoj inteligenciji, ali i mnoštvo drugih izvora koje bi bilo teško pobrojati – često su to bili materijali ili samo pojedinačni komentari sa interneta o specifičnim osobinama algoritama opisanih u knjizi.

Za nedostatke i propuste u knjizi, odgovorni su samo autori.

Ova knjiga dostupna je (besplatno) u elektronskom obliku preko internet strana autora.

*Predrag Janičić i Mladen Nikolić*

Beograd, juli 2020.



## Glava 1

---

# Uvod

---

Veštačka inteligencija jedna je od retkih oblasti nauke o kojoj skoro svi – i eksperti i oni koji to nisu – imaju neki stav. Neko smatra da ona donosi velike koristi, neko smatra da od nje prete opasnosti, a neko veruje i u jedno i u drugo. Neobično je onda da, s druge strane, ne postoji opšta saglasnost o tome šta je uopšte veštačka inteligencija i čime se ona bavi. Pod inteligencijom se obično podrazumeva sposobnost usvajanja, pamćenja i obrade određenih znanja. Iako postoji i shvatanje po kojem je centralni cilj veštačke inteligencije oponašanje ljudske inteligencije, većina podoblasti veštačke inteligencije ima sasvim drugačiji cilj. Obično je to rešavanje problema u kojima se javlja kombinatorna eksplozija, tj. u kojima je broj mogućnosti toliko veliki da se ne može sistematično ispitati u razumnom vremenu. Najveći deo zadataka veštačke inteligencije može se opisati u terminima algoritmike, pretrage, deduktivnog i induktivnog zaključivanja, i drugih preciznih matematičkih pojmoveva. Tek mali deo istraživača bavi se metodama koje pretenduju da dostignu opšte rasuđivanje u stilu čoveka.

Veštačka inteligencija sastoji se od više podoblasti, naizgled slabo povezanih svojim sadržajem. Pitanje je onda šta je to zajedničko za njih, sem to što se uglavnom sve bave problemima u kojima se javlja mnoštvo mogućnosti koje se ne mogu ispitati sistematičnom pretragom. Postoji više pristupa koji objedinjuju sve podoblasti veštačke inteligencije u jedan, jedinstveni okvir. Jedan takav je pristup zasnovan na agentima, na primer, kao u knjizi „Veštačka inteligencija: moderan pristup“ Rasela (Stuart Russell) i Norviga (Peter Norvig). U tom pristupu, sve komponente problema koji se rešava objašnjavaju se u terminima agenata koji imaju neke akcije na raspolaganju. Drugi pristup (korišćen u ovoj knjizi) centralno mesto daje procesu rešavanja problema. Za skoro sve metode veštačke inteligenije zajednička je struktura procesa rešavanja problema. Faze rešavanja su obično: modelovanje, tj. opisivanje zadatog problema u strogim, matematičkim terminima; rešavanje problema opisanog u matematičkim terminima; interpretiranje i analiza rešenja. Ove faze razlikuju se između različitih podoblasti veštačke inteligencije.

### 1.1 Kratka istorija veštačke inteligencije

---

Veštačka inteligencija kao samostalna informatička disciplina prvi put je promovisana na znamenitoj konferenciji *The Dartmouth Summer Research Conference on Artificial Intelligence* organizovanoj u Dartmutu (Sjedinjene Američke Države), 1956. godine. Tom prilikom predloženo je, od strane Džona Makartija, i sâmo ime discipline, ne sasvim srećno sročeno, jer je baš ono često izazivalo nedoumice i podozrenje. Konferencija je trajala mesec dana i bila je pre svega usmerena ka profilisanju nove oblasti. Konferenciju su organizovali Džon Makarti (John McCarthy), Marvin Minski (Marvin Minsky), Nataniel Ročester (Nathaniel Rochester) i Klad Šenon (Claude Shannon). U početnim godinama, pa i decenijama, nizali su se uspesi nove oblasti, a očekivanja su rasla još i brže – ranih šezdesetih, na primer, „mislimo se da je nešto poput računarskog vida (prepoznavanja oblika na slikama) pogodno za letnji projekat studenta master studija“. Minski je 1967. godine izjavio da će problem kreiranja veštačke inteligencije (u stilu ljudske inteligencije) biti rešen u okviru jedne generacije. Ta velika očekivanja ovekovečena su već šezdesetih i sedamdesetih godina u naučno-fantastičnim filmovima i knjigama u kojima računar nadmašuje čoveka u svim intelektualnim aktivnostima. Veštačka inteligencija postala je i polje preko kojeg se stremilo nadmoći u tadašnjem hladnom ratu između Zapada i Istoka. Osamdesetih godina počelo je da dominira shvatanje da su očekivanja od veštačke inteligencije bila previšaka. Fondovi za istraživanja počeli su da se smanjuju i nastupila je takozvana zima veštačke inteligencije. Oblast je, pomalo usporeno, ipak preživela krizu da bi krajem devedesetih godina dvadesetog veka ponovo privukla pažnju čitavog sveta: računari su pobedili svetskog prvaka u šahu i dokazali teoreme koje čovek nije ranije uspevao da dokaže. Bilo je i drugih uspeha, da bi se u dvadeset prvom veku javio talas izuzetno uspešnih sistema zasnovanih na dubokom učenju – sistema koji su uspešno vršili prepoznavanje lica na fotografijama, prevođenje prirodnih jezika, navigaciju

vozila i drugo. Veštačka inteligencija danas je skoro sveprisutna, a opšte je uverenje da će se ta sveprisutnost u budućnosti proširivati i produbljivati.

## 1.2 Znanje i zaključivanje

Za pojam inteligencije suštinske su dve komponente: znanje i zaključivanje. Komponenta zaključivanja predstavlja takođe neku vrstu znanja – to je znanje o procesu izvođenja novog znanja iz raspoloživog znanja. Znanje ovog tipa zvaćemo meta-znanjem. Pod znanjem ćemo podrazumevati i istinite, potvrđene činjenice, ali i hipoteze, nepotpune informacije i informacije date sa određenim verovatnoćama. Zaključivanje može biti deduktivno – zasnovano na rigoroznim opštim pravilima koje daju nove konkretnе činjenice. Zaključivanje može biti i induktivno: u njemu se na osnovu mnoštva činjenica pokušava izvođenje opštih pravila. Postoje i druge forme i drugi okviri zaključivanja.

## 1.3 Simbolički i statistički zasnovana veštačka inteligencija

Mnoge metode veštačke inteligencije zasnovane su na simboličkoj reprezentaciji: i problem i algoritam rešavanja opisani su eksplicitno, a osobine algoritma mogu se analizirati rigorozno, matematički. I opis problema i algoritam su vrlo specifični, prilagođeni jednom konkretnom zadatku. Ti opisi obično koriste teoriju grafova ili matematičku logiku. Za ove metode često se kaže da su GOFAI (od engleskog „Good Old-Fashioned AI“ – „dobra stara veštačka inteligencija“) i čine takozvatni „prvi talas“ veštačke inteligencije. One se najčešće oslanjaju na deduktivno zaključivanje.

Od 2010-ih godina, novi moćni računari omogućili su povratak neuronskih mreža (koje su razvijane još 1950-ih), kroz takozvano „duboko učenje“ i ostvarili fantastične rezultate u mnogim poljima, kao što je računarska vizija, automatsko prevođenje, automatsko upravljanje vozilima, igranje strateških igara, itd. Ovi rezultati zasnovani su na statistici i mašinskom učenju, odnosno na induktivnom zaključivanju. U njima nema eksplicitnog opisivanja procesa rešavanja konkretnih primeraka problema, već se koriste meta-algoritmi kojima se, koristeći raspoložive podatke, kreiraju implicitni algoritmi, modeli i rešenja. U primenama je glavna uloga čoveka u modelovanju problema i pripremi podataka za obučavanje. O ovim implicitnim algoritmima i rešenjima i njihovim osobinama znatno je teže formalno rasuđivati nego o eksplicitnim algoritmima. Moguće je izvesti statističke garancije kvaliteta, ali one u praksi često nisu od velikog značaja. Ovaj pristup čini takozvani „drugi talas“ veštačke inteligencije.

Rasprostranjeno je uverenje da će u budućnosti dosadašnja dva pristupa morati da se integrišu, vodeći do takozvanog „trećeg talasa“ veštačke inteligencije. Očekuje se da će u trećem talasu sistemi veštačke inteligencije samostalno kreirati modele koji će moći da objasne kako stvari funkcionišu.

U ovoj knjizi, simbolički zasnovana veštačka inteligencija obrađuje se u prva dva dela, a statistički zasnovana veštačka inteligencija u trećem delu.

## 1.4 Uska i opšta veštačka inteligencija

Do sada dominantan pravac razvoja veštačke inteligencije je razvoj sistema specijalizovanih za konkretnе zadatke. To je pravac koji se naziva „uska veštačka inteligencija“. Alternativa je „veštačka opšta inteligencija“ (eng. *artificial general intelligence, AGI*). Njen cilj je razvoj jedinstvenog sistema koji može da uči, rasuđuje i rešava sve probleme koje može i čovek. Taj cilj neki istraživači pokušavaju da postignu matematičkim modelovanjem ljudskog mozga, iako mnogi smatraju da je to teško dostižno ili nemoguće. Istraživač i futurista Rej Kercvajl (Raymond Kurzweil, trenutno radi u kompaniji Google), u svojoj uticajnoj knjizi „Singularnost je blizu“ iz 2005, tvrdi da nije daleko trenutak kada će računari prevazići čoveka u svim intelektualnim aktivnostima i procenjuje da će računari već 2029. godine moći da prođu Tjuringov test (videti poglavlje 1.5), što će, dalje, oko 2045. godine dovesti do „sveopštег preokreta u ljudskim mogućnostima“.

## 1.5 Filozofski i etički aspekti veštačke inteligencije

Jedno od ključnih pitanja vezanih za veštačku inteligenciju je da li ona ima za cilj oponašanje prirodne (ljudske ili životinjske) inteligencije. Pitanje koje prethodi ovom pitanju je šta je uopšte inteligencija. Možemo smatrati da inteligencija podrazumeva sledeće sposobnosti: sposobnost pamćenja, skladištenja znanja i mogućnost njegove obrade. Postoje i drugi aspekti inteligencije, kao što su brzina obrade znanja i sposobnost učenja – usvajanja novih znanja, sposobnost komunikacije sa drugim intelligentnim bićima ili mašinama itd. Može se smatrati, dakle, da biće ili mašina imaju attribute intelligentnog, ako imaju navedena svojstva. Alen Tjuring (Alan Turing) formulisao je sledeći znameniti test: ako su u odvojene dve prostorije smeštene jedna ljudska osoba i neki uređaj i ako na identična pitanja pružaju odgovore na osnovu kojih se ne može pogoditi u kojоj

sobi je čovek, a u kojoj uređaj, onda možemo smatrati da taj uređaj ima atribute veštačke inteligencije. Tjuring je 1947. govoreći o računarima koji simuliraju čovekovo rasuđivanje predložio i kreiranje mašina „programiranih da uče i kojima je dopušteno da čine greške“ jer „ako se od mašine očekuje da bude nepogrešiva, onda ona ne može biti inteligentna“. Ovakvo razmatranje u skladu je sa savremenim mašinskim učenjem.

Mnogi problemi veštačke inteligencije mogu se opisati u terminima matematičkih problema. Pitanje je za koje klase problema postoje opšti načini rešavanja (koje mogu da primene ljudi ili mašine). Rezultati Gedela i Tjuringa iz prve polovine dvadesetog veka pokazali su da postoje matematičke teorije (uključujući i jednostavne kakva je aritmetika) koje su nepotpune i neodlučive: postoje tvrđenja o prirodnim brojevima koja su tačna, ali se ne mogu dokazati iz aksioma aritmetike. Štaviše, teoriju aritmetike nije moguće proširiti tako da se to promeni. Ovi rezultati govore da za neke probleme ne mogu da postoje mašine koje mogu da reše svaku njihovu instancu, te da preostaje da se njima bave kao i ljudi: da koriste zapažanja i specičnosti za konkretnе date instance. Drugim rečima, za instance ovakvih problema traganje za rešenjem neće biti uvek isto i neće garantovati uspeh.

Pored filozofskih pitanja o tome gde su granice ljudske i veštačke inteligence, važna su i etička pitanja koja se odnose na mašine koje samostalno mogu da donose nekakve odluke. Još od ranih dana veštačke inteligencije postoji strah od „mašina koje misle“, a sa njihovim razvojem ta pitanja i ti strahovi često samo jačaju. Neke od najčešćih etičkih dilema odnose se na sisteme za autonomnu vožnju. Pitanje je kako definisati ponašanje vozila u slučaju da mora da ugrozi jednog od dva učesnika u saobraćaju. Još pre toga, pitanje je da li takve odluke prepustiti samom autonomnom sistemu, bez eksplisitne odluke ugrađene unapred. Ukoliko dođe do ugrožavanja bezbednosti učesnika u saobraćaju, povreda, ili štete, postavaljaju se i mnoga dodatna, pravna pitanja, na primer – da li za štetu odgovara proizvođač autonomnog automobila ili čovek koji je u njemu sedeo. Mnoga ovakva pitanja će vrlo uskoro biti od praktičnog značaja, a odgovori na njih tek treba da se formulišu.



*Deo I*

---

## Pretraga

---

Elektronsko izdanie (2020)



## Glava 2

# Rešavanje problema korišćenjem pretrage

Veštačka inteligencija bavi se, prevashodno, problemima u kojima se javlja kombinatorna eksplozija, odnosno ogroman broj mogućnosti. Rešavanje takvih problema obično se svodi na neku vrstu pretrage skupa mogućnosti. Kako je često nemoguće sve razmotriti u razumnom vremenu, potrebno je pretragu usmeravati tj. navoditi kako bi se ranije razmotrile mogućnosti za koje je izglednije da vode rešenju problema. Neke od čestih realnih primena algoritama pretrage su pronalaženje najkraćih puteva i planiranje putovanja, navigacija robota, rutiranje u računarskim mrežama, igranje strateških igara, automatsko nalaženje redosleda sklapanja delova u industriji, dizajniranje čipova, dizajniranje proteina sa traženim svojstvima, rešavanje logističkih problema i slično.

Problemi pretrage obično zahtevaju pronalaženje niza *koraka* ili *akcija* kojima se ostvaruje cilj (onda kada to ne može biti ostvareno pojedinačnim koracima ili akcijama). *Rešenje problema pretrage* tada je niz koraka (akcija) koji vodi od polaznog stanja do ciljnog stanja. Pretraga može biti shvaćena i šire, pa će u daljem tekstu biti opisani i neki problemi i algoritmi matematičke optimizacije.

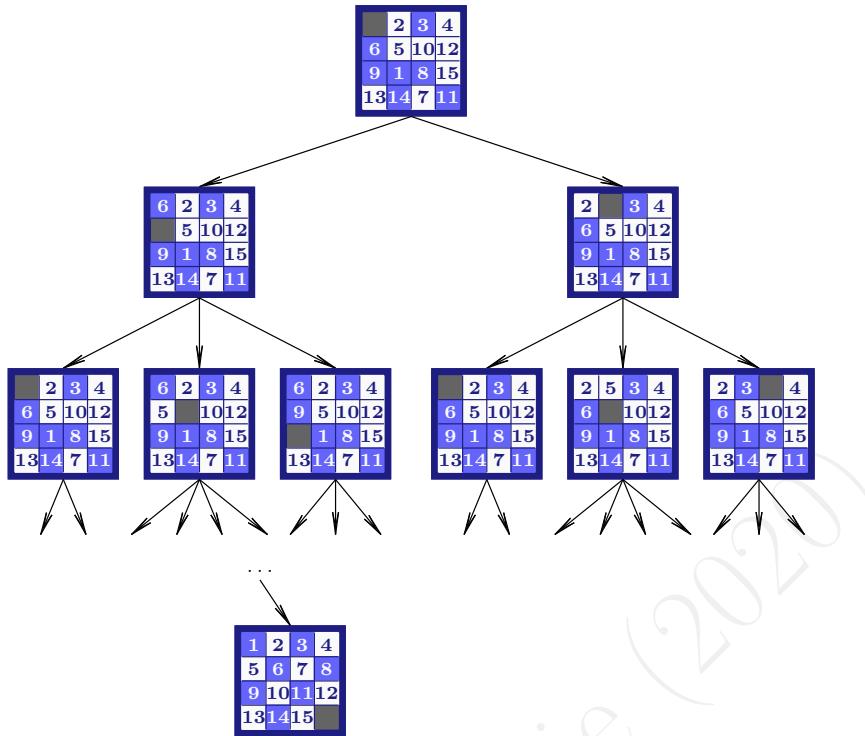
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Slika 2.1: Ciljni raspored za Lojdovu slagalicu.

**Primer 2.1.** Lojdova slagalica (ili „slagalica 15“) sastoji se od 15 kvadrata raspoređenih na tabli veličine  $4 \times 4$  polja. Kvadrati su numerisani brojevima od 1 do 15. Slagalicu je potrebno uređiti tako da su polja poređana redom od prvog reda i da je poslednje polje u četvrtom redu prazno. Taj raspored polja može se kompaktno zapisati kao  $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, \_]$  i prikazan je na slici 2.1.

Kada je dat proizvoljan raspored polja na tabli, u svakom koraku može se pomeriti jedno od dva ili jedno od tri ili jedno od četiri polja. Dakle, za svaki raspored broj mogućih akcija je između dva i četiri.

Slagalicu je moguće složiti tako što se razmatraju svi mogući koraci, a zatim svi mogući koraci u dobijenim stanjima i tako dalje, sve dok se ne nađe na traženi, ciljni raspored (razmatranje svih mogućih koraka za početni raspored  $[\_, 2, 3, 4, 6, 5, 10, 12, 9, 1, 8, 15, 13, 14, 7, 11]$  ilustrovano je na slici 2.2). Očigledno, ovaj pristup sigurno dovodi do rešenja za bilo koju početnu poziciju koja je dobijena regularnim potezima (ako se slagalica rasklopi i poljima 14 i 15 se zamene mesta, onda više nije moguće složiti slagalicu). Očigledno je i da je ovaj pristup potpuno nepraktičan i zahteva razmatranje ogromnog broja mogućnosti. Zaista, za proizvoljnu početnu poziciju, poziciju koja je dobijena od početne pozicije regularnim potezima, slagalicu je moguće složiti u najviše 80 koraka a to je najmanje gornje ograničenje — postoje početne pozicije za koje ne postoji rešenje u manje od 80 koraka. To znači da je za garantovano pronalaženje rešenja potrebno ispitati više od  $2^{80}$  mogućnosti, što je naravno praktično neizvodivo. Zbog toga, praktično sprovodivo rešenje zahteva neku dodatnu ideju i usmeravanje pretrage, kako ne bi bile razmatrane sve mogućnosti. To potrebno



Slika 2.2: Stablo pretrage za Lojdovu slagalicu.

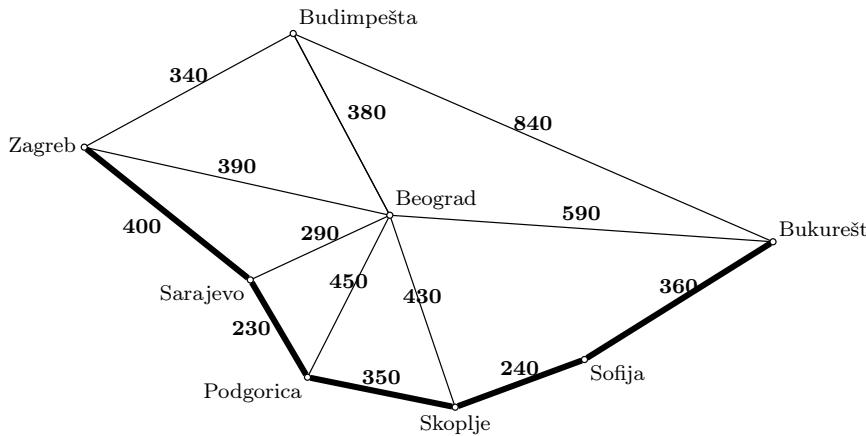
usmeravanje često nije jednostavno. Na primer, ako se u slaganju slagalice razmatraju samo koraci koji vode do pozicija koje su bliže rešenju, pri čemu se za određivanje „rastojanja pozicije od ciljne pozicije“ može uzeti zbir rastojanja svakog polja od njegove ciljne pozicije, time se ne dolazi uvek do rešenja. Naime, u nekim pozicijama nema koraka koji vodi ka boljoj poziciji (pozicija u korenu stabla na slici 2.2, jedna je takva pozicija).

**Primer 2.2.** U skupu gradova od kojih su neki međusobno povezani putevima, zadatak je od jednog grada stići do nekog drugog zadatog grada. Ovaj problem može se razmatrati kao problem pretrage: pretraga može da kreće od početnog grada, da se zatim razmatraju svi gradovi do kojih se može doći neposredno, i tako dalje, sve dok se ne dođe do ciljnog grada. Primer ovakvog problema ilustrovan je na slici 2.3. Konkretni zadatak može biti, na primer, nalaženje puta od Zagreba do Bukurešta.

Od navedenog bitno je razlikit problem u kojem se ne traži bilo koji nego najkraći put između dva grada. I taj problem ima više varijanti – sve dužine između povezanih gradova su unapred poznate ili ne, vazdušna rastojanja između svih gradova su poznata ili ne, itd.

**Primer 2.3.** Knjiga „Propositiones ad Acuendos Juvenes“ („Problemi za britkost mladih“) iz devetog veka n.e., verovatno autora Alkoina iz Jorka (Alcuin of York), jedna je od prvih knjiga popularne matematike. Knjiga sadrži pedesetak zanimljivih problema od kojih su mnogi u opticaju i dan-danas. Jedan od najpoznatijih je problem o vuku, kozi i kupusu: „Jedan čovek kupio je vuka, kozu i kupus. Na putu kući, čovek je došao na obalu reke i iznajmio čamac. U čamcu je pored njega mogao da bude još samo vuk, samo koza ili samo kupus. Ako bi koza i kupus bili na istoj obali i bez prisustva čoveka, vuk bi pojeo kozu. Kako da čovek pređe reku i sačuva sve što je kupio?“

I ovaj problem može se razmatrati kao problem pretrage: pretraga može da kreće od početnog rasporeda (svi su na prvoj obali), zatim se sistematično razmatraju svi mogući koraci (svi mogući prelasci reke), sve dok se ne dobije traženi, ciljni raspored (svi su na drugoj obali).



Slika 2.3: Graf koji opisuje problem puteva između gradova.

**Primeri primena algoritama pretrage.** Algoritmi za usmerenu pretragu koriste se u mnogim oblastima. Algoritam za usmereno traženje puta između dva čvora grafa standardno se koristi u video igrama. Uspešno se koristi i u planiranju putovanja, rutiranju, pa čak i u pomalo neočekivanim oblastima kao što je parsiranje za verovatnosne konteksno-slobodne gramatike, koje imaju primene u obradi prirodnog jezika.

Igranje strateških igara jedno je od klasičnih polja veštačke inteligencije u kojem su postignuti i neki od najznačajnijih rezultata. Godine 1997. specijalizovani računar Deep Blue kompanije IBM pobedio je tadašnjeg svetskog šampiona u šahu Garija Kasparova koristeći efikasne algoritme pretrage zasnovane na alfa-beta odsecanju. Jedan od trenutno najboljih šahovskih programa je Stockfish, takođe zasnovan na alfa-beta pristupu. Tokom prethodnih nekoliko godina pojavili su se i neki veoma uspešni alternativni pristupi, zasnovani na Monte Karlo pretrazi i na neuronskim mrežama, kao što su Komodo MCTS i AlphaZero. Program za igru go, AlphaGo, zasnovan na neuronskim mrežama 2017. godine pobedio je tada prvoplasiranog svetskog igrača Ke Džija.

Genetski algoritmi, koji se mogu smatrati metaheurističkim algoritmom pretrage, imaju primene u širokom spektru optimizacionih problema. Genetski algoritmi koriste se, na primer, za izbor najboljeg plana upita u objektno-relacionim sistemima za upravljanje bazama podataka kao što je PostgreSQL. NASA je početkom ovog veka dizajnirala antene za satelite i svemirske letelice korišćenjem genetskih algoritama koji su se izvršavali na tridesetak umreženih računara. Dizajnirane antene bile su jeftinije, a davale su bolje performanse od drugih. Jedna od konkretnih primena genetskih algoritama je i optimizacija profit-a berzanskih portfolija.

**Faze rešavanja problema korišćenjem pretrage.** Tôk rešavanja problema korišćenjem pretrage obično obuhvata sledeće osnovne faze:

- modelovanje problema;
- rešavanje problema opisanog u matematičkim terminima;
- interpretiranje i analiza rešenja.

## 2.1 Modelovanje problema

Modelovanje problema predstavlja formulisanje problema precizno, u matematičkim terminima, korišćenjem pogodnih matematičkih struktura. Takva formulacija obično ima sledeće elemente:

**Skup mogućih stanja:** U toku procesa pretrage razmatraju se različita *stanja*. U nekim algoritmima, za odlučivanje u nekom trenutku potrebno je poznavanje samo stanja koja su neposredno dostupna, dok je u nekim algoritmima potrebno poznavanje svih stanja unapred.

**Polazno stanje:** Rešavanje problema kreće od jednog određenog stanja, koje nazivamo *polaznim stanjem*.

**Test cilja:** Problem je rešen ako se dođe do ciljnog stanja, *završnog stanja*. Potrebno je da postoji raspoloživ efektivan test koji proverava da li se došlo do ciljnog stanja tj. do završetka procesa pretrage.

**Skup mogućih akcija:** U svakom koraku pretrage može se preduzeti neki korak, neka akcija. Niz akcija preduzetih u odgovarajućim trenucima treba da dovede do rešenja problema. Skup mogućih akcija može biti isti u svakom stanju ili može da se razlikuje od stanja do stanja, što zavisi od problema koji se rešava.

**Funkcija prelaska:** Ova funkcija preslikava par stanje-akcija u novo stanje, dobijeno izborom neke akcije u nekom stanju.<sup>1</sup> Stanja koja su neposredno dostupna iz nekog stanja zovemo i *susednim stanjima*.

**Funkcija cene:** Ova funkcija preslikava par stanje-akcija u numeričku vrednost — cenu preuzimanja date akcije u datom stanju.

Kod nekih problema nabrojani elementi se lako i prirodno uočavaju, dok je kod drugih najpre potrebno problem preformulisati na neki pogodan način.

**Primer 2.4.** Elementi problema iz primera 2.1 mogu biti definisani na sledeći način:

- Skup stanja: *skup svih permutacija*  $[s_1 s_2 \dots s_{16}]$  za  $s_i \in \{\_, 1, 2, \dots, 15\}$ .
- Polazno stanje: *bilo koje stanje slagalice (za neke od njih ciljni raspored nije moguće dobiti)*.
- Test cilja: *provera da li je stanje jednako*  $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, \_]$ .
- Skup mogućih akcija: *za neka stanja može biti {levo, desno, gore, dole}, gde se date akcije odnose na pomeranje praznog polja levo, desno, gore i dole. Iako je naizgled prirodnije kao akcije razmatrati pomeranje kvadrata susednih praznog polja na prazno polje, ovakva formulacija je jednostavnija zbog uniformnosti. Za neka stanja (kada je prazno polje na rubu slagalice) moguće su samo neke dve ili neke tri od navedenih akcija.*
- Funkcija prelaska: *preslikava par stanje-akcija u stanja koja nastaju pomeranjem praznog polja na neku od četiri moguće strane.*
- Funkcija cene: *može biti konstantna za svaku akciju (na primer, 1), pošto se sva pomeranja mogu smatrati jednakim skupom. Cena rešenja je u tom slučaju jednaka ukupnom broju pomeranja potrebnih za slaganje slagalice.*

**Primer 2.5.** Elementi problema stizanja iz jednog grada u drugi (primer 2.2) su:

- Skup stanja: *skup gradova*.
- Polazno stanje: *grad iz kojeg se kreće*.
- Test cilja: *provera da li je tekući grad jednak ciljnog gradu.*
- Skup mogućih akcija: *kretanje ka neposredno dostupnim gradovima (skup mogućih akcija u ovom problemu se razlikuje od stanja do stanja, jer su za različite gradove različiti i skupovi neposredno dostupnih gradova).*
- Funkcija prelaska: *određena je vezama između gradova.*
- Funkcija cene: *na primer, cena za jednu akciju jednaka je dužini puta ili ceni goriva potrebnog za prevoz između susednih gradova.*

Problemi pretrage često se pogodno opisuju u terminima grafova (i mogu se vizualizovati na odgovarajući način). *Graf prostora stanja* onda opisuje skup mogućih stanja i mogućih akcija i svakom čvoru grafa pridruženo je jedno stanje, a svakoj grani jedna akcija. Takav graf može da bude usmeren ili neusmeren. Neusmeren je ako za svako stanje  $A$  iz kojeg se može nekom akcijom doći do stanja  $B$ , postoji odgovarajuća akcija iste cene kojom se iz stanja  $B$  može doći do stanja  $A$ . U primeru slagalice, graf prostora stanja je neusmeren, svakom čvoru grafa pridružen je jedan raspored, a granama odgovaraju pojedinačni koraci. I u primeru gradova, graf prostora stanja je neusmeren, a svakom čvoru pridružen je jedan grad (slika 2.3). Za igru šah, međutim, graf bi bio usmeren (jer postoje pozicije  $A$  i  $B$  takve da se iz  $A$  može jednim potezom doći do  $B$ , ali ne i obratno). Graf prostora stanja može da bude veoma veliki, ali je obično konačan.

<sup>1</sup>Ukoliko ova funkcija nije poznata, nije poznato u koje će se stanje dospeti posle preuzimanja određene akcije i proces odlučivanja postaje kompleksniji. Funkcija prelaska nije poznata, na primer, u slučaju delovanja u nepoznatoj ili promenljivoj okolini. Jedan način rešavanja ovakvih problema je korišćenjem informacija iz iskustva i mašinskog učenja i aproksimiranjem ove funkcije pomoću procesa koji se zasniva na analizi pokušaja i grešaka.

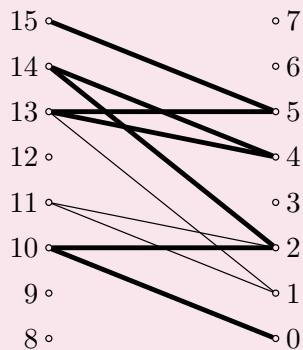
**Primer 2.6.** Modelujmo problem opisan u primeru 2.3:

- Skup stanja je skup mogućih rasporeda na dve obale reke (taj raspored zapravo je određen rasporedom na jednoj obali). Ako je čovek na prvoj obali, označavaćemo to jedinicom, a ako nije – označavaćemo to nulom. Analogno za vuka, kozu i kupus, u tom poretku. Dakle, svako stanje možemo da reprezentujemo nizom od četiri binarne cifre.
- Polazno stanje je stanje 1111.
- Test cilja je provjeri da li je stanje jednako 0000.
- Iz jednog stanja u drugo moguće je doći ako su ispunjeni uslovi: (i) vrednost prve cifre se menja (čovek uvek prelazi sa obale na obalu); (ii) ako je prva cifra prvog stanja jedinica, onda od preostalih jedinica najviše jedna može da postane nula; (iii) ako je prva cifra prvog stanja nula, onda od preostalih nula najviše jedna može da postane jedinica (jer čovek čamcem može da preveze samo vuka, samo kozu ili samo kupusa). U stanja 1100, 1001, 1000, 0111, 0110, 0011 se ne sme doći (da ne bi vuk pojeo kozu ili koza kupusa) i iz njih nema mogućih akcija. Ako zapis stanja tretiramo kao binarni broj, možemo ga kraće označiti i dekadnim brojem.
- Na osnovu mogućih akcija, funkcija prelaska određuje sledeće veze između stanja:

15	→	5	7	–
14	→	4, 2	6	–
13	→	5, 4, 1	5	→ 15, 13
12	–	–	4	→ 14, 13
11	→	2, 1	3	–
10	→	2, 0	2	→ 14, 11, 10
9	–	–	1	→ 13, 11
8	–	–	0	→ 10

- Funkcija cene nije relevantna ako se u zadatku traži bilo kakvo rešenje, ali to se može pouštiti zahtevom za najkraćim rešenjem, tj. rešenjem u kojem ima najmanje prelaženja reke i tada za svaki prelazak možemo smatrati da ima cenu 1.

Opisani problem može se elegantno opisati u terminima grafova. Graf prostora stanja čine čvorovi  $0, 1, \dots, 15$ . Grane su određene navedenom tabelom (dobijeni graf je bipartitni graf). Ako se može doći iz jednog stanja u drugo, onda se i iz tog drugog može doći u prvo. Dakle, graf je neusmeren. Dobijeni graf prikazan je na narednoj slici:



Zadatak u ovako formulisanom problemu postaje: u dobijem grafu pronaći (najkraći) put od čvora 15 do čvora 0.

## 2.2 Rešavanje problema opisanog u matematičkim terminima

U fazi rešavanja često je u grafu prostora stanja potrebno pronaći čvor sa nekim svojstvom ili najkraći put do nekog čvora. Pretraživanjem grafa nastaje stablo pretraživanja ili stablo pretrage (slika 2.2). To stablo ne kreira se eksplicitno, kao struktura u memoriji računara, već samo implicitno procesom obilaska čvorova. U stablu

pretrage svakom čvoru pridruženo je jedno stanje, ali jedno stanje može da bude posećeno više puta tokom obilaska, te može da se nalazi u više čvorova stabla pretrage. Zato stablo pretrage može da bude beskonačno i onda kada je prostor stanja konačan. Kada se kaže „čvor“, obično je iz konteksta jasno da li se misli na čvor prostora stanja ili na čvor u stablu pretrage, a često se isto označavaju čvor i stanje koje mu je pridruženo.

**Primer 2.7.** Problem reprezentovan kao u primeru 2.6, može se rešiti sistematičnim pretraživanjem grafa: početni čvor je 15, ciljni 0. Ako se želi rešenje sa najmanjim brojem prelazaka reke, treba primeniti sistematični obilazak grafa u širinu. Tako se može dobiti sledeće rešenje:  $15 - 5 - 13 - 4 - 14 - 2 - 10 - 0$ .

U mnogim problemima, ciljni čvor u grafu nije moguće pronaći u realnom vremenu sistematičnim pretraživanjem, te je radi bržeg pronalaženja cilja pretragu potrebno usmeravati, navoditi dodatnim informacijama – ukoliko su one raspoložive – o kvalitetu pojedinačnih stanja, specifičnim za dati problem. Probleme pretrage za koje su raspoložive takve informacije zovemo *problem informisane pretrage*, a one druge – *problem neinformisane pretrage*. Algoritmi pretrage dele se, prema tome kojoj grupi problema su prilagođeni, na *algoritme usmerene, informisane pretrage* i na *algoritme neinformisane pretrage*. Algoritmi usmerene, informisane pretrage u nekim slučajevim obezbeđuju isti (ali brže pronađen) rezultat kao i sistematični algoritmi, a u nekim nude rezultate koji su obično dovoljno dobri (na primer, možda neće biti pronađen najkraći put do neke lokacije u gradu, ali će biti pronađen put koji je dovoljno dobar).

**Primer 2.8.** U problemu iz primera 2.2, ako su raspoložive samo informacije o neposredno povezanim gradovima (pa i cene prelaska u njih), u pitanju je problem neinformisane pretrage i za rešavanje se koriste (neinformisani) algoritmi kao što su algoritmi za sistematičnu pretragu u širinu ili u dubinu. Ukoliko su, na primer, poznata vazdušna rastojanja između gradova, te informacije mogu se iskoristiti za navođenje pretrage, pa problem pripada grupi problema informisane pretrage, a za njegovo rešavanje može se koristiti algoritam kao što je  $A^*$ .

## 2.3 Interpretiranje i analiza rešenja

Dobijeno rešenje matematički formulisanog problema potrebno je formulisati u terminima početnog problema i potrebno je razumeti njegova svojstva. U okviru rešavanja problema korišćenjem pretrage ne koristi se evaluacija dobijenih rešenja kakva se koristi, na primer, u mašinskom učenju (videti glavu 10). Nema podataka za trening i podataka za testiranje, niti parametara koje treba naučiti. Naime, algoritmi koji se koriste su obično egzaktni i rešenja koja vraćaju sigurno ispunjavaju neke zadate uslove. U nekim situacijama ipak je potrebno analizirati svojstva korišćenog algoritma, kako bi se znala i preciznija svojstva dobijenog rešenja (na primer, da li je dobijeno rešenje nužno optimalno, a ako nije koliko može da odstupa od optimalnog rešenja). Dodatno, iako su algoritmi koji se koriste za pretragu jasno definisani, neki njihovi parametri mogu biti predmet podešavanja i mogu biti određivani metodama mašinskog učenja. Tada bi taj deo rešavanja problema prolazio kroz faze uobičajene za mašinsko učenje.

Najvažnija opšta svojstva koje algoritmi pretrage mogu da imaju su sledeća:

**Potpunost** je svojstvo koje garantuje da će algoritam naći neko rešenje problema ako rešenja uopšte postoje. Ovo svojstvo je očito poželjno, ali se u nekim slučajevima ne zahteva. Naime, u slučaju vrlo teških problema često je moguće formulisati heuristike koje ne garantuju pronalaženje rešenja, ali u visokom procentu slučajeva nalaze dobra rešenja mnogo brže nego potpuni algoritmi.

**Optimalnost** je svojstvo koje garantuje nalaženje rešenja sa najmanjom cenom (pri čemu se cenom rešenja može smatrati zbir cena akcija koje se preduzimaju)<sup>2</sup>. *Optimalno rešenje* je rešenje sa najmanjom cenom i ono ne mora biti jedinstveno. Moguće je da algoritam koji nema svojstvo optimalnosti često pronalazi rešenja bliska optimalnim, ali u značajno kraćem vremenu.

**Vremenska i prostorna složenost** govore, kao i za druge vrste algoritama, o tome koliko je vremena i memoriskog prostora potrebno za sprovođenje procesa pretrage. Obično se razmatra složenost najgoreg i prosečnog slučaja.

<sup>2</sup>Svojstvo optimalnosti algoritma može da ima i drugačije značenje: da je (vremenska ili prostorna) složenost algoritma najbolja među svim algoritmima koji rešavaju taj problem.

**Primer 2.9.** U okviru primera 2.7, kao rešenje je dat sledeći put u grafu: 15–5–13–4–14–2–10–0. Ako se oznake čvorova grafa prikažu kao binarni brojevi koji označavaju stanja, dobijeni put je: 1111 – 0101 – 1101 – 0100 – 1110 – 0010 – 1010 – 0000. Lako se može rekonstruisati prvi korak algoritma – 1111 – 0101: na prvoj obali su bili čovek, vuk, koza i kupus, a onda samo vuk i kupus. Dakle, u prvom koraku čovek na drugu obalu odvozi kozu. Slično se može rekonstruisati čitavo rešenje u terminima polaznog problema: čovek prevozi kozu–čovek se vraća sâm–čovek prevozi kupus–čovek prevozi kozu–čovek prevozi vuka–čovek se vraća sâm–čovek prevozi kozu.

Ako je rešenje dobijeno obilaskom grafa u širinu, onda je ono najkraće moguće (mogu postojati i druga rešenja iste dužine).



## Neinformisana pretraga

U svim problemima pretrage, podrazumeva se da je moguće opaziti tekuće stanje, preduzimati akcije i prepoznati ciljno stanje. Specifično za „neinformisanu pretragu“ (eng. *uninformed*) je to što nema dodatnih informacija koje mogu pomoći u navođenju koje ubrzava pronalaženje ciljnog stanja. U primeru pronalaženja najkraćeg puta od nekog grada do ciljnog grada, scenario neinformisane pretrage odgovara, na primer, situaciji u kojoj se u svakom gradu zna koji je to grad, moguće je izabrati jedan od puteva ka drugim gradovima (do kojih su poznata rastojanja), moguće je pamtitи posećene gradove i prepoznati odredišni, ali nema informacija o blizini pojedinačnih gradova odredištu (koje bi mogle da usmeravaju pretragu). Tipičan primer problema neinformisane pretrage je i problem labyrintha koji je opisan u nastavku.

**Primer 3.1.** Labyrinth (*u ravni*) sastoji se od skupa povezanih hodnika kojima je moguće kretati se (slika 3.1 (levo)). Svaki hodnik ima jedno ili više polja i dva kraja. Jedno polje je ulaz, a jedno izlaz iz labyrintha. Ulaz, izlaz, krajevi hodnika, kao i polja koja su zajednička za dva hodnika zovemo čvorovima labyrintha. Cilj je pronaći put od ulaza do izlaza preko čvorova labyrintha. Elementi ovog problema su sledeći:

*Skup stanja: skup čvorova labyrintha.*

- *Polazno stanje: ulaz u labyrin.*
- *Test cilja: izlaz iz labyrintha.*
- *Skup mogućih akcija: izbor puta (tj. sledećeg čvora labyrintha) u svakom koraku.*
- *Funkcija prelaska: određena je vezama između čvorova labyrintha.*
- *Funkcija cene: cena za prelazak iz jednog polja u susedno je, na primer, 1.*

Elementi problema pretrage (skup stanja i funkcija prelaska), pa i problema neinformisane pretrage najčešće se prirodno izražavaju pomoću grafova, tako da su i algoritmi neinformisane pretrage najčešće formulisani u vidu algoritama obilaska grafova. U slučaju labyrintha, radi se o grafu čiji čvorovi su čvorovi labyrintha, a grane putevi između tih čvorova labyrintha (slika 3.1 (desno)).

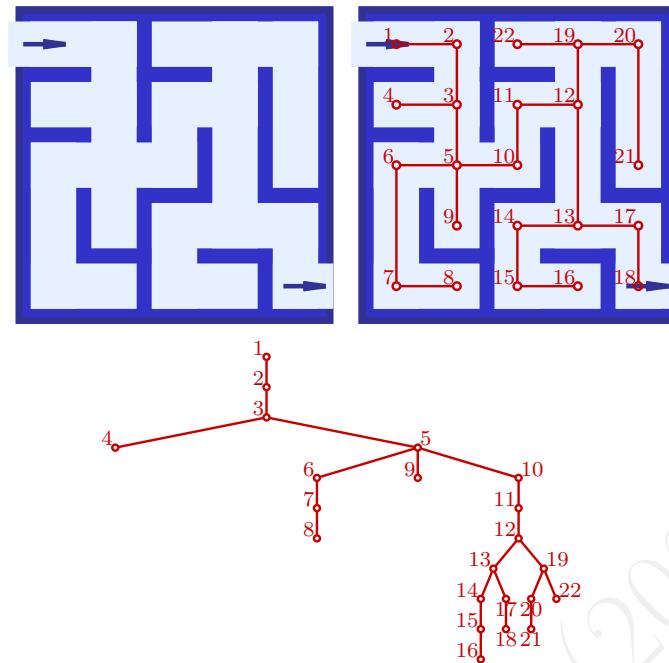
Jedan način pronalaženja izlaza u labyrintru je držati se desne strane hodnika i pratiti zidove dok se ne nađe na izlaz. Kao što će biti navedeno u nastavku, to rešenje odgovara upravo obilasku grafa u dubinu (sve dok se ne nađe na traženi čvor grafa).

### 3.1 Obilazak grafa u dubinu i širinu

Obilazak grafa u dubinu (eng. *depth-first search* — *DFS*) i u širinu (eng. *breadth-first search* — *BFS*) su metode neinformisane pretrage koje ispituju čvorove u grafu tražeći rešenje, obično – neki specifičan čvor. Oni sistematski pretražuju ceo graf bez ikakvog navođenja (i staju sa pretragom kada pronađu traženi čvor). Posebna pažnja biće posvećena *bektrekingu*, jednoj modifikaciji obilaska grafa u dubinu.

#### 3.1.1 Pretraga u dubinu

Pretraga u dubinu je pretraga koja kreće od polaznog čvora i zatim napreduje ka što daljim čvorovima. Ukoliko u nekom čvoru nema suseda koji još uvek nisu ispitani, pretraga se vraća unazad do čvora čiji svi susedi nisu ispitani i nastavlja dalje. U nerekurzivnoj verziji, čvorovi na tekućem putu od polaznog obično se čuvaju



Slika 3.1: Primer labyrintha (gore levo), graf prostora stanja koji odgovara labyrintru (gore desno), stablo koje odgovara obilasku grafa labyrintha u dubinu (dole). Ukoliko se traži izlaz iz labyrintha, pretraga tj. obilazak datog stabla staje nakon dolaska u čvor 18.

na steku, tj. u LIFO listi. Da ne bi došlo do beskonačne petlje, potrebno je čuvati informaciju o čvorovima koji su već posećeni. Ovaj postupak opisan je algoritmom DFS na slici 3.2 a slika 3.3 ilustruje obilazak jednog grafa primenom algoritma DFS. Prikazani algoritam, ukoliko pronađe ciljni čvor, u tom trenutku na steku *put* sadrži redom čvorove koji čine traženi put.

#### Algoritam: DFS (pretraga u dubinu)

**Uzorak:** Graf  $G$ , polazni čvor i ciljni čvor

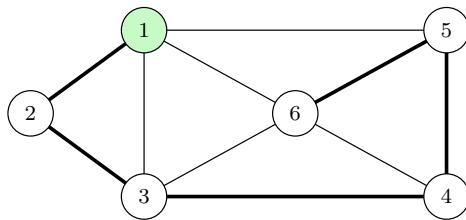
**Izlaz:** Put od polaznog do ciljnog čvora u grafu  $G$  (ako postoji takav put)

- 1: na stek *put* i u skup posećenih čvorova stavi samo polazni čvor;
- 2: **dok god** stek *put* nije prazan **radi**
- 3:   uzmi čvor  $n$  sa vrha steka *put*;
- 4:   **ako** je  $n$  ciljni čvor **onda**
- 5:     izvesti o uspehu i vrati put konstruisan na osnovu sadržaja steka *put*;
- 6:   **ako**  $n$  nema susednih čvorova koji nisu posećeni **onda**
- 7:     izbacи  $n$  sa steka *put*;
- 8:   **inače**
- 9:     izaberi prvi neposećeni susedni čvor  $m$  i dodaj ga na vrh steka *put* i u skup posećenih čvorova;
- 10: izvesti da traženi put ne postoji.

Slika 3.2: DFS — algoritam pretrage u dubinu.

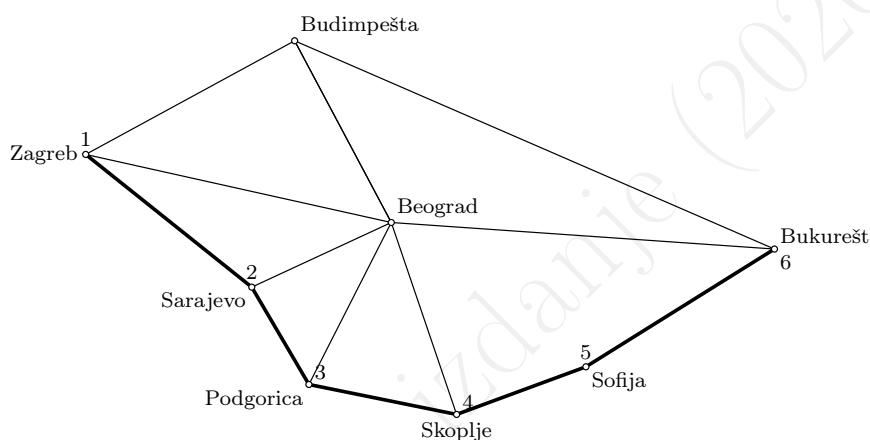
**Primer 3.2.** Algoritam DFS može se upotrebiti za pronalaženje izlaza iz labyrintha koji je opisan grafom (pri čemu algoritam vraća ceo put od ulaza do izlaza). Praćenje hodnika labyrintha držeći se desne strane odgovara pretrazi u dubinu.

**Primer 3.3.** Ukoliko se, pošavši od Zagreba traži put do Bukurešta primenom algoritma DFS i ukoliko



Slika 3.3: Primer obilaska grafa primenom algoritma DFS (oznake čvorova ukazuju na poredak obilaska čvorova).

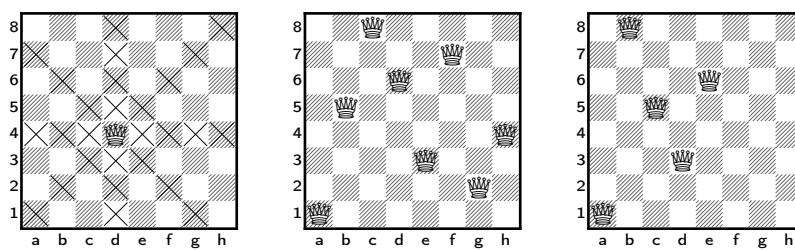
se prilikom izbora sledećeg grada prednost daje južnjem, bio bi pronađen put Zagreb-Sarajevo-Podgorica-Skopije-Sofija-Bukurešt. Ovaj put je po dužini vrlo loš izbor, što je i bilo očekivano pošto algoritam ne uzima u obzir dužine puteva između gradova. Kako se može naći najkraći put, biće prikazano kasnije.



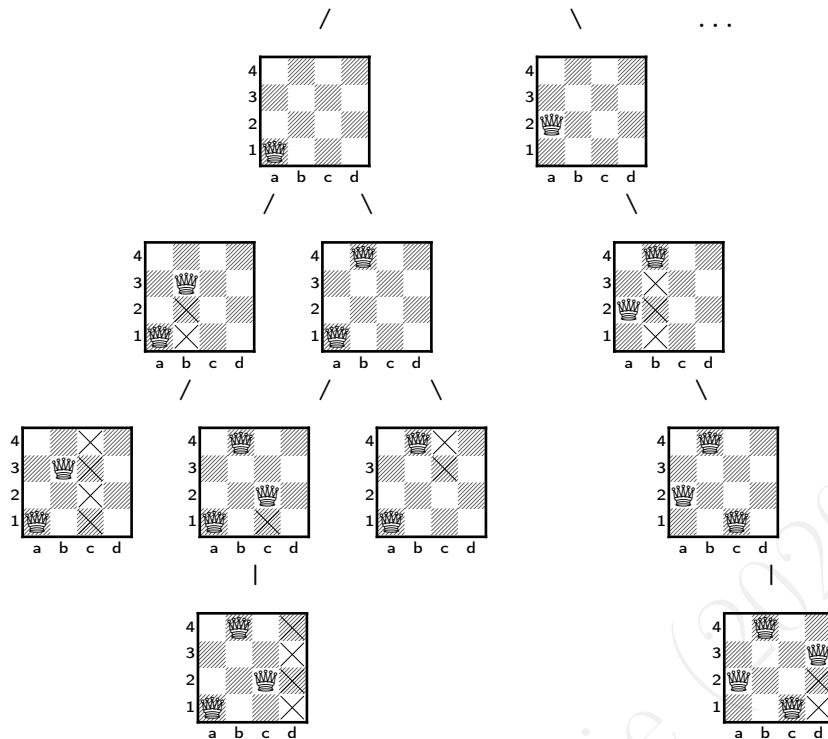
Slika 3.4: Traženje puta od Zagreba do Bukurešta primenom algoritma DFS.

*Bektreking* (eng. *backtracking*) je modifikovana varijanta pretrage u dubinu. Modifikacija se sastoji u tome da se bektrekingom ne mora obići ceo graf, već se napredovanje u dubinu prekida i ranije ako se ustanovi da se ciljni čvor ne nalazi među potomcima tekućeg čvora i tada nastupa vraćanje na prethodni čvor. Prirodan primer za bektreking je rešavanje problema osam dama.

**Primer 3.4.** Problem osam dama formulisan je 1848. godine i od tada je bio predmet mnogih matematičkih i informatičkih istraživanja. Problem ima jednostavnu formulaciju: rasporediti osam dama na šahovskoj tabli tako da se nikoje dve dame ne napadaju. Skup polja koja jedna dama napada definisan je u skladu sa opštim pravilima šaha i ilustrovan je na slici 3.5 (levo). Jedno moguće rešenje ovog problema prikazano je na slici 3.5 (sredina). Problem „n dama“ je uopštenje problema na n dama koje treba rasporediti na tabli



Slika 3.5: Problem osam dama: kretanje dame u šahu (levo), jedno rešenje problema (sredina), situacija kada je u pretrazi nužno vratiti se na prethodni izbor (desno).



Slika 3.6: Prikaz dela stabla pretrage za rešavanje problema četiri dame primenom bektrekinga. Oznaka  $\times$  označava polja na kojima je pokušano postavljanje dame, ali je ustanovljeno da se u tom slučaju napada sa nekom od već postavljenih dama. Ukoliko iz nekog čvora sve četiri mogućnosti vode do neuspeha, onda nema potrebe ispitivati sledeće kolone i dalja pretraga iz tog čvora se prekida.

dimenzija  $n \times n$  tako da se nikije dve ne napadaju.

U svakom rešenju, očigledno, u jednoj koloni ne mogu biti dve dame, pa se problem može preformulisati na sledeći način: na tabli  $n \times n$  rasporediti  $n$  dama tako da u svakoj koloni bude po jedna i da se nikije dve ne napadaju.

Problem dama predstavlja jedan od tipičnih problema koji se rešavaju primenom bektrekinga. Prostor stanja koji se analizira u ovom slučaju čine svi različiti rasporedi od 0 do 8 dama (uključujući i one u kojima se neke dame međusobno napadaju). Postoji grana od jednog stanja (rasporeda) ka drugom ukoliko se drugi može dobiti od prvog dodavanjem jedne dame na slobodno polje na tabli (u prvoj slobodnoj koloni). Neki raspored moguće je dobiti različitim redosledima dodavanja dama polazeći od prazne table, ali se dodavanjem dama ne može dobiti tabla sa manjim brojem dama. Dakle, radi se o usmerenom acikličnom grafu. Polazno stanje je prazna tabla, a ciljno stanje je bilo koje stanje koje zadovoljava uslove problema (za osam dama postoji 92 rešenja).

Opisani graf prostora stanja sadrži i rasporede u kojima se neke dve dame napadaju. U stablu pretrage, na svakom putu od prazne table do takvog rasporeda postoji raspored sa barem dve dame koje se napadaju. Kako se duž puteva kroz stablo pretrage dame samo dodaju, nakon tog rasporeda i svi drugi rasporedi na putu imaju dame koje se napadaju. Dakle, postupak pretrage ne isplati se nastavljati nakon što se naiđe na prvi takav raspored. Na slici 3.5 (desno) prikazan je jedan raspored koji nema smisla ispitivati dalje. Na slici 3.6, prikazan je deo stabla pretrage koja koristi bektreking za problem četiri dame (za problem osam dama stablo pretrage preveliko je za ilustraciju).

Na prethodnom primeru mogu se uočiti neke tipične osobine bektrekinga. Bektreking se zasniva na proširivanju tekućeg parcijalnog rešenja. Polazno parcijalno rešenje je prazno rešenje. U prethodnom primeru, to je prazna šahovska tabla, a proširivanje parcijalnog rešenja se vrši dodavanjem dame na tablu. Proširivanje parcijalnog rešenja u nekim slučajevima nije isplativo ili nije moguće i tada se pretraga vraća unazad, odakle dolazi i ime tehnike. U problemu dama, nije isplativo nastaviti pretragu ukoliko je dostignut raspored u kojem se dve dame napadaju. Prilikom izbora naredne grane u pretrazi, prati se neki poredak izbora, u slučaju problema dama, na primer – sledeće prazno polje u skladu sa nekom numeracijom polja.

### 3.1.2 Pretraga u širinu

Pretraga u širinu razmatra čvorove koji su susedni nekom čvoru, a onda, redom sve njihove susede, pa redom sve njihove susede, i tako dalje. U traganju za čvorom koji zadovoljava neki uslov, biće pronađen onaj na najmanjem rastojanju (pri čemu se pod rastojanjem misli na broj grana) od polaznog čvora. Čvorovi koji se razmatraju obično se čuvaju u redu, tj. u FIFO listi. I u ovom pristupu, da ne bi došlo do beskonačne petlje, potrebno je čuvati informaciju o čvorovima koji su već posećeni. Ovaj postupak opisan je algoritmom BFS na slici 3.7. U prikazanom algoritmu, informacija o posećenim čvorovima ne čuva se eksplicitno, već kroz informaciju o čvoru prethodniku. Slika 3.8 ilustruje obilazak grafa primenom algoritma BFS.

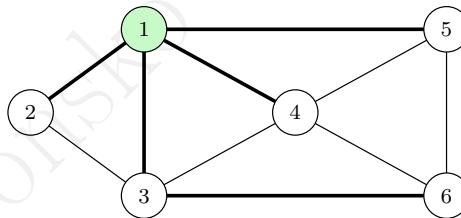
#### Algoritam: BFS (pretraga u širinu)

**Uzorak:** Graf  $G$ , polazni čvor i ciljni čvor

**Izlaz:** Put od polaznog do ciljnog čvora u grafu  $G$  (ako postoji takav put)

- 1: stavi samo polazni čvor u red  $S$ ;
- 2: **dok god** red  $S$  nije prazan **radi**
- 3:     uzmi čvor  $n$  sa početka reda  $S$  i obriši ga iz reda;
- 4:     **ako** je  $n$  ciljni čvor **onda**
- 5:         izvesti o uspehu i vrati put od polaznog do ciljnog čvora (idući unazad od ciljnog čvora, prateći roditeljske čvorove);
- 6:     **za** svaki od susednih čvorova  $m$  čvora  $n$  za koji nije definisan roditelj **radi**
- 7:         zapamti  $n$  kao roditelja i dodaj  $m$  na kraj reda  $S$ ;
- 8: izvesti da traženi put ne postoji.

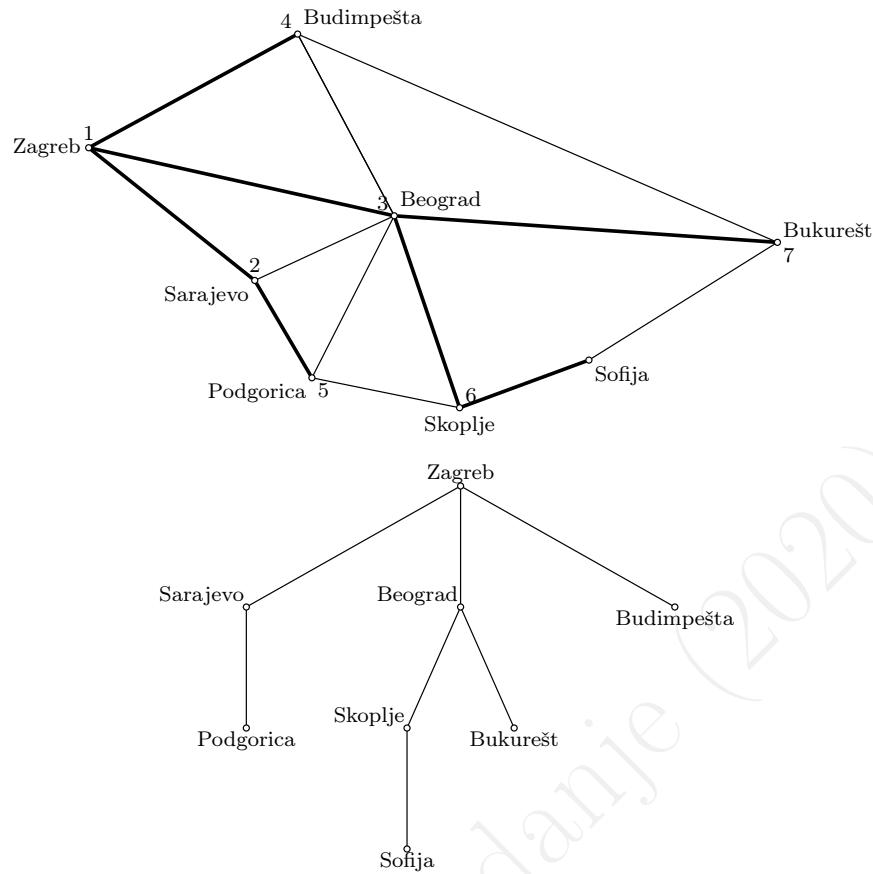
Slika 3.7: BFS — algoritam pretrage u širinu.



Slika 3.8: Primer obilaska grafa primenom algoritma BFS (oznake čvorova ukazuju na poredak obilaska čvorova).

**Primer 3.5.** U slučaju pronalaženja puta od Zagreba do Bukurešta, biće pronađen put Zagreb-Beograd-Bukurešt (slika 3.9). Na početku, tekući grad je Zagreb. Iz Zagreba, pronalaze se Sarajevo, Beograd i Budimpešta (prepostavimo da su tim redom, u smeru suprotnom od smera kazaljke na satu, poređani kao susedi čvora Zagreb). Oni čine novi red  $S$  i za njih se pamti da je prethodni grad Zagreb, koji se uklanja iz reda. Iz Sarajeva se pronalazi put do Podgorice koja se dodaje na kraj reda  $S$ , a Sarajevo se iz njega uklanja. Iz Beograda se pronalazi put do Skoplja i Bukurešta, koji se dodaju na kraj reda  $S$ , a Beograd se iz njega uklanja. Iz Budimpešte se ne pronalazi put ni do jednog grada koji već nije obrađen. Budimpešta se uklanja iz reda. Iz Podgorice se ne pronalazi put ni do jednog grada koji već nije obrađen. Podgorica se uklanja iz reda. Iz Skoplja se pronalazi put do Sofije koja se dodaje na kraj reda  $S$ , a Skoplje se iz njega uklanja. Kada Bukurešt postane grad koji se analizira, konstatiše se da je to ciljni grad, konstruiše se put i algoritam se zaustavlja.

DFS pretraga je pogodnija od BFS pretrage za usmeravanje koje bira čvorove koji više obećavaju. Vremenska složenost oba algoritma je reda zbiru broja čvorova i broja grana grafa koji se pretražuje ( $O(|V| + |E|)$ ), a prostorna je reda broja čvorova ( $O(|V|)$ ).



Slika 3.9: Traženje puta od Zagreba do Bukurešta primenom algoritma BFS: gore je prikazan graf prostora stanja, a dole stablo pretrage.

## 3.2 Dejkstrin algoritam

Dejkstrin algoritam je algoritam za pretragu grafa koji nalazi najkraće puteve u grafu sa nenegativnim cenama koje su pridružene granama. Razvio ga je holandski informatičar Edzger Dejkstra (Edsger Dijkstra) 1959. godine. Algoritam se može koristiti za određivanje najkraćeg puta od datog čvora do datog ciljnog čvora, ali i za određivanje najkraćih puteva od datog čvora do svih drugih čvorova grafa.

Ideja Dejkstrinog algoritma može se, za povezane grafove, ilustrovati na sledeći način. Pretpostavimo da je skup čvorova vezan nitima (i da cennom puta od jednog do drugog čvora smatramo dužinu niti koja ih vezuje). Uzmimo čvor koji je izabran za polazni i počnimo da podižemo celu konfiguraciju (tako da nikoje dve niti nisu upletene). Postepeno se čvorovi, jedan po jedan, odvajaju od tla. Najmanje rastojanje između nekog od tih čvorova i polaznog čvora je upravo direktno rastojanje između njih. Opšta ideja algoritma je slična: postoje čvorovi koji su već podignuti sa tla i oni koji su još uvek na tlu. Za one koji su podignuti sa tla već znamo najkraće puteve od polaznog čvora. Za one koji su još uvek na tlu, smatramo da je rastojanje od polaznog čvora beskonačno. U svakom koraku možemo još jedan čvor „podići sa tla“ i izračunati njegovo najmanje rastojanje od polaznog čvora (razmatrajući samo one čvorove koji su mu susedni i koji su već iznad tla). Ukoliko na kraju ovog postupka na tlu ostanu još neki čvorovi, to znači da do njih ne postoji put od polaznog čvora.

Dejkstrin algoritam prikazan je na slici 3.10. U svakoj iteraciji, bira se čvor  $n$  iz skupa čvorova  $Q$  (to su čvorovi koji su „na tlu“) takav da je vrednost tekućeg najmanjeg rastojanja od polaznog čvora do njega najmanja. Taj čvor se tada briše iz skupa  $Q$ . Ukoliko je to ciljni čvor, onda se konstruiše traženi put od polaznog čvora (koristeći informaciju o roditeljskim čvorovima). Inače, za svaki čvor  $m$  iz  $Q$  koji je susedan čvoru  $n$  proverava se da li se (preko  $n$ ) može popraviti tekuće najmanje rastojanje od polaznog čvora i, ako može, čvor  $n$  se postavlja za roditelja čvora  $m$ . Invarijanta petlje je da se za čvorove koji nisu u  $Q$  zna najkraće rastojanje od polaznog čvora.

U najjednostavnijoj implementaciji Dejkstrinog algoritma, skup  $Q$  se implementira kao obična povezana lista ili niz. Složenost algoritma sa takvom implementacijom skupa  $Q$  je  $O(|V|^2 + |E|) = O(|V|^2)$ , gde je  $V$  skup čvorova, a  $E$  skup grana grafa. Za retke grafove (koji imaju mnogo manje grana od  $|V|^2$ ), Dejkstrin algoritam

**Algoritam:** Dejkstrin algoritam

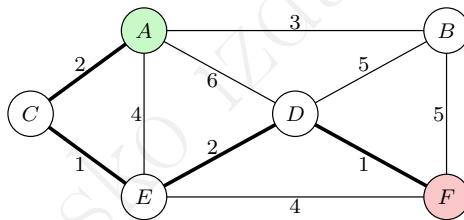
**Ulaz:** Graf  $G$ , polazni čvor i ciljni čvor

**Izlaz:** Najkraći put od polaznog do ciljnog čvora u grafu  $G$  (ako postoji takav put)

- 1: stavi sve čvorove grafa u skup  $Q$ ;
- 2: za sve čvorove iz  $Q$  upamti trenutno najmanje rastojanje od polaznog čvora: za polazni čvor je 0, za sve ostale čvorove je  $\infty$ ; za svaki čvor zapamti da je roditelj nedefinisan;
- 3: **dok god** skup  $Q$  nije prazan **radi**
- 4: izaberi iz  $Q$  čvor  $n$  sa najmanjim ustanovljenim rastojanjem od polaznog čvora i obriši ga iz  $Q$ ;
- 5: **ako je**  $n$  ciljni čvor **onda**
- 6: konstruiši put od polaznog do ciljnog čvora (idući unazad od ciljnog čvora, prateći roditeljske čvorove) i izvesti o uspehu;
- 7: **za** svaki čvor  $m$  iz  $Q$  koji je direktno dostupan iz  $n$  **radi**
- 8: **ako** je tekuće rastojanje od polaznog čvora do  $m$  veće od rastojanja od polaznog čvora do  $m$  preko čvora  $n$  **onda**
- 9: promeni informaciju o roditelju čvora  $m$  na čvor  $n$  i upamti novo rastojanje;
- 10: izvesti da traženi put ne postoji.

Slika 3.10: Dejkstrin algoritam.

može se implementirati efikasnije. Na primer, varijanta koja koristi binarni min-hip<sup>1</sup> za određivanje tekućeg najbližeg čvora ima složenost  $O((|E| + |V|) \log |V|)$ .



Slika 3.11: Primer primene Dejkstrinog algoritma.

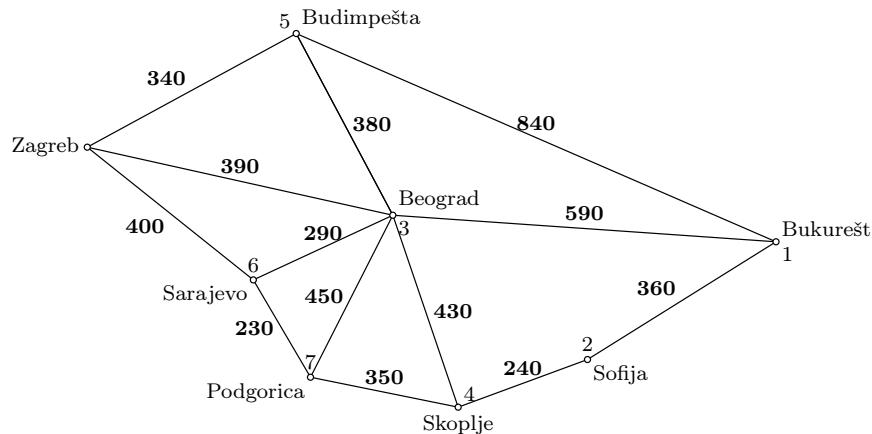
**Primer 3.6.** Naredna tabela prikazuje efekat primene Dejkstrinog algoritma na graf prikazan na slici 3.11:

korak	B	C	D	E	F	čvor $n$ (roditelj)
1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	A
2	3	<b>2</b>	6	4	$\infty$	C (A)
3	<b>3</b>		6	3	$\infty$	B (A)
4			6	<b>3</b>	8	E (C)
5			<b>5</b>		7	D (E)
6					<b>6</b>	F (D)

U ovom primeru, čvor  $A$  je polazni, a čvor  $F$  ciljni čvor. Polje tabele za neki čvor prikazuje vrednost najkraćeg do tada nađenog rastojanja od polaznog do tog čvora.

**Primer 3.7.** Naredna tabela ilustruje izvršavanje Dejkstrinog algoritma na problemu nalaženja puta od

<sup>1</sup>Min-hip je specifična stablolika struktura koja zadovoljava hip-svojstvo: ako je  $B$  sused čvora  $A$ , onda je vrednost pridružena čvoru  $A$  manja od vrednosti pridružene čvoru  $B$ . U skladu sa tim, najmanji element je uvek koren stabla. Max-hip se definije analogno.



Slika 3.12: Traženje puta od Bukurešta do Podgorice primenom Dejkstrinog algoritma.

Bukurešta do Podgorice (najkraći put je Bukurešt-Sofija-Skoplje-Podgorica, slika 3.12).

korak	<i>Bg</i>	<i>So</i>	<i>Bud</i>	<i>Sk</i>	<i>Pg</i>	<i>Sa</i>	<i>Zg</i>	čvor n (roditelj)
1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	Bukurešt
2	590	<b>360</b>	840	$\infty$	$\infty$	$\infty$	$\infty$	Sofija (Bukurešt)
3	<b>590</b>		840	600	$\infty$	$\infty$	$\infty$	Beograd (Bukurešt)
4			840	<b>600</b>	1040	880	980	Skoplje (Sofija)
5			<b>840</b>		950	880	980	Budimpešta (Bukurešt)
6					950	<b>880</b>	980	Sarajevo (Beograd)
7					<b>950</b>		980	Podgorica (Skoplje)

## Informisana pretraga

Informisana (ili heuristička) pretraga koristi ne samo informaciju o mogućim akcijama (koracima) u svakom stanju, već i dodatno znanje o konkretnom problemu koje može da usmerava pretragu ka stanjima koja više obećavaju, za koje postoji nekakvo očekivanje da brže vode ciljnog stanju, tj. rešenju problema. Ta informacija može biti nekakva ocena, mera „kvaliteta“ stanja, a može da bude zasnovana i na informacijama vezanim za početno ili ciljno stanje. Ta mera kvaliteta često nije egzaktna, nego predstavlja nekakvu procenu, heurističku meru.<sup>1</sup>

Funkciju koja ocenjuje kvalitet stanja zovemo *funkcija evaluacije*. Ukoliko je funkcija evaluacije označena sa  $f$ , onda  $f(n)$  označava ocenu stanja  $n$ . Podrazumevaće se da su cene akcija (ili cene grana grafa) nenegativne. Već je rečeno da se problemi pretrage često mogu pogodno zadati u terminima grafova koji opisuju prostor stanja, pa će se umesto „stanja“ i „ocena stanja“ govoriti i „čvor“ i „ocena čvora“.

Prilikom rešavanja problema pretragom, generiše se stablo pretrage (obično samo implicitno) čijim su čvorovima pridružena stanja. Pošto kroz jedno isto stanje može da se prođe više puta tokom pretrage, može da bude više čvorova stabla pretrage sa istim tim stanjem. Pošto ocena stanja može da zavisi od trenutnog konteksta procesa pretrage, obično je preciznije reći „ocena čvora (stabla pretrage)“ nego „ocena stanja“.

### 4.1 Pohlepna pretraga

Pohlepnim algoritmom naziva se algoritam koji teži neposrednom povećanju vrednosti neke ciljne funkcije. Ovakav algoritam ne procenjuje dugoročni kvalitet izabranih akcija, tj. koliko one doprinose ostvarenju konačnog cilja, već bira akciju koja se na osnovu znanja dostupnog u trenutku izbora procenjuje kao najbolja među raspoloživim akcijama. Postoji više varijanti pohlepnih algoritama za raznovrsne namene, a jedna varijanta pohlepne pretrage u grafu prikazana je na slici 4.1, gde je sa  $f(n)$  označena funkcija evaluacije, funkcija koja usmerava rad algoritma, tj. funkcija koja procenjuje kvalitet raspoloživih akcija.

**Primer 4.1.** U primeru pronalaženja najkraćih puteva između gradova (primer 2.2), ukoliko su, na osnovu mape, poznata sva vazdušna rastojanja između gradova, pohlepni informisani algoritam kao funkciju evaluacije  $f(n)$  može da koristi vazdušno rastojanje od  $n$  do ciljnog grada (tj. da uvek za sledeći grad bira onaj koji je najbliži ciljnog gradu vazdušnim putem). Ukoliko se traži put od Podgorice do Budimpešte (slika 4.2) i ukoliko su iz Podgorice neposredno dostupni Sarajevo, Beograd i Skoplje, u prvom koraku biće izabran Beograd.

Ako je u svakom gradu moguće videti i tablu sa tačnim rastojanjima do susednih gradova i poznata su vazdušna rastojanja između gradova, pohlepni informisani algoritam bi mogao da koristi i napredniju funkciju evaluacije  $f(n)$  – zbir tačnog rastojanja od tekućeg čvora do čvora  $n$  i vazdušnog rastojanja od  $n$  do ciljnog grada. Tada se, u svakom koraku, kao sledeći grad, kao grad koji najviše obećava, bira grad  $n$  za koji je zbir tačnog rastojanja od tekućeg grada do  $n$  i vazdušnog rastojanja od  $n$  do ciljnog grada najmanji. Time se put i dalje traži pohlepno, ali se, u nekoj meri, razmatra i celokupna dužina puta. Ovaj pristup

<sup>1</sup>Heuristike su tehnike za usmeravanje i sužavanje pretrage u problemima u kojima se javlja kombinatorna eksplozija. Reč „heuristika“ potiče od grčke reči „heurisko“ koja znači „tražiti“ ili „otkrivati“. Srodna grčka reč „heureka“ ili „eureka“ znači „našao sam“ ili „otkrio sam“ i obično se vezuje za Arhimeda i njegov uzvik kada je došao do jednog znamenitog otkrića. Aristotel je koristio termin „heuristika“ za otkrivanje novog znanja (ili demonstriranje postojećeg) kroz komunikaciju i interakciju između izlagača i slušalaca. Perl (1984) pod heuristikama smatra „kriterijume, metode ili principe za izbor između nekoliko mogućih akcija one akcije koja obećava da će biti najkorisnija za postizanje nekog cilja“.

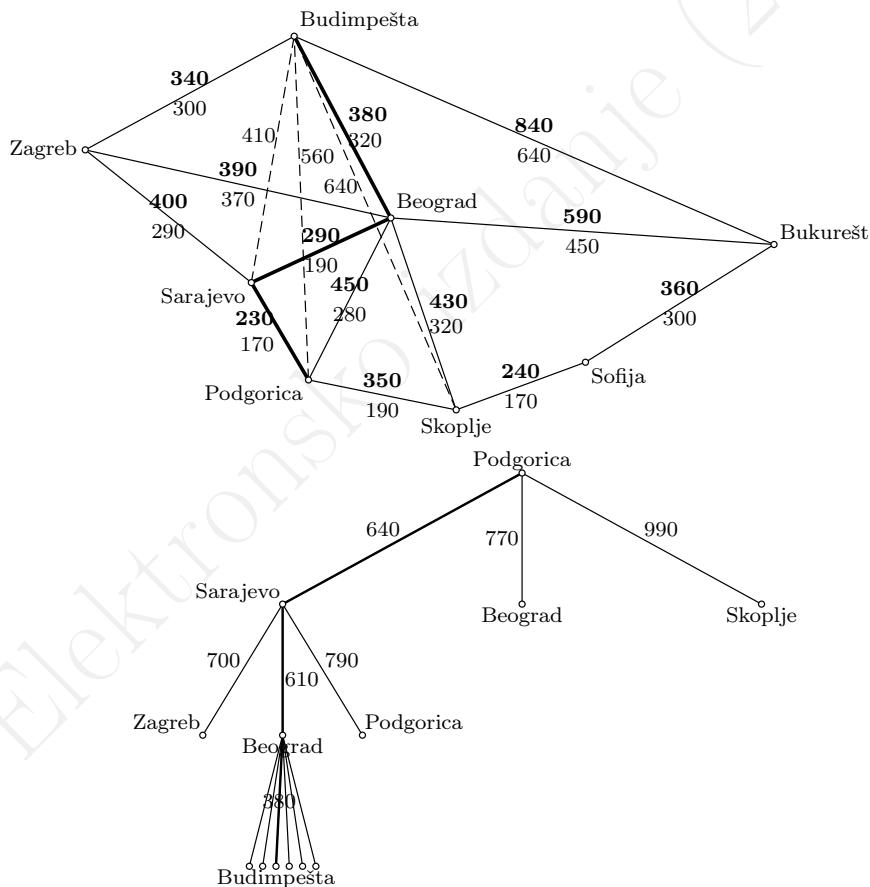
**Algoritam:** Pohlepna pretraga u grafu

**Ulaz:** Graf  $G$ , polazni čvor i ciljni čvor

**Izlaz:** Niz koraka od polaznog do ciljnog čvora ili neuspeh (i niz koraka do najboljeg pronađenog čvora)

- 1: tekući čvor  $n$  postavi na polazni čvor;
- 2: **ponavljam**
- 3:   **ako** je  $n$  ciljni čvor **onda**
- 4:       izvesti o uspehu i vrati rešenje konstruišući put od polaznog do ciljnog čvora;
- 5:   **ako** nema nijednog čvora  $m$  koji je direktno dostupan iz tekućeg čvora i ima bolju ocenu  $f(m)$  od tekućeg čvora **onda**
- 6:       izvesti o neuspehu i vrati tekući čvor kao najbolji pronađeni (i konstruiši put od polaznog do tog čvora);
- 7:   **inace**
- 8:       od čvorova koji su direktno dostupni iz tekućeg čvora izaberi čvor  $m$  koji ima najbolju ocenu  $f(m)$ , zapamti  $n$  kao njegovog roditelja i postavi  $m$  da bude novi tekući čvor  $n$ .

Slika 4.1: Algoritam pohlepne pretrage.

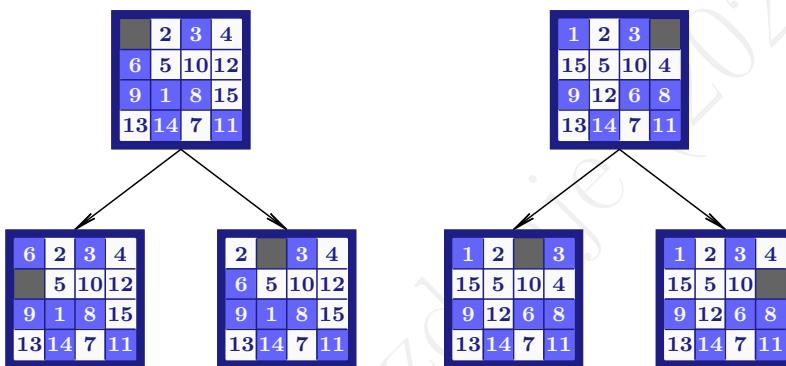


Slika 4.2: Traženje puta od Podgorice do Budimpešte primenom pohlepnog algoritma: na slici gore je prikazan graf koji opisuje prostor stanja, a na slici dole prikazano je stablo pretrage. Na prvom grafu, podebljanim ciframa ispisana su kopnena, običnim ciframa vazdušna rastojanja između dva grada, a isprekidanim linijama spojeni su gradovi za koje ne postoje direktna kopnena veza.

biće ilustrovan primerom traženja puta od Podgorice do Budimpešte (slika 4.2). Iz Podgorice su neposredno dostupni Sarajevo, Beograd i Skoplje, a ocene dužina puteva do cilja preko tih gradova su  $230\text{km}+410\text{km} = 640\text{km}$ ,  $450\text{km}+320\text{km} = 770\text{km}$  i  $350\text{km}+640\text{km} = 990\text{km}$ , te se ide u Sarajevo. Iz Sarajeva su neposredno

dostupni Zagreb, Beograd i Podgorica, a ocene dužina puteva preko tih gradova su  $400\text{km}+300\text{km} = 700\text{km}$ ,  $290\text{km}+320\text{km} = 610\text{km}$  i  $230\text{km}+560\text{km} = 790\text{km}$ , te se ide u Beograd. Iz Beograda su neposredno dostupni Sarajevo, Zagreb, Budimpešta, Bukurešt, Skoplje, Podgorica, a ocene dužina puteva preko tih gradova su  $290\text{km}+410\text{km} = 700\text{km}$ ,  $390\text{km}+300\text{km} = 690\text{km}$ ,  $380\text{km}+0\text{km} = 380\text{km}$ ,  $590\text{km}+640\text{km} = 1230\text{km}$ ,  $430\text{km}+640\text{km} = 1070\text{km}$  i  $450\text{km}+560\text{km} = 1010\text{km}$ , te se ide u Budimpeštu. Pronađeni put je, dakle, Podgorica-Sarajevo-Beograd-Budimpešta i njegova stvarna (kopnena) dužina je 900km. Međutim, stvarna (kopnena) dužina puta Podgorica-Beograd-Budimpešta je manja i iznosi 830km, što znači da napravljeni izbor nije najbolji mogući.

Navedeni primer ilustruje opšti problem pohlepne pretrage: pohlepni algoritmi ne garantuju optimalnost, pa ni potpunost procesa rešavanja. Naime, moguće je da pohlepni algoritam vrati rešenje koje nije najbolje ili da ne nađe put do ciljnog čvora i ako on postoji. Dobra strana algoritama zasnovanih na pohlepnoj pretrazi je to što su veoma jednostavni, ponekad veoma efikasni i mogu da daju dovoljno dobre rezultate. Pohlepna pretraga obično se ponaša dobro u slučaju problema kod kojih kvalitet odluke u nekom stanju pretrage ne zavisi od budućih odluka. Pohlepni algoritmi nikad se ne vraćaju u stanje u kojem su već bili. To svojstvo, u slučaju konačnih grafova, obezbeđuje zaustavljanje algoritma.



Slika 4.3: Stanje slagalice (levo) u kojem nema poteza koji vodi u stanje sa boljom ocenom rastojanja i stanje slagalice (desno) iz kojeg postoji niz koraka koji poboljšavaju ocenu rastojanja (pomeranjem praznog polja nadole), ali se završava u stanju iz kojeg svi potezi pogoršavaju ocenu.

**Primer 4.2.** U slučaju Lojdove slagalice, kao ocena rastojanja od tekućeg do ciljnog stanja može se koristiti zbir Menhetn rastojanja svakog od 15 polja slagalice do njegovog ciljnog mesta. Menhetn rastojanje između dva polja  $A$  i  $B$  definiše se kao najmanji broj polja koji je potrebno preći kako bi se došlo od  $A$  do  $B$ , krećući se isključivo horizontalno ili vertikalno (ovo rastojanje zove se Menhetn, jer podseća na kretanje ulicama Menhetna koje su međusobno normalne ili paralelne: od jednog do drugog bloka moguće je kretati se ulicama, ali nije moguće prolaziti blokove diagonalno). U slučaju stanja slagalice u korenu levog stabla na slici 4.3, Menhetn rastojanje polja 1 do njegovog pravog mesta je 3, zato što je na tom putu potrebno preći preko dva polja krećući se naviše, a potom jedno polje krećući se uлево. Mogući su i drugi putevi, ali njihova dužina nije manja. Ukupna ocena tog stanja slagalice je  $0+0+0+1+1+2+1+0+3+2+2+0+0+2+2=16$ . Algoritam pohlepne pretrage prikazan na slici 4.1. traži potez kojim ocena stanja može da se popravi.

U slučaju stanja slagalice u korenu slike 4.3 (levo), pohlepna pretraga ne može da nastavi. Naime, dato stanje predstavlja lokalni minimum ocene stanja, jer se bilo kojim potezom ta ocena uvećava za 1.

S druge strane, ukoliko je polazna konfiguracija 4.3 (desno) ocena stanja se smanjuje pomeranjem polja 4 naviše a zatim i pomeranjem naviše polja 8 i 11. Nakon toga se dolazi do stanja koje je lokalni minimum i pretraga se zaustavlja.

Navedeni primjeri pokazuju da predloženom jednostavnom pohlepnom pretragom nije moguće rešiti Lojdovu slagalicu.

#### 4.1.1 Penjanje uzbrdo

Slični po duhu pohlepnoj pretrazi su algoritmi *penjanja uzbrdo*. Oni imaju svojih specifičnosti, a ponekad se smatraju vrstom pohlepne pretrage.

Algoritmi penjanja uzbrdo pripadaju grupi algoritama *lokalne pretrage* za matematičku optimizaciju. Zadatak je pronaći vektor-argument  $\mathbf{x}$  koji daje maksimalnu vrednost neke funkcije cilja  $f(\mathbf{x})$ . Moguće vrednosti tog

argumenta čine *skup dopustivih rešenja*, a tražena vrednost argumenta naziva se *optimalnim rešenjem* problema optimizacije. Algoritam treba da kao rezultat uvek vrati neko dopustivo rešenje (i ako ne može da tvrdi da je to rešenje zaista maksimalno, tj. optimalno). Inicijalno rešenje bira se obično slučajno. U svakoj iteraciji bira se novo tekuće rešenje, ali samo iz specifične *okoline* tekućeg rešenja, tj. iz nekog skupa suseda. Taj postupak se nastavlja dok god može da se popravi kvalitet tekućeg rešenja. Funkcija  $f(\mathbf{x})$  ne mora biti neprekidna, već može biti i diskretna i tada se problem optimizacije može razmatrati i kao problem pretrage nad grafom.

Algoritmi penjanja užbrdo obično su veoma jednostavni i često dovoljno uspešni, ali imaju i slabosti. Penjanje užbrdo ne garantuje pronalaženje optimalne vrednosti. Na primer, algoritam može doći do *lokalnog maksimuma* (kada je u svakom susednom stanju vrednost funkcije cilja manja od tekuće), završava rad i vraća tekuću vrednost  $\mathbf{x}$ , bez ikakve garancije da ne postoji bolji lokalni maksimum ili drugačiji globalni maksimum. Slično, algoritam može da se nađe na *platou* (kada je u svakom susednom stanju vrednost funkcije cilja jednaka tekućoj), završava rad jer ne može da odredi sledeću iteraciju i vraća tekuću vrednost  $\mathbf{x}$ .

Postoje razne varijacije osnovnog penjanja užbrdo koje pokušavaju da se izbore sa navedenim problemima. Takvo je, na primer, *stohastičko penjanje užbrdo* u kojem je dozvoljeno u nekim iteracijama birati i stanje koje nije susedno, a kako bi se povećale šanse da se pređe u deo prostora pretrage u kojem se nalazi traženi maksimum ciljne funkcije.

Potpuno analogni algoritmima koji na opisani način traže maksimum ciljne funkcije su algoritmi koji traže minimum ciljne funkcije.

#### 4.1.2 Matematička optimizacija u slučaju diferencijabilne funkcije cilja i metode gradijentnog uspona i spusta

Metode pretrage i matematičke optimizacije često pokušavaju da iskoriste neku zakonitost u strukturi prostora pretrage, odnosno prostora dopustivih rešenja. U slučaju diferencijabilne ciljne funkcije, često se koristi metod koji je sličan opštem metodu penjanja užbrdo, a koristi gradijent ciljne funkcije. Ukoliko je data diferencijabilna funkcija  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , gradijent se definiše kao vektor parcijalnih izvoda te funkcije:

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

Gradijent izračunat u određenoj tački  $\mathbf{a} \in \mathbb{R}^n$  predstavlja vektor u prostoru  $\mathbb{R}^n$  u čijem smeru funkcija  $f$  najbrže raste u okolini tačke  $\mathbf{a}$ . Stoga se pravljenjem koraka u ovom smeru može približiti lokalnom maksimumu, a kretanjem u suprotnom smeru – lokalnom minimumu. S obzirom na to da se ne radi o diskretnom prostoru, u praksi se obično ne očekuje pronalaženje sâme tačke lokalnog maksimuma, te se opisani postupak zaustavlja kada razlika u vrednosti funkcije  $f$  u odnosu na njenu prethodnu vrednost postane dovoljno mala. Ako se primeni kretanje u smeru najbržeg rasta, može se desiti da se „ode predaleko“ i da je vrednost funkcije  $f$  u novoj iteraciji lošija od vrednosti u tekućoj iteraciji, te metode zasnovane na gradijentu nisu, u strogom smislu, vrsta penjanja užbrdo. Na slici 4.4 prikazan je opšti algoritam *Gradijentni uspon* (ili *najstrmiji uspon*), zasnovan na opisanoj ideji i najjednostavniji među metodama lokalne optimizacije za diferencijabilne funkcije.

**Algoritam:** Gradijentni uspon

**Ulaz:** Diferencijabilna funkcija  $f(\mathbf{x})$ , polazna tačka  $\mathbf{a}_0$  i preciznost  $\varepsilon$

**Izlaz:** Aproksimacija maksimuma funkcije  $f$

- 1: postavi  $n$  na 0;
- 2: **ponavljam**
- 3: izračunaj vrednost  $\nabla f(\mathbf{a}_n)$ ;
- 4: izvrši kretanje u smeru gradijenta do sledeće tačke  $\mathbf{a}_{n+1}$ ;
- 5: uvećaj  $n$  za 1;
- 6: **dok nije ispunjen** uslov  $|f(\mathbf{a}_n) - f(\mathbf{a}_{n-1})| \leq \varepsilon |f(\mathbf{a}_{n-1})|$
- 7: vrati  $\mathbf{a}_n$  kao rešenje.

Slika 4.4: Algoritam Gradijentni uspon.

Navedeni opšti algoritam potrebno je precizirati. Poznavanje gradijenta i proizvoljno kretanje u njegovom smeru ne garantuje nalaženje maksimuma, jer je u zavisnosti od dužine koraka moguće preći preko maksimuma, nastaviti dalje i doći do rešenja goreg od tekućeg. Stoga je u svakom koraku potrebno imati pogodnu vrednost

$\lambda_n$  koja se koristi u izboru nove tačke (tj. koja određuje koliko daleko će se vršiti kretanje u smeru gradijenta):

$$\mathbf{a}_{n+1} = \mathbf{a}_n + \lambda_n \nabla f(\mathbf{a}_n)$$

Vrednosti  $\lambda_n$  mogu se definisati na različite načine. Dovoljan uslov za konvergenciju dat je Robins-Monroovim uslovima koji kažu da se za vrednosti  $\lambda_n$  mogu uzeti bilo koji brojevi koji zadovoljavaju sledeće uslove:

$$\sum_{n=0}^{\infty} \lambda_n = \infty \quad \sum_{n=0}^{\infty} \lambda_n^2 < \infty$$

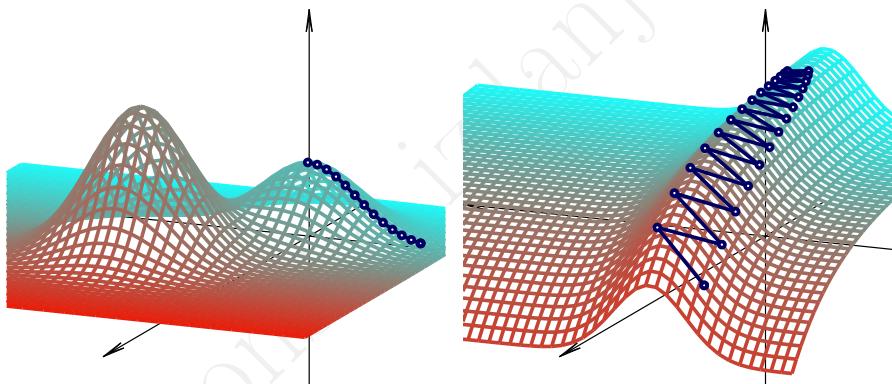
Intuitivno, prvi uslov garantuje da su koraci pretrage dovoljno veliki da pretraga ne uspori prerano i da stoga uopšte ne stigne do maksimuma, dok drugi garantuje da su koraci dovoljno mali da optimizacioni proces ne divergira. Jedan izbor koji zadovoljava ove uslove je  $\lambda_n = \frac{1}{n+1}$ .

Prikazani algoritam ne garantuje pronalaženje globalnog maksimuma. Naime, rešenje koje algoritam daje može da zavisi od izabrane polazne tačke i može se desiti da ono bude samo lokalni (a ne i globalni) maksimum.

Algoritam **Gradijentni uspon** veoma je jednostavan i često dovoljno uspešan, ali ima i sledeće slabosti (slično kao i penjanje uzbrdo):

**Opasnost od lokalnih maksimuma:** Algoritam **Gradijentni uspon** može doći do lokalnog maksimuma i tada će ga vratiti kao rezultat. Algoritam nema načina da utvrdi da li je maksimum u kojem se nalaze globalni ili samo lokalni (slika 4.5, levo).

**Opasnost od platoa:** Platoi predstavljaju oblasti u kojima funkcija cilja ima konstantnu vrednost. Zbog toga je gradijent jednak nuli, te je nemoguće odrediti vrednost za sledeću iteraciju.



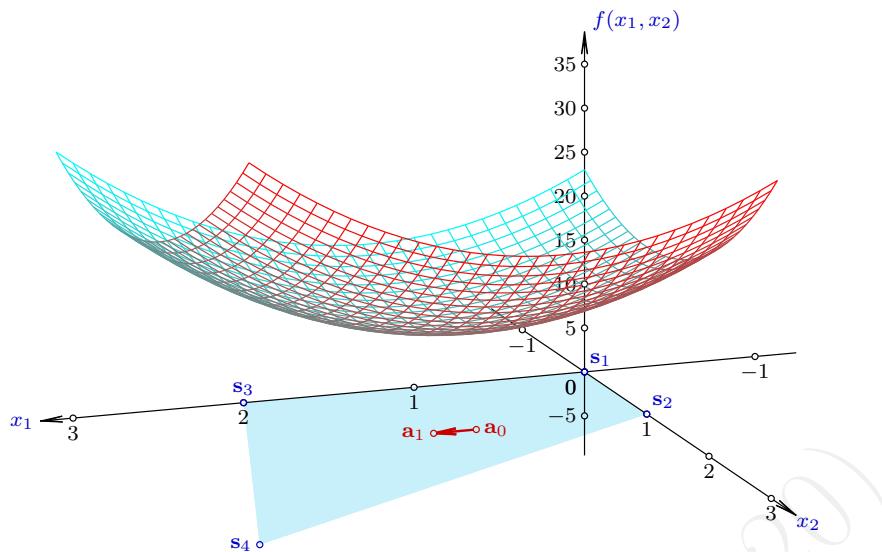
Slika 4.5: Situacija u kojoj **Gradijentni uspon** može da vrati samo lokalni maksimum (levo) i situacija sa „grebenom“ u kojoj **Gradijentni uspon** otežano dolazi do rešenja (desno).

**Neefikasnost u slučaju grebena:** Grebeni predstavljaju uske staze koje opadaju ili rastu duž nekog pravca (slika 4.5, desno). U takvim situacijama, kretanje ka većim vrednostima može biti usporeno. Naime, moguće je da se pravac najbržeg rasta funkcije preskače (zbog neodgovarajuće dužine kretanja u smeru gradijenta) i umesto bolje usmerenog rasta (duž grebena) primenjuje se veliki broj cik-cak koraka.

**Spora konvergencija:** Konvergencija algoritma često je spora. Brže alternative su ili komplikovanije ili imaju dodatne prepostavke o svojstvima ciljne funkcije (poput jačih varijanti konveksnosti) ili zahtevaju dodatne informacije o ciljnoj funkciji (poput parcijalnih izvoda drugog reda).

U problemima matematičke optimizacije često se traži i minimum date funkcije. Za ove probleme može se koristiti opšti algoritam **Gradijentni spust** (ili *najstrmiji spust*). Jedina razlika u odnosu na algoritam **Gradijentni uspon** je u tome što se kretanje ne vrši u smeru gradijenta, već u suprotnom smeru, tj. nova tačka bira se na sledeći način:  $\mathbf{a}_{n+1} = \mathbf{a}_n - \lambda_n \nabla f(\mathbf{a}_n)$ . Delovanje ovog algoritma može se intuitivno ilustrovati sledećim primerom: ukoliko je površina zemlje aproksimirana glatkom površinom, onda bi se primenom algoritma **Gradijentni spust** od tačke izvora neke reke stiglo do njenog ušća.

**Primer 4.3.** Potrebno je izgraditi medicinsku stanicu koja bi opsluživala četiri sportske lokacije. Stanica bi trebalo da bude relativno blizu svim lokacijama. Jeden povoljan izbor njene lokacije bi bila tačka  $\mathbf{x}$  takva



Slika 4.6: Ilustracija problema pronalaženja optimalne pozicije za medicinsku stanicu.

da je zbir

$$f(\mathbf{x}) = \sum_{i=1}^4 \|\mathbf{x} - \mathbf{s}_i\|^2$$

minimalan, gde važi  $\mathbf{x} = (x_1, x_2)$ ,  $\mathbf{s}_1 = (0, 0)$ ,  $\mathbf{s}_2 = (0, 1)$ ,  $\mathbf{s}_3 = (2, 0)$  i  $\mathbf{s}_4 = (3, 3)$ .

Zapisano drugačije, funkcija  $f$  jednaka je (slika 4.6):

$$\begin{aligned} f(\mathbf{x}) &= (x_1 - 0)^2 + (x_2 - 0)^2 + (x_1 - 0)^2 + (x_2 - 1)^2 + \\ &\quad (x_1 - 2)^2 + (x_2 - 0)^2 + (x_1 - 3)^2 + (x_2 - 3)^2 \\ &= 4x_1^2 + 4x_2^2 - 10x_1 - 8x_2 + 23 \end{aligned}$$

Gradijent funkcije  $f$  jednak je

$$\nabla f(\mathbf{x}) = (8x_1 - 10, 8x_2 - 8)$$

Neka je polazna tačka  $\mathbf{a}_0 = (1, 1)$  i  $\varepsilon = 10^{-6}$ . Vrednost gradijenta  $\nabla f(\mathbf{a}_0)$  u prvoj iteraciji je  $(-2, 0)$ . Vrednost parametra  $\lambda_n$  je  $\frac{1}{n+1}$ . Sledеća tabela prikazuje kako se menjaju relevantne vrednosti prilikom primene gradijentnog spusta:

$n$	$\mathbf{a}_n$	$\lambda_n$	$\nabla f(\mathbf{a}_n)$
0	(1, 1)	1	(-2, 0)
1	(3, 1)	1/2	(14, 0)
2	(-4, 1)	1/3	(-42, 0)
3	(10, 1)	1/4	(70, 0)
4	(-7.5, 1)	1/5	(-70, 0)
5	(6.5, 1)	1/6	(42, 0)
6	(-0.5, 1)	1/7	(-14, 0)
7	(1.5, 1)	1/8	(2, 0)
8	(1.25, 1)	1/9	(0, 0)
9	(1.25, 1)	-	-

Primetno je da na početku algoritam pravi velike korake, što je posledica relativno velike vrednosti  $\lambda_n$ . Međutim, kako se ona smanjuje, koraci postaju manji i dolazi se do tačnog rešenja. U opštem slučaju, retko se dešava zaustavljanje sa tačnim rešenjem. U ovom konkretnom slučaju, rešenje je moglo da bude pronađeno i analitički – rešavanjem jednačina  $\nabla f(\mathbf{x}) = 0$ , ali to u opštem slučaju nije moguće.

## 4.2 Pretraga Prvo najbolji

Pristup pretrage *prvo najbolji* (eng. *best-first search*) predstavlja osnovu za različite algoritme pretrage grafa (pri čemu je u vidu grafa opisan prostor stanja i akcija za neki problem). Rešenjem se smatra niz čvorova (tj. put) od polaznog do ciljnog čvora u grafu. U toku primene algoritma, svakom čvoru stabla pretrage pridružuje se informacija o njegovom prethodniku (roditelju) u mogućem rešenju, isto kao u Dejkstrinom algoritmu.

Da bi se izbegle beskonačne petlje (tj. beskonačno obrađivanje istog stanja, tj. beskonačni nizovi čvorova stabla pretrage u kojima je isto stanje), održavaju se dve liste čvorova:

- *zatvorena lista* (ili *lista zatvorenih stanja*) – lista stanja za koje su već ispitani svi susedi (tj. sva neposredno dostupna stanja);
- *otvorena lista* (ili *lista otvorenih stanja*) – lista stanja koja su već posećena, ali nisu obrađeni svi njihovi susedi.

Implementacija otvorene liste treba da omogućava efikasan pristup čvoru sa najboljom ocenom  $f(n)$ . Jednostavnosti radi, u nastavku će se često isto označavati čvor stabla pretrage i stanje koje mu je pridruženo.

Na početku je u otvorenoj listi samo polazno stanje, a zatvorena lista je prazna. Suštinska ideja je da se u svakoj iteraciji, analizira element otvorene liste sa najboljom ocenom i obrađuju se iz njega neposredno dostupna stanja. Ukoliko se nađe na ciljno stanje – zadatak je rešen i algoritam završava rad. Precizniji opis algoritma dat je na slici 4.7.

### Algoritam: Prvo najbolji

**Ulaz:** Graf  $G$ , polazni čvor i ciljni čvor

**Izlaz:** Niz koraka od polaznog do ciljnog čvora (ako postoji put između ova dva stanja)

- 1: zatvorenu listu postavi na praznu listu, u otvorenu listu stavi samo polazni čvor (sa izračunatom vrednošću funkcije  $f$ );
- 2: **dok god** ima elemenata u otvorenoj listi **radi**
- 3:     izaberi čvor  $n$  (*tekući čvor*) iz otvorene liste koji ima najbolju ocenu  $f(n)$ ;
- 4:     **ako** je  $n$  ciljni čvor **onda**
- 5:         izvesti o uspehu i vrati rešenje konstruišući put od polaznog do ciljnog čvora (idući unazad — od ciljnog čvora);
- 6:     **za** svaki čvor  $m$  koji je direktno dostupan iz  $n$  **radi**
- 7:         **ako**  $m$  nije ni u otvorenoj ni u zatvorenoj listi **onda**
- 8:             dodaj ga u otvorenu listu i označi  $n$  kao njegovog roditelja;
- 9:         izbacи  $n$  iz otvorene liste i dodaj ga u zatvorenu listu;
- 10: izvesti da traženi put ne postoji (otvorena lista je prazna i uspeh nije prijavljen).

Slika 4.7: Algoritam Prvo najbolji.

Algoritam Prvo najbolji ne pretende da daje optimalno rešenje (tj. da otkrije najbolji put do ciljnog čvora) niti pruža ikakve garancije u tom smislu. Ipak, da bi se uvećale šanse da se pronađe što kraći put između dva čvora, kada se analizira čvor  $m$  koji je direktno dostupan iz tekućeg čvora  $n$  (u skladu sa prikazanim algoritmom), u slučaju da je čvor  $m$  već u otvorenoj ili zatvorenoj listi, može se proveriti da li je put od polaznog čvora do čvora  $m$  preko čvora  $n$  bolji od postojećeg puta do  $m$ ; ako jeste, treba promeniti informaciju o roditelju čvora  $m$  na čvor  $n$ , a ako je  $m$  bio u zatvorenoj listi, prebaciti ga u otvorenu.

Ako je broj stanja i akcija konačan, algoritam se očigledno zaustavlja i ima svojstvo potpunosti, o čemu govori naredna teorema.

**Teorema 4.1.** *Ako je broj stanja i akcija konačan, algoritam Prvo najbolji se zaustavlja i nalazi traženi put ako on postoji.*

Ako funkcija  $f(n)$  vraća dubinu čvora  $n$  u BFS obilasku grafa počev od polaznog čvora, onda se navedeni algoritam ponaša kao algoritam obilaska u širinu (zapravo – ponaša se slično, možda ne identično, jer čvorovi na istoj dubini možda neće biti posećeni u istom poretku). Ako funkcija  $f(n)$  vraća zbir cena od polaznog čvora do čvora  $n$ , onda se navedeni algoritam ponaša kao Dejkstrin algoritam.

Opšti algoritam Prvo najbolji predstavlja bitnu modifikaciju algoritma jednostavnog pohlepnog pristupa. Iako oba u jednom čvoru koriste slično navođenje i biraju (najpre) najbolji susedni čvor (tj. čvor  $n$  sa najboljom vrednošću  $f(n)$ ), algoritam Prvo najbolji, za razliku od jednostavnog pohlepnog pristupa, omogućava vraćanje na čvorove koji nisu bili ispitani (jer je neka od alternativa obećavala više). Ovim pristupom, zahvaljujući alternativama u otvorenoj listi, omogućava se uspešan nastavak pretrage i u slučajevima kada bi pohlepna pretraga naišla na plato ili na lokalni optimum. Dodatno, ne postoji mogućnost beskonačnih petlji (zahvaljujući pamćenju obrađenih čvorova u zatvorenoj listi).

**Primer 4.4.** U slučaju primera slagalice diskutovanog u primeru 4.2 (slika 4.3), situacija prikazana levo predstavlja lokalni minimum, zbog čega se pohlepna pretraga zaustavlja. Algoritam Prvo najbolji će odabrat jedan od mogućih poteza, ali će alternativno stanje čuvati u otvorenoj listi i možda ga obraditi kasnije. U situaciji prikazanoj desno, pohlepnom pretragom se prazno polje spušta do donjeg desnog ugla, čime se dolazi do lokalnog optimuma i pohlepna pretraga ne može da nastavi. Međutim, u slučaju algoritma Prvo najbolji, stanja koja su bila alternative ispitanim stanjima su i dalje u otvorenoj listi i ispituju se dalje. Stoga je algoritam Prvo najbolji u stanju da reši slagalicu, ali ne garantuje nalaženje rešenja koje se sastoji od najmanjeg broja poteza.

### 4.3 Algoritam A\*

Algoritam A\* pretraga ili, kraće, algoritam A\* (čita se „a zvezda“, eng. „a star“) za određivanje najkraćeg puta između dva čvora grafa, jedan je od fundamentalnih i najpopularnijih algoritama veštačke inteligencije. Zasnovan je na korišćenju heuristika za usmeravanje pretrage, ali ipak ima svojstva kao što su potpunost i optimalnost. Prvu verziju algoritma A\* razvili su Hart (Peter Hart), Nilson (Nils Nilson) i Rafael (Bertram Raphael) 1968. godine, a u narednim godinama uvedeno je nekoliko modifikacija.

Algoritam A\* je varijanta algoritma Prvo najbolji u kojoj se koristi funkcija evaluacije  $f$  koja ima sledeću specifičnu formu:

$$f(n) = g(n) + h(n),$$

gde je  $g(n)$  cena puta od polaznog čvora do čvora  $n$ , a  $h(n)$  je procenjena (heuristička) cena najjeftinijeg puta od čvora  $n$  do ciljnog čvora. Dok se traga za najkraćim putem, uvek se zna tekuća minimalna cena (a može se menjati tokom primene algoritma) od polaznog čvora do čvora  $n$  (tj. tekuća vrednost  $g(n)$ ), ali se vrednost  $h(n)$  može samo procenjivati. Od kvaliteta heuristike u velikoj meri zavisi ponašanje i efikasnost algoritma. Izbor kvalitetne heuristike jedan je od najvažnijih i najtežih izazova u dizajniranju konkretnih implementacija algoritma A\*. Ciljni čvor  $t$  može se prepoznati i za njega je vrednost heuristike jednaka 0, ali za sve druge čvorove ne postoji opšti pristup koji daje kvalitetnu heuristiku.

Algoritam A\* traži optimalno rešenje (tj. otkriva najbolji put do ciljnog čvora) i zato za svaki čvor na koji nađe proverava da li je do njega ranije već bio pronađen neki lošiji put i, ako jeste, zamenjuje ga novim, boljim putem. Takva provera je u algoritmu Prvo najbolji bila opcionala, a u algoritmu A\* je obavezna.

Pored toga što je specijalan slučaj metoda Prvo najbolji, algoritam A\* je uopštenje Dejkstrinog algoritma. Kao i u Dejkstrinom algoritmu, čvorovi koje tek treba obraditi čuvaju se u listi, sortiranoj prema nekom kriterijumu. Algoritam A\* često ispituje manje čvorova nego Dejkstrin algoritam. To smanjenje proističe iz korišćenja heuristike koja procenjuje rastojanje do ciljnog čvora. Ključna razlika između dva algoritma je u tome što Dejkstrin algoritam (kao algoritam neinformisane pretrage) uzima u obzir samo cenu od polaznog do tekućeg čvora — vrednost  $g(n)$ , a A\* (kao algoritam informisane pretrage) koristi vrednost funkcije evaluacije  $f(n) = g(n) + h(n)$ .

Opis algoritma A\* dat je na slici 4.8. Prilikom dodavanja čvora  $m$  u otvorenu listu, vrednost  $g(m)$  se može izračunati na inkrementalan i efikasan način: vrednost  $g(m)$  jednaka je zbiru vrednosti funkcije  $g$  za roditelja čvora  $m$  i ceni puta od roditelja do  $m$ .

Ako algoritam nađe na čvor  $m$  koji je već u otvorenoj ili zatvorenoj listi, to znači da je pronađen novi put do čvora  $m$ . Tada se proverava da li je put od polaznog čvora do već posećenog čvora  $m$  preko čvora  $n$  bolji od postojećeg puta. Ako jeste bolji, potrebno je ažurirati vrednost  $g(m)$ . To može da se desi i za čvor  $m$  koji pripada zatvorenoj listi: ako to jeste slučaj, potrebno je čvor  $m$  ponovo ispitati kao otvoreni čvor. Ovo je neophodno kako bi se obezbedilo pronalaženje najboljeg puta od polaznog do ciljnog čvora.

**Primer 4.5.** U primeru pronalaženja najkraćih puteva između gradova (primer 2.2), ako su poznata rastojanja između gradova vazdušnim putem, algoritam A\* može kao heurističku funkciju  $h(n)$  da koristi vazdušno rastojanje od čvora  $n$  do ciljnog čvora, kao i u primeru 4.1. Naredna tabela ilustruje izvršavanje

**Algoritam: A\***

**Ulaz:** Graf  $G$ , polazni čvor i ciljni čvor

**Izlaz:** Najkraći put od polaznog do ciljnog čvora (ako postoji put između ova dva čvora)

- 1: zatvorenu listu postavi na praznu listu, u otvorenu listu stavi samo polazni čvor (sa izračunatom vrednošću funkcije  $f$ );
- 2: **dok god** ima elemenata u otvorenoj listi **radi**
- 3: izaberi čvor  $n$  (*tekući čvor*) iz otvorene liste koji ima najbolju ocenu  $f(n)$ ;
- 4: **ako** je  $n$  ciljni čvor **onda**
- 5: izvesti o uspehu i vrati rešenje konstruišući put od polaznog do ciljnog čvora (idući unazad — od ciljnog čvora);
- 6: **za** svaki čvor  $m$  koji je direktno dostupan iz  $n$  **radi**
- 7: **ako**  $m$  nije ni u otvorenoj ni u zatvorenoj listi **onda**
- 8: dodaj ga u otvorenu listu i označi  $n$  kao njegovog roditelja. Izračunaj i pridruži vrednosti  $g(m)$  i  $f(m)$  čvoru  $m$ ;
- 9: **inace**
- 10: **ako** je put od polaznog čvora do čvora  $m$  preko čvora  $n$  bolji (kraći ili jeftiniji) od postojećeg puta do  $m$  (trenutna vrednost  $g(m)$ ) **onda**
- 11: promeni informaciju o roditelju čvora  $m$  na čvor  $n$  i ažuriraj vrednosti  $g(m)$  i  $f(m)$  i
- 12: **ako** je čvor  $m$  bio u zatvorenoj listi **onda**
- 13: prebac ga u otvorenu;
- 14: izbaci  $n$  iz otvorene liste i dodaj ga u zatvorenu listu;
- 15: izvesti da traženi put ne postoji (otvorena lista je prazna i uspeh nije prijavljen).

Slika 4.8: Algoritam A\*.

algoritma A\* na primeru puta Podgorica-Budimpešta.

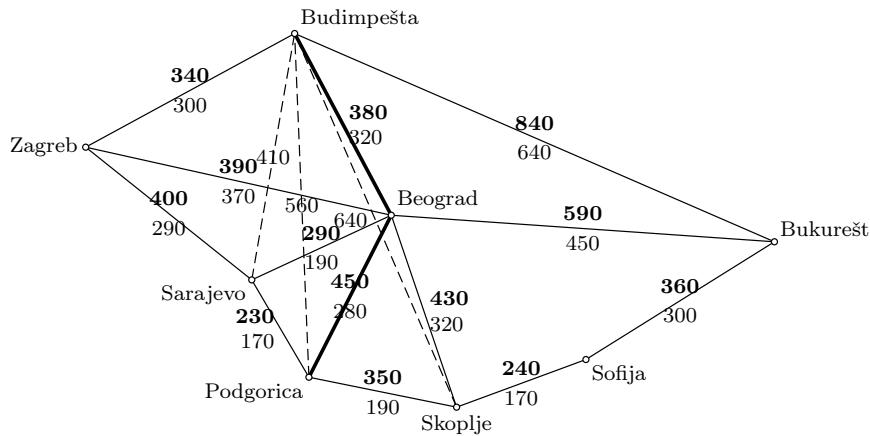
tekući čvor	stanje otvorene liste [čvor(roditelj, g+h)]	u zatvorenu listu se dodaje
$Pg$	$Pg(-, 0+560)$	
$Sa$	$Sa(Pg, 230+410), Bg(Pg, 450+320), Sk(Pg, 350+640)$	$Pg(-)$
$Bg$	$Bg(Pg, 450+320), Sk(Pg, 350+640), Zg(Sa, 630+300)$	$Sa(Pg)$
$Bud$	$Sk(Pg, 350+640), Zg(Sa, 630+300), Bud(Bg, 830+0), Buk(Bg, 1040+640)$	$Bg(Pg)$

Pronađeni put je Podgorica-Beograd-Budimpešta, za razliku od pohlepne pretrage koja pronalazi put Podgorica-Sarajevo-Beograd-Budimpešta (slika 4.9).

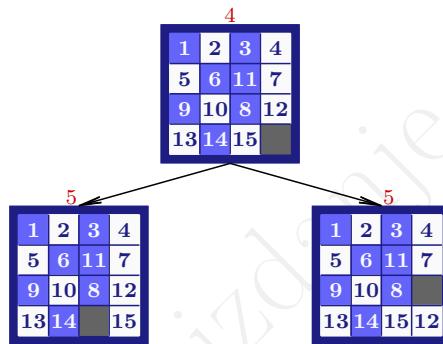
**Primer 4.6.** U slučaju Lojdove slagalice, kao u primeru 4.2, za heurstiku se koristi zbir Menhetn rastojanja svakog od 15 polja slagalice do njegovog ciljnog mesta. Slika 4.10 prikazuje stanje slagalice i dva moguća naslednika, pri čemu oba imaju ocenu veću od ocene polaznog stanja. Stoga, kako se polazno stanje nalazi u lokalnom minimumu, pristup čiste pohlepne pretrage nemoćan je već na početku.

Za isto polazno stanje, algoritam A\* pronalazi rešenje od šest poteza – gore, levo, gore, desno, dole, dole. Stablo pretrage vršene algoritmom A\* prikazano je na slici 4.11.

Korišćenje algoritma A\* nije uvek jednostavno. Često je algoritam potrebno prilagoditi specifičnom problemu a uvek je, u kontekstu aplikacija koje rade u realnom vremenu, važno imati u vidu vremensku složenost, prostornu složenost, upravljanje memorijom i različite dodatne faktore. Neki od dodatnih, specifičnih zahteva mogu da iziskuju dodatno matematičko znanje i izračunavanja i specifične implementacione tehnike i strukture. Svi ti moduli treba da budu uklopljeni u kompaktan i efikasan sistem za nalaženje puta.



Slika 4.9: Graf koji opisuje problem puteva između gradova. Na grafu podebljanim ciframa ispisana su kopnena, a običnim ciframa vazdušna rastojanja između dva grada.



Slika 4.10: Stablo pohlepne pretrage na primeru slagalice u kojem se polazno stanje nalazi u lokalnom minimumu.

### 4.3.1 Svojstva algoritma A\*

Može se dokazati da je algoritam A\* potpun i da je pod određenim uslovima optimalan:

**Potpunost:** Ako su broj čvorova i broj akcija konačni, ako postoji put između dva čvora, algoritam A\* će, kao i svaki Prvo najbolji algoritam, naći jedan takav (ukoliko je raspoloživo dovoljno vremena i memorijskog prostora). Čak i ako je heuristička funkcija veoma loša, ciljni čvor će biti dostignut u konačnom broju koraka.

**Optimalnost:** Od svih puteva između dva data čvora, algoritam A\* vratiće najkraći (tj. vratiće optimalno rešenje) ako je funkcija  $h$  dopustiva (eng. *admissible*). Funkcija  $h$  je dopustiva ako nikada ne precenjuje stvarno rastojanje između tekućeg čvora i ciljnog čvora, tj. ako za svaki čvor važi:

$$h(n) \leq h^*(n),$$

gde je  $h^*(n)$  cena najkraćeg puta od čvora  $n$  do ciljnog čvora (tj.  $h^*$  je idealna, optimalna heuristika).

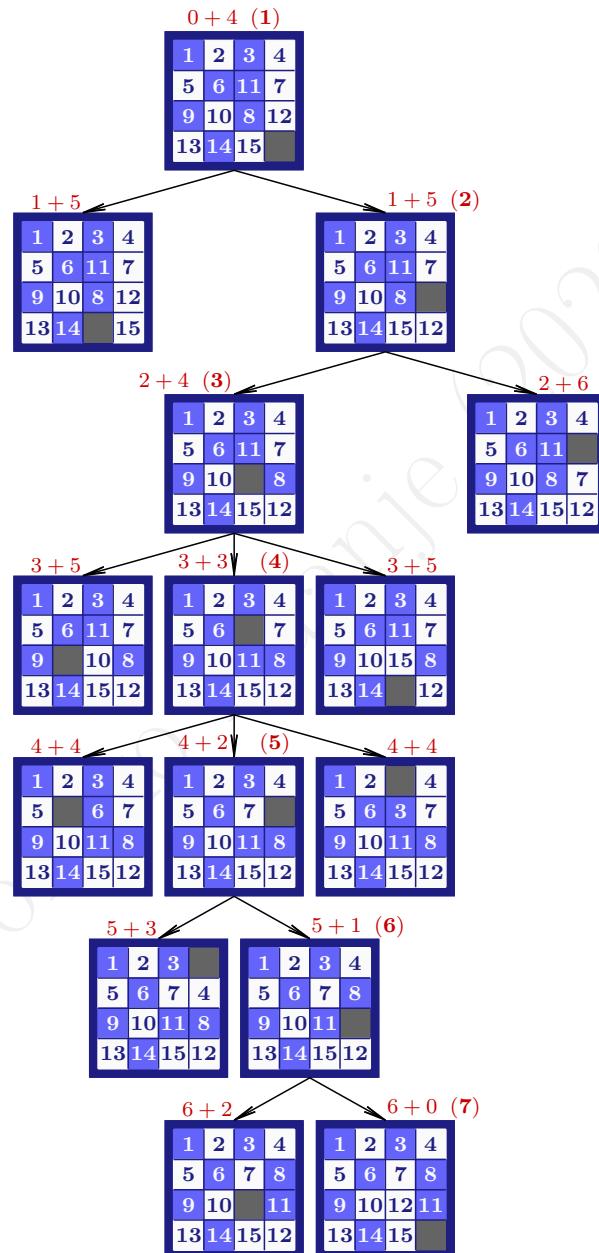
Ukoliko funkcija  $h$  nije dopustiva, ali ne precenjuje stvarnu cenu za više od  $d$ , onda je cena puta koji će pronaći algoritam A\* viša od cene najkraćeg za ne više od  $d$ .

Funkcija  $h$  je *konzistentna* (eng. *consistent*) ako ima vrednost 0 za ciljni čvor i za bilo koja dva susedna čvora  $n$  i  $m$  važi:

$$c(n, m) + h(m) \geq h(n)$$

gde je  $c(n, m)$  cena pridružena (moguće usmerenoj) grani  $(n, m)$ . Ako je funkcija  $h$  konzistentna, onda je ona i dopustiva (kao što tvrdi naredna teorema). Obratno ne važi nužno: funkcija  $h$  može da bude dopustiva, a da ne bude konzistentna.

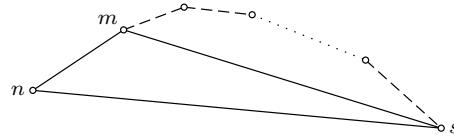
**Teorema 4.2.** Ako je  $h$  konzistentna heuristika, onda je ona i dopustiva.



Slika 4.11: Stablo pretrage vršene algoritmom A\*, na primeru slagalice u kojem se polazno stanje nalazi u lokalnom minimumu. U prikazu stabla, među naslednicima svakog stanja su samo stanja koja nisu već dodata u zatvorenu listu. U zagradama su navedeni redni brojevi pod kojim stanja postaju tekuća (poredak obilaska potomaka mogao je da bude i drugačiji).

**Dokaz:** Neka je  $h^*(n)$  jednakoj najkraćem rastojanju od čvora  $n$  do ciljnog čvora  $s$  (tj. neka je  $h^*$  optimalna heuristika). Dokažimo da, ako je  $h$  konzistentna heuristika, onda za svaki čvor  $n$  važi  $h(n) \leq h^*(n)$ . Dokaz izvedimo po broju čvorova između  $n$  i ciljnog čvora  $s$  na najkraćem putu između njih.

Ako između  $n$  i  $s$  na najkraćem putu nema čvorova, onda iz uslova konzistentnosti važi  $c(n, s) + h(s) \geq h(n)$ , pa kako je  $h^*(n) = c(n, s)$  i  $h(s) = 0$ , važi  $h^*(n) \geq h(n)$ .



Pretpostavimo da tvrđenje važi za svaki čvor za koji je broj čvorova do ciljnog čvora na najkraćem putu manji od  $k$ , za  $k > 0$ . Ako između  $n$  i  $s$  na najkraćem putu ima  $k$  čvorova, gde je  $k > 0$ , neka je  $m$  prvi čvor na koji se nađe posle čvora  $n$  na najkraćem putu do  $s$ . Od čvora  $m$  do  $s$  na najkraćem putu ima  $k - 1$  čvorova, pa na osnovu induktivne hipoteze ( $h^*(m) \geq h(m)$ ) i na osnovu svojstva konzistentnosti važi

$$h^*(n) = c(n, m) + h^*(m) \geq c(n, m) + h(m) \geq h(n),$$

što je i trebalo dokazati.  $\square$

Ako je funkcija  $h$  konzistentna, nije potrebno proveravati da li je put preko tekućeg čvora do jednom zatvorenog čvora bolji od postojećeg (jer sigurno nije). Dakle, ako je funkcija  $h$  konzistentna, algoritam A\* je optimalan i još jednostavniji nego u opštem slučaju. Dokaz optimalnosti dat je u nastavku i zasniva se na nekoliko pomoćnih tvrđenja. U tvrđenjima se pominju vrednosti  $f$  i  $g$  i treba imati na umu sledeće: vrednosti funkcije  $g$ , pa onda i  $f$  za neko stanje mogu se menjati u toku primene algoritma, ali u stablu pretrage svaki čvor ima vrednosti  $g$  i  $f$  koje se ne menjaju. Može da bude više čvorova stabla pretrage kojima je pridruženo jedno isto stanje, tj. isti čvor grafa.

**Lema 4.1.** *Ako je  $h$  konzistentna heuristika, onda u svakom trenutku primene algoritma, duž svakog puta kroz stablo pretrage, vrednosti  $f(n)$  su neopadajuće.*

**Dokaz:** Ako je u nekom trenutku primene algoritma čvor stabla pretrage  $m$  tekući i ako je njegov roditelj čvor  $n$ , onda važi:

$$f(m) = g(m) + h(m) = g(n) + c(n, m) + h(m) \geq g(n) + h(n) = f(n)$$

Tvrđenje leme onda sledi na osnovu jednostavnog induktivnog argumenta.  $\square$

**Lema 4.2.** *Ako je  $h$  konzistentna heuristika, za niz čvorova  $n$  redom proglašenih za tekuće, niz vrednosti  $f(n)$  čini neopadajući niz.*

**Dokaz:** U svakoj iteraciji, algoritam bira za tekući čvor iz otvorene liste sa najmanjom vrednošću  $f(n)$  (te svi preostali čvorovi u skupu otvorenih čvorova imaju veće ili jednake vrednosti  $f$ ). Svi budući tekući čvorovi su preostali čvorovi iz otvorene liste, ili njihovi potomci. Na osnovu prethodne leme onda sledi da svi budući čvorovi imaju vrednosti  $f$  veće ili jednake  $f(n)$ . Kako ovo važi za svaki tekući čvor  $n$ , sledi tvrđenje leme, tj. algoritam proglašava čvorove tekućim u neopadajućem poretku po  $f(n)$ .  $\square$

**Lema 4.3.** *Ako je  $h$  konzistentna heuristika, kad neki čvor stabla pretrage  $n$  postane tekući, do njegovog stanja je već pronađen optimalan put. Drugim rečima, svaki čvor koji postaje tekući biće čvor sa najmanjom cenom za to stanje.*

**Dokaz:** Kada algoritam proglaši neki čvor tekućim, pri čemu je to prvi takav čvor za odgovarajuće stanje  $n$ , on ima neke vrednosti  $g(n) = g_0$  i  $f(n) = f_0$ . Pretpostavimo da  $g(n)$  nije optimalan put i pretpostavimo da je optimalan put do istog stanja moguće dostići u nekoj kasnijoj iteraciji, u nekom budućem čvoru koji ima vrednosti  $g_1$  i  $f_1$ . Kako je  $g_1$  cena optimalnog puta do  $n$ , važi  $g_0 > g_1$ , pa i  $g_0 + h(n) > g_1 + h(n)$ , tj.  $f_0 > f_1$ . S druge strane, na osnovu prethodne leme, važi  $f_0 \leq f_1$ , što je kontradikcija.  $\square$

**Teorema 4.3.** Ako je  $h$  konzistentna heuristika, ako je pronađen put do ciljnog čvora, on je sigurno optimalan.

**Dokaz:** Algoritam vraća nađeni put čim ciljni čvor po prvi put postane tekući. Na osnovu prethodne leme, ako je  $h$  konzistentna heuristika, kad ciljni čvor postane tekući, do njega je već pronađen optimalan put, što daje tvrđenje teoreme.  $\square$

Upravo lema 4.3 govori da za čvorove dostupne iz tekućeg čvora koji su već zatvoreni, ne mora da se proverava da li njihova vrednost  $g$  treba da bude ažurirana. Ovo tvrđenje u slučaju konzistentne heuristike obezbeđuje jednostavniju i efikasniju implementaciju algoritma A\*.

**Složenost:** Složenost algoritma A\* (i vremenska i prostorna) bitno zavisi od heuristike. Ako je heuristika jednak nuli, i broj otvorenih čvorova, i prostorna i vremenska složenost algoritma A\* jednake su kao za Dejkstrin algoritam. Često je važno i razmatranje složenosti algoritma A\* u zavisnosti od dužine najkraćeg puta. Ukoliko svaki čvor ima  $b$  susednih čvorova, a najkraći put od nekog čvora do ciljnog je dužine  $d$ , onda je složenost u najgorem slučaju jednak  $O(b^d)$ , tj. broj obrađenih čvorova eksponencijalno zavisi od  $d$ . Može se dokazati da broj obrađenih čvorova polinomski zavisi od  $d$  ako heuristika  $h$  zadovoljava sledeći uslov:

$$|h(x) - h^*(x)| \leq O(\log h^*(x))$$

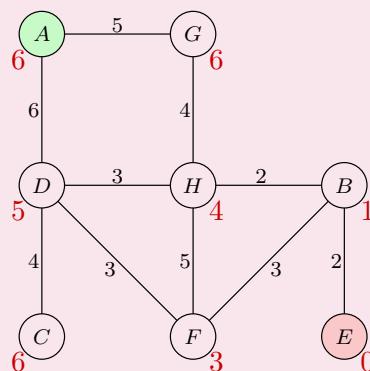
gde je  $h^*$  idealna heuristika, tj. funkcija koja vraća cenu najkraćeg puta od čvora  $x$  do ciljnog čvora.

Ukoliko je  $c$  najkraće rastojanje od polaznog čvora  $n$  do ciljnog čvora i ukoliko se koristi dopustiva heuristika, može se dokazati da će algoritam A\* otvoriti sve čvorove za koje važi  $f(m) < c$ , kao i neke čvorove  $m$  za koje važi  $f(m) = c$ .

Za algoritam A\* se kaže i da je *optimalno efektivan*, jer je dokazano da ne postoji algoritam koji je potpun a nikad ne otvara više čvorova od algoritma A\*.

Prethodna tvrđenja sugerisu da algoritam A\* najbolje performanse (najmanji broj obrađenih čvorova) daje kada je funkcija heuristike bliska idealnoj funkciji heuristike. S druge strane, optimalnost je garantovana samo ako funkcija heuristike nikada ne precenjuje stvarnu cenu puta. Zajedno, to govori da dobra funkcija heuristike mora da bude veoma pažljivo konstruisana, tako da bude što bliža idealnoj funkciji, ali da je nikada ne premašuje.

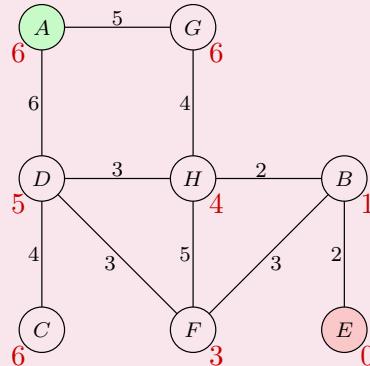
**Primer 4.7.** U sledećem grafu zadatak je naći najkraći put od  $A$  do  $E$ . Pored čvorova grafa zapisane su procenjene dužine puta do čvora  $E$ , tj. vrednosti funkcije  $h$ . Naredna tabela ilustruje primenu algoritma A\* (odgovarajuće stablo pretrage prikazano je na slici 4.12).



tekući čvor	stanje otvorene liste [čvor(roditelj, $g+h$ )]	u zatvorenu listu se dodaje
	$A(-, 0+6)$	
$A(6)$	$D(A, 6+5), G(A, 5+6)$	$A(-)$
$D(11)$	$G(A, 5+6), F(D, 9+3), H(D, 9+4), C(D, 10+6)$	$D(A)$
$G(11)$	$F(D, 9+3), H(D, 9+4), C(D, 10+6)$	$G(A)$
$F(12)$	$B(F, 12+1), H(D, 9+4), C(D, 10+6)$	$F(D)$
$B(13)$	$H(D, 9+4), E(B, 14+0), C(D, 10+6)$	$B(F)$
$H(13)$	$B(H, 11+1), E(B, 14+0), C(D, 10+6)$	$H(D)$
$B(12)$	$E(B, 13+0), C(D, 10+6)$	$B(H)$
$E(13)$	$C(D, 10+6)$	

Na kraju primene algoritma, kada je čvor  $E$  postao tekući čvor, konstruiše se traženi put – koristeći informacije o roditeljima za čvorove iz zatvorene liste:  $A - D - H - B - E$ . Korišćena heurstika je dopustiva, pa je pronađeni put optimalan.

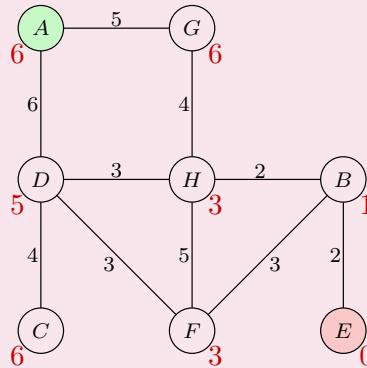
Za neke heurstike (kao što je, na primer, euklidsko rastojanje) postoji jednostavan, opšti argument da je dopustiva i konzistentna, a u nekim situacijama, kao što je i graf u ovom primeru, dopustivost i konzistentnost mogu da se ispitaju samo neposrednim proveravanjem za sve čvorove. Upotrebljena heurstika  $h$  nije konzistentna jer važi  $c(H, B) + h(B) = 2 + 1 < 4 = h(H)$  (primetimo da vrednosti  $f$  za čvorove koji postaju tekući nisu neopadajuće). Zato je nužno i za zatvorene čvorove proveravati da li se put do njih može popraviti. To i jeste bio slučaj za čvor  $B$ : u koraku u kojem se  $H$  briše iz zatvorene liste, u nju se dodaje čvor  $B$  jer je do njega pronađen bolji put (preko  $H$ ) od ranije postojećeg. Ukoliko to ne bi bilo rađeno, algoritam bi se ponašao na sledeći način:



tekući čvor	stanje otvorene liste [čvor(roditelj, $g+h$ )]	u zatvorenu listu se dodaje
	$A(-, 0+6)$	
$A(6)$	$D(A, 6+5), G(A, 5+6)$	$A(-)$
$D(11)$	$G(A, 5+6), F(D, 9+3), H(D, 9+4), C(D, 10+6)$	$D(A)$
$G(11)$	$F(D, 9+3), H(D, 9+4), C(D, 10+6)$	$G(A)$
$F(12)$	$B(F, 12+1), H(D, 9+4), C(D, 10+6)$	$F(D)$
$B(13)$	$H(D, 9+4), E(B, 14+0), C(D, 10+6)$	$B(F)$
$H(13)$	$E(B, 14+0), C(D, 10+6)$	$H(D)$
$E(14)$	$C(D, 10+6)$	

Na kraju primene algoritma, kada je čvor  $E$  postao tekući čvor, konstruiše se put:  $A - D - F - B - E$ . Ovo jeste put od čvora  $A$  do čvora  $E$ , ali nije najkraći mogući. Ovo ponašanje posledica je činjenice da funkcija  $h$  nije konzistentna: kada heurstika nije konzistentna, neophodno je proveravati i zatvorene čvorove.

Ukoliko se za isti problem koristi konzistentna funkcija, rezultat će biti optimalan put od  $A$  do  $E$ , a neće biti potrebno proveravati jednom zatvorene čvorove. U narednom primeru koristi se konzistentna heurstika  $h$  čija se vrednost razlikuje u odnosu na prethodnu samo za čvor  $H$  i daje optimalni put  $A - D - H - B - E$ .



tekući čvor	stanje otvorene liste [čvor(roditelj, g+h)]	u zatvorenu listu se dodaje
	$A(-, 0+6)$	
$A(6)$	$D(A, 6+5), G(A, 5+6)$	$A(-)$
$D(11)$	$G(A, 5+6), H(D, 9+3), F(D, 9+3), C(D, 10+6)$	$D(A)$
$G(11)$	$H(D, 9+3), F(D, 9+3), C(D, 10+6)$	$G(A)$
$H(12)$	$B(H, 11+1), F(D, 9+3), C(D, 10+6)$	$H(D)$
$B(12)$	$F(D, 9+3), E(B, 13+0), C(D, 10+6)$	$B(H)$
$F(12)$	$E(B, 13+0), C(D, 10+6)$	$F(D)$
$E(13)$	$C(D, 10+6)$	

### 4.3.2 Specijalni slučajevi primene

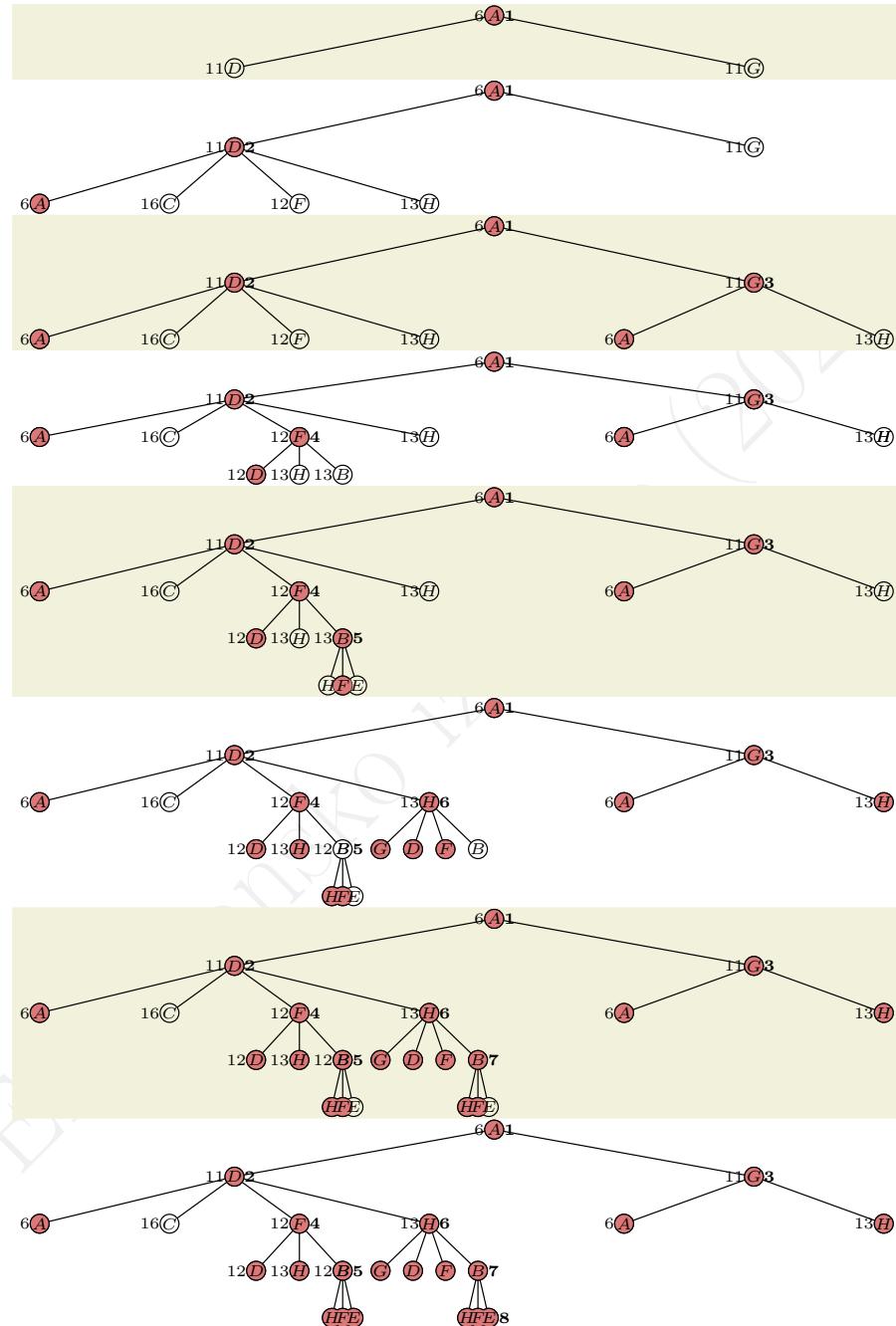
Obilasci grafa u dubinu i širinu mogu se smatrati specijalnim slučajevima algoritma A\*.

Za obilazak grafa u dubinu, može se koristiti algoritam A\* sa  $g(n) = 0$  i pogodno kreiranom funkcijom  $h$ . Na primer, neka je vrednost  $C$  inicijalizovana na neku veoma veliku vrednost. Kad god se obrađuje neki čvor, vrednost  $C$  se pridružuje kao vrednost funkcije  $h$  svim njegovim susedima koji nisu posećeni ranije. Nakon svake dodele neka se smanjuje vrednost  $C$  za jedan. Time će vrednost  $h(n)$  da bude veća za čvorove na koje se ranije naišlo. Ovako definisana funkcija  $h$  nije nužno dopustiva.

Za  $g(n) = 0$ , algoritam A\* predstavlja specijalnu varijantu pristupa **Prvo najbolji**, koja najpre obrađuje čvorove sa najboljom heurističkom vrednošću. Ova varijanta algoritma nije nužno optimalna.

Dejkstrin algoritam, kao specijalni slučaj obilaska grafa u širinu, takođe je specijalni slučaj algoritma A\* u kojem je  $h(n) = 0$  za svaki čvor  $n$ . Ovakva funkcija  $h$  je konzistentna i garantuje nalaženje optimalnog puta. Skup otvorenih čvorova širi se ravnomerno, slično koncentričnim krugovima oko polaznog čvora, baš kao kod Dejkstrinog algoritma. S druge strane, sa boljom heurstikom, skup otvorenih čvorova će se brže širiti ka ciljnou čvoru.

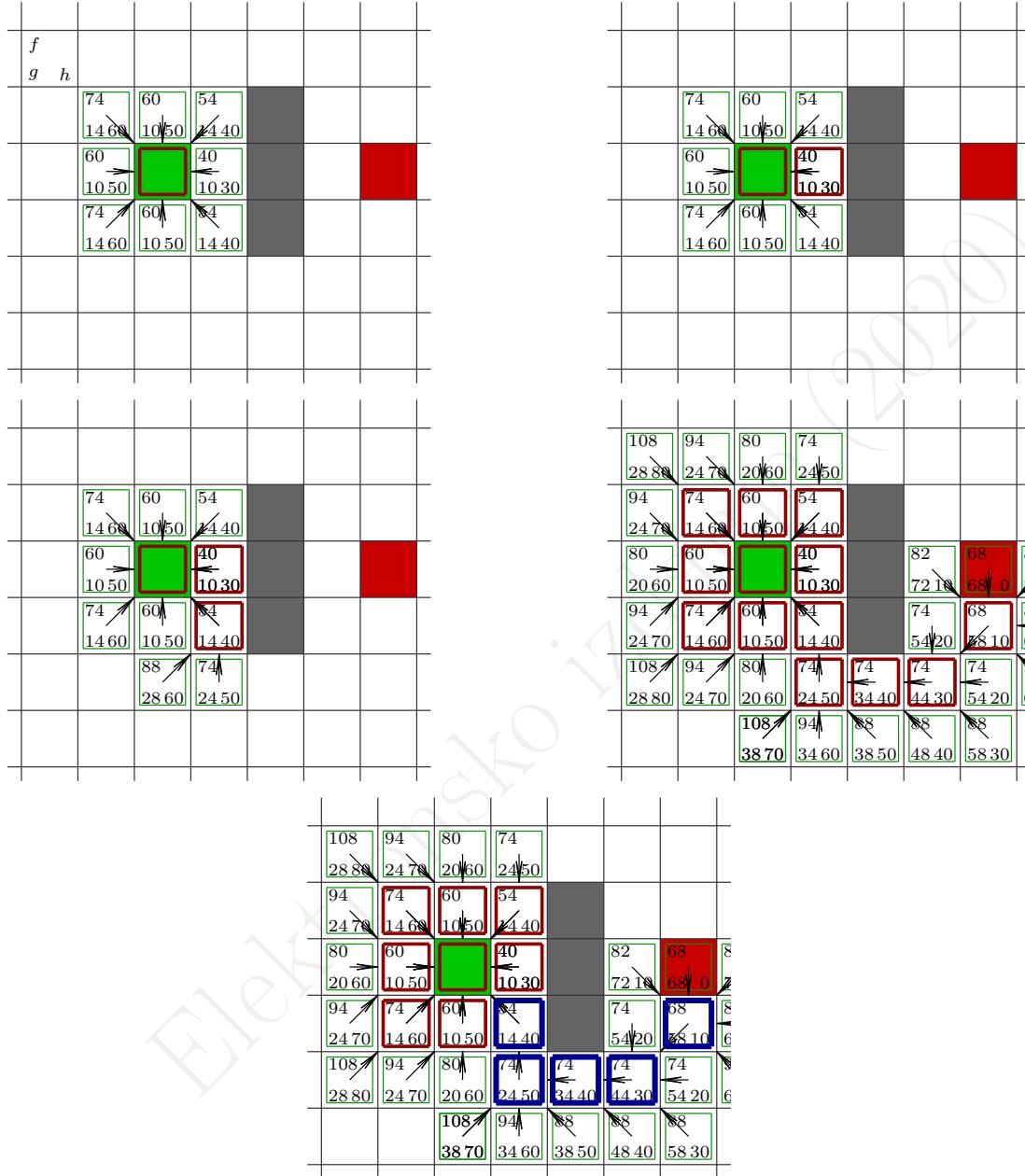
Opšti algoritam A\* često se primenjuje za pronalaženje puta na uniformnoj, kvadratnoj mreži čvorova (koja odgovara, na primer, diskretizovanoj ili rasterizovanoj mapi). Tada on dobija specifičnu formu. Prepostavimo da je mreža pravilna (sačinjena od kvadrata) i da ima pravougaonu formu. Dodatno, prepostavljamo da neki čvorovi (tj. neki kvadrati, neka polja mreže) nisu dostupni i da oni predstavljaju *prepreke*. Svako polje povezano je sa svakim susednim poljem osim sa preprekama, te ima najviše četiri susedna polja. Svakom horizontalnom ili vertikalnom potezu obično se pridružuje cena 1. Funkcija heuristike  $h$  može se zadati na različite načine. Kada se izračunava vrednost  $h$ , obično se, jednostavnosti i efikasnosti radi, ignorisu sve prepreke jer vrednost  $h(n)$  je procenjeno a ne stvarno rastojanje, a ignorisanjem prepreka biće *potcenjeno* stvarno rastojanje (što upravo i jeste poželjan uslov). Jedna mogućnost za heuristiku  $h$  je euklidsko rastojanje između dva polja:  $(d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2})$ . Ova funkcija je konzistentna i dopustiva te obezbeđuje optimalnost, ali je zahtevna što se tiče vremena izračunavanja (što može biti kritično za mape sa milionima čvorova). Drugi primer funkcije heuristike je Menhetn rastojanje u kojem se broji ukupan broj polja pređenih horizontalno ili vertikalno da bi se došlo od jednog do drugog polja:  $d((x_1, y_1), (x_2, y_2)) = |x_2 - x_1| + |y_2 - y_1|$ . Ova heuristika je konzistentna (i dopustiva), te garantuje pronalaženje optimalnog puta. Ukoliko su na mreži dozvoljeni i dijagonalni potezi, onda se svakom horizontalnom ili vertikalnom potezu obično pridružuje cena 1, a svakom dijagonalnom potezu cena  $\sqrt{2} \approx 1.414$  (ovakva cena odgovara euklidskom rastojanju između središta polja; da bi se koristilo celobrojno izračunavanje, ove vrednosti obično se množe nekom konstantom, na primer, 10, i zaokružuju na ceo broj). U ovom slučaju, Menhetn rastojanje može da precenjuje rastojanje do ciljnog



Slika 4.12: Stablo pretrage tokom primene algoritma A\* na problem iz primera 4.7 (levo od čvora je zapisana njegova  $f$  vrednost, a desno redni broj u nizu tekućih čvorova).

čvora, te nije dopustiva heuristika i ne garantuje pronađenje najkraćeg puta. No, ovo rastojanje u praksi često daje dobre rezultate i pronađeni putevi su obično dovoljno dobri, čak i ako nisu najkraći. U slučaju da su dozvoljeni i dijagonalni potezi, kao heuristika koja je konzistentna (i dopustiva) može se koristiti Čebiševljevo rastojanje:  $d((x_1, y_1), (x_2, y_2)) = \max(|x_2 - x_1|, |y_2 - y_1|)$ .

I kada heuristika nije konzistentna, mogu da se ne ažuriraju (i otvaraju ponovo) zatvoreni čvorovi. I ovakav pristup često daje dovoljno dobra i efikasna rešenja, iako ne nužno optimalna.



Slika 4.13: Ilustracija rada algoritma A\* na uniformnoj mreži uz korišćenje Menhetn rastojanja za heuristiku.

**Primer 4.8.** Primenom algoritma A\* potrebno je pronaći put od polaznog do ciljnog čvora na uniformnoj mreži prikazanoj na slici 4.13. Polazni čvor označen je zelenom, a ciljni crvenom bojom. Dozvoljeni su horizontalni, vertikalni i dijagonalni potezi. Nije moguće dijagonalno kretanje ka polju gore-desno ako je desno polje prepreka. Nije moguće dijagonalno kretanje ni u drugim analognim situacijama. Ovakvo ograničenje zavisi od prirode konkretne primene (na primer, ovakvo ograničenje može da opisuje moguće kretanje vozila).

Vrednosti funkcija  $f$ ,  $g$  i  $h$  zapisane su u poljima uniformne mreže: vrednost funkcije  $f$  zapisana je

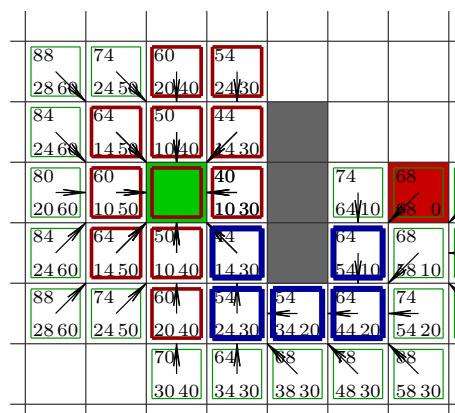
gore-levo, vrednost funkcije  $g$  dole-levo, a vrednost funkcije  $h$  dole-desno. Vrednost funkcije  $f$  za svako polje je, kao i obično, zbir vrednosti funkcija  $g$  i  $h$ . Za rastojanje između dva polja susedna horizontalno ili vertikalno uzima se vrednost 10, a za rastojanje između dva polja susedna dijagonalno uzima se vrednost 14. Kao heuristika  $h$  koristi se Menhetn rastojanje do ciljnog polja pomnoženo sa 10 (prepreka se zanemaruje). Ova heuristika nije dopustiva jer su dozvoljeni i dijagonalni potezi (ova tema biće diskutovana i naknadno u okviru ovog primera). Otvorena polja označena su dodatnim tankim (zelenim) kvadratima u okviru polja, zatvorena polja označena su dodatnim unutrašnjim (crvenim) kvadratima srednje debljine, a polja koja čine pronađeni put označena su dodatnim unutrašnjim debljim (plavim) kvadratima. Strelice ukazuju na tekućeg roditelja polja.

Pretraga kreće od polaznog polja jer je na početku samo ono u otvorenoj listi. Polazno polje briše se iz otvorene liste i dodaje u zatvorenu listu. U otvorenoj listi je onda njegovih osam susednih polja. Od svih njih, bira se ono sa najmanjom vrednošću funkcije  $f$  (40), to je polje neposredno desno od polaznog polja i ono će biti sledeće tekuće polje. Novo tekuće polje izbacuje se iz otvorene liste, dodaje se u zatvorenu listu i onda se proveravaju njegova susedna polja. Ta četiri polja već su u otvorenoj listi, pa je potrebno proveriti da li put preko tekućeg čvora popravlja njihove trenutne ocene. Razmotrimo, na primer, polje neposredno iznad tekućeg polja: vrednost funkcije  $g$  za njega je 14. Ukoliko bi se do njega dolazio preko tekućeg polja, vrednost funkcije  $g$  bila bi 20 (10 je cena od polaznog do tekućeg čvora i 10 je cena prelaska od tekućeg polja). Dakle, na ovaj način se ne može popraviti vrednost funkcije  $g$  u polju iznad i ona ostaje nepromenjena. Pretraga se i u nastavku sprovodi kao kod opštег algoritma. Kada ciljno polje postane tekuće, traženi put od polaznog čvora konstruiše se jednostavno: kreće se od ciljnog čvora i prelazi na roditeljski sve dok se ne dođe do polaznog čvora. Ovako određen niz polja u suprotnom poretku daje traženi put.

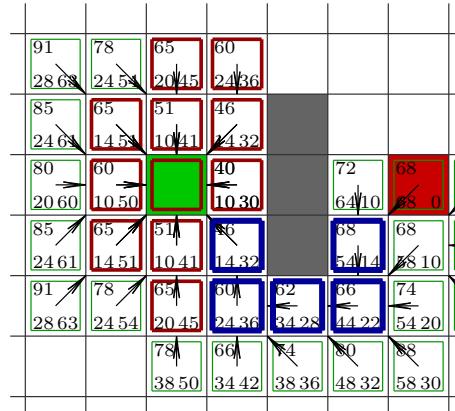
Primetimo kako vrednost funkcije  $f$  za neko polje može da se promeni tokom primene algoritma. Nakon nekoliko iteracija, vrednosti funkcija  $g$  i  $f$ , kao i roditeljsko polje, promenili su se za polje koje se nalazi dva polja ispod polaznog polja. Ranije je ovo polje imalo vrednost funkcije  $g$  jednaku 28 (i vrednost funkcije  $f$  jednaku 88) i roditeljsko polje je bilo gore-desno. Kasnije, ovo isto polje ima vrednost funkcije  $g$  jednaku 20 (i vrednost funkcije  $f$  jednaku 80), a roditeljsko polje je gore. Ova izmena dogodila se u nekoj iteraciji u međuvremenu.

Primetimo da Menhetn rastojanje koje je korišćeno ne daje dopustivu heuristiku. Na primer, za polje koje je neposredno ispod prepreke ocena heuristike je 40 i ona je veća od stvarnog rastojanja do ciljnog čvora koji je jednak 28. Kako su dozvoljeni dijagonalni potezi, a za heuristiku je korišćeno Menhetn rastojanje, ne može se garantovati da će uvek biti dobijen optimalni put. Optimalnost puta bila bi garantovana da je korišćeno Čebiševljevo rastojanje. Uprkos ovome, i u ovakvim situacijama može da se koristi Menhetn rastojanje jer može brže da vodi cilju (iako možda ne dajući optimalan put).

Rad algoritma A\* na istom grafu, ali za heuristike zasnovane na Čebiševljevom rastojanju i na euklidiskom rastojanju ilustrovan je na slikama 4.14 i 4.15. U ova dva slučaja, kako su ove heuristike konzistentne, duž pronađenog puta niz vrednosti  $f$  od početnog ka ciljnog čvoru je neopadajući niz. To ne važi kada se za heuristiku koristi rastojanje Menhetn (jer ona nije konzistentna kada su dozvoljeni i dijagonalni potezi, slika 4.13).



Slika 4.14: Ilustracija rada algoritma A\* na uniformnoj mreži uz korišćenje Čebiševljevog rastojanja za heuristiku.



Slika 4.15: Ilustracija rada algoritma A\* na uniformnoj mreži uz korišćenje euklidskog rastojanja za heuristiku.

### 4.3.3 Implementaciona pitanja

Algoritam A\* obično se primjenjuje u aplikacijama koje rade u realnom vremenu, te je neophodno da je efikasno implementiran. Otvorena lista često se implementira kao binarni min-heap (kako bi se brzo dolazilo do elementa sa najmanjom vrednošću funkcije  $f$ ), a zatvorena lista kao heš tabela. Korišćenjem ovih struktura, operacije za dodavanje i brisanje elemenata iz otvorene liste zahtevaju vreme  $O(\log |V|)$ , gde je  $V$  skup čvorova grafa, a operacije proveravanja da li je element u zatvorenoj listi, dodavanja u zatvorenu listu, kao i brisanja iz zatvorene liste zahtevaju prosečno vreme  $O(1)$ .

Zahtevi za memorijskim prostorom su za algoritam A\* često još veći problem nego vremenska složenost. Ipak, ukoliko broj čvorova grafa nije preveliki, može da bude isplativo i statičko alociranje potrebnog prostora (ili dinamičko alociranje većih blokova) koji onda može da se koristi u savezu sa min-heap strukturom, kako bi se izbegle česte i skupe operacije dinamičkog alociranja (i dealociranja) za pojedinačne čvorove.

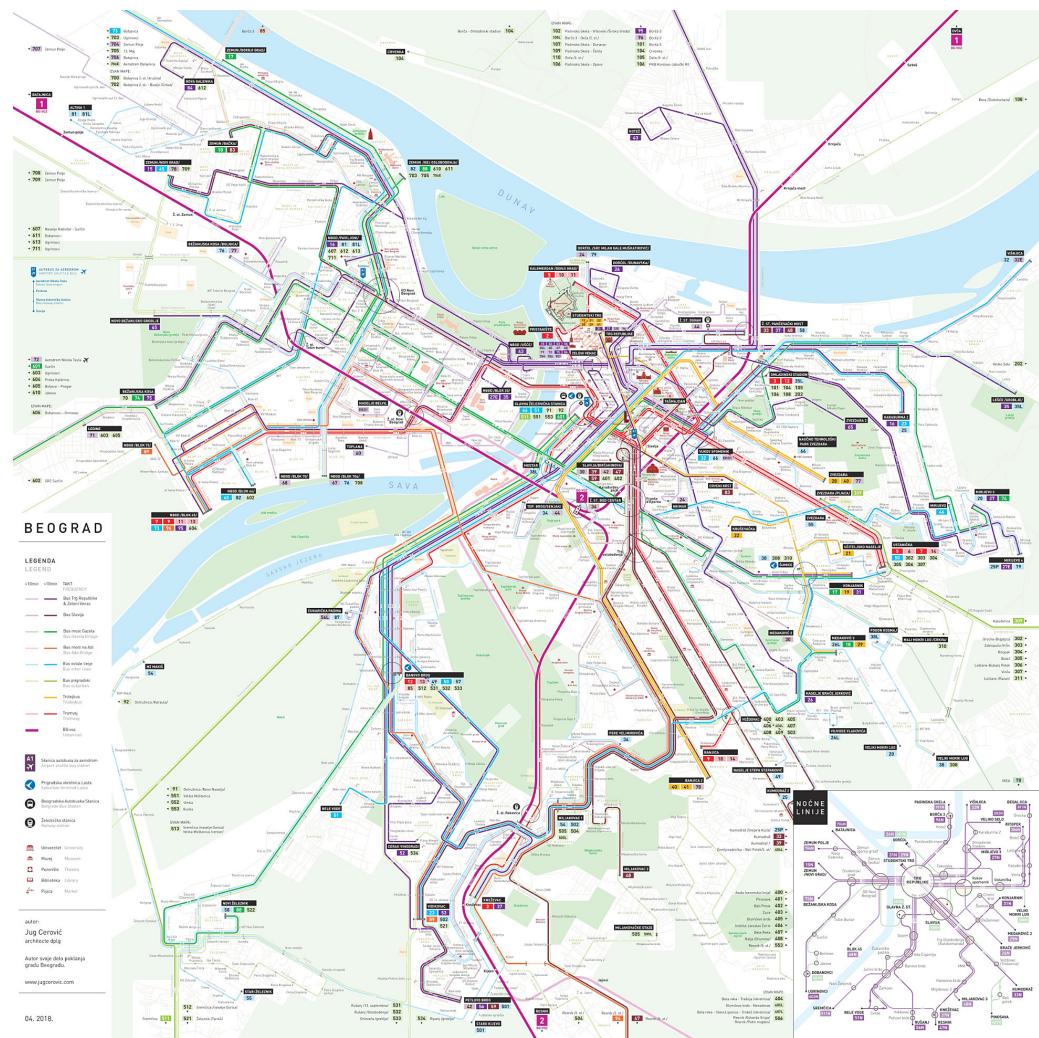
Najgori slučaj za algoritam A\* je kada ne postoji put između polaznog i ciljnog čvora. U tu svrhu može se implementirati brza provera da li uopšte postoji put između dva čvora: dva čvora su povezana ako i samo ako pripadaju povezanim delovima grafa. Ako se za svaki čvor može lako proveriti kom delu grafa pripada, onda je i navedena provera jednostavna (posebno ako mapa može biti obrađena unapred i podeljena na povezane delove).

Kada se algoritam A\* koristi za pronalaženje puta na uniformnoj mreži, on daje korake u osam mogućih smerova što može da dovede do putanja koje ne izgledaju prirodno (posebno ako one treba da opišu kretanje nekog vozila). Takve puteve potrebno je unaprediti *omekšavanjem*, tj. zameniti sličnim putevima koji izgledaju prirodnije.

### 4.3.4 Primer rešavanja problema korišćenjenje algoritma A\*

Razmotrimo sledeći realan problem: potrebno je za neki grad napraviti aplikaciju koja predlaže najbolji put od jednog do drugog stajališta javnog prevoza (Beograd, na primer, ima oko 160 linija javnog prevoza i oko 2400 stajališta, slika 4.16). Kvalitet pronađenog puta može se meriti dužinom puta ili procenom potrebnog vremena za njegovo prelaženje.

Opisani zadatak može se efikasno rešiti primenom algoritma A\*, sa skupom stajališta koji čini skup čvorova grafa. Ukoliko je cena puta definisana kao njegova dužina, pogodna heurstika može biti zasnovana na euklidiskom rastojanju i ona je očigledno dopustiva. Ukoliko je cena puta definisana kao utrošeno vreme, onda stvarna cena može da se zasniva na prosečnom vremenu čekanja na prevoz (izračunatom na osnovu frekvencije vozila na konkretnim linijama), a kao dopustiva heurstika može da se koristi procena vremena koja podrazumeva da nema nikakvog čekanja i koja podrazumeva maksimalnu brzinu kretanja vozila. U najjednostavnijoj varijanti, skup čvorova (usmerenog) grafa jednak je skupu stajališta. Grana između dva čvora postoji ako i samo ako neka linija povezuje ta dva stajališta. To stvara sledeći problem: algoritam ne može da razlikuje kvalitet putanja sa presedanjima i bez presedanja. Alternativa je da svako stajalište daje onoliko čvorova grafa koliko linija staje na tom stajalištu. Na primer, ako na stajalištu 10 staju linije 22 i 29, postojiće čvorovi 10-22 i 10-29. Granama će biti spojeni sva susedna stajališta neke linije. Cena takve grane biće određena na osnovu rastojanja između ta dva stajališta. Dodatno, granama će biti spojeni čvorovi koji odgovaraju istom stajalištu a različitim linijama. Cena ovakvih grana može da se bira slobodno i ona kontroliše koliko su prihvatljiva presedanja. Kao i cene puteva u grafu, i funkcije  $f$ ,  $g$  i  $h$  će odgovarati dužini pređenog puta ili proceni utrošenog vremena. Ukoliko odgovaraju utrošenom vremenu, bolji rezultati će se dobijati ako se uzimaju u obzir informacije koje utiču na



Slika 4.16: Mapa linija javnog prevoza Beograda.

brzinu saobraćaja: lokacija grane, doba dana, dan u nedelji i slično.

U Beogradu, na jednom stajalištu staje prosečno po 5 različitih linija, pa bi ukupan broj čvorova grafa bio oko 12000. Za ovakav graf, kvalitetna implementacija trošiće za jedan upit samo delić sekunde.

Opisani pristup ne razmatra (realnu) opciju da putnik pešači između dva (relativno bliska) stajališta. Da bi i to bilo omogućeno, grafu treba dodati i „pešačke čvorove“ za svako stajalište, grane koje povezuju svaki postojeći čvor stajališta sa tim novim čvorom, kao i grane koje povezuju „pešačke čvorove“ bliskih stajališta (na primer, na rastojanju manjem od 300m). Cena takvih grana može biti grubo procenjena na bazi euklidskog rastojanja. Ukoliko je cena puta bazirana na vremenu, onda se ovo rastojanje množi nekim težinskim faktorom (onoliko koliko je pešak sporiji od autobusa). Moguće su i mnoge druge modifikacije osnovnog rešenja koje mogu dati kvalitetnije i upotrebljivije odgovore.

## Glava 5

# Igranje strateških igara

Automatsko igranje strateških igara kao što je šah davnašnji je izazov. Još početkom dvadesetog veka španski pronalazač Torres Kevedo (Torres y Quevedo) konstruisao je (i prikazao na svetskoj izložbi u Parizu 1914. godine) elektro-mehanički uređaj *El Ajedrecista* („Šahista“) koji je, kao beli, igrao šahovsku završnicu „kralj i top protiv kralja“ i iz svake pozicije nepogrešivo pobedivao (iako ne u najmanjem mogućem broju poteza). Opšte razmatranje teorije igara započeo je Džon fon Nojman (John von Neumann) postavljanjem opštег problema (1928. godine): *Igrači  $S_1, S_2, \dots, S_n$  igraju datu igru  $\Gamma$ . Kako treba da igra igrač  $S_m$  da bi ostvario najbolji mogući rezultat?* Već od polovine dvadesetog veka, problemi ove vrste bili su važan i često pokretački, motivišući izazov za oblast u nastajanju – veštačku inteligenciju. Neki od najvećih (ili makar najšire poznatih) uspeha veštačke inteligencije ostvareni su upravo na polju strateških igara: računari su već odavno pobedili svetske šampione u igrama bekemon, dame i šah (tada važeći svetski šampion Gari Kasparov partiju šaha izgubio je od računara 1997. godine), a početkom 2016. godine i u igri go. Iako su ovi programi veoma uspešni, njihovi principi odlučivanja kvalitativno su (po pitanjima apstrahovanja, analogija, pravljenja planova i sl.) veoma različiti od ljudskih. Većina najznačajnijih programa za igranje igara zasnovana je na efikasnim algoritmima pretrage, a odnedavno i na naprednim tehnikama mašinskog učenja.

U nastavku neće biti upuštanja u analize pojedinačnih igara, već će biti opisani opšti pojmovi i algoritmi koji mogu da se koriste za širok spektar strateških igara. Biće razmatrani algoritmi za takozvane igre nulte sume bez nepoznatih informacija za dva igrača, dakle – igre kod kojih igrači, grubo rečeno, imaju analogne, simetrične mogućnosti i svaki igrač zna koje poteze na raspolažanju ima protivnik. U ovu kategoriju spadaju, na primer, igre šah, dame, go, reversi, iks-oks, četiri u nizu, mankala, a ne spadaju, na primer, igre u kojima igrač ne zna karte koje ima protivnik, nepoznati broj koji treba pogoditi itd.

### 5.1 Opšte strategije za igranje igara

Moderna istorija programiranja strateških igara počinje člankom *Programming a digital computer for playing Chess* Kloda Šenona iz 1950. godine. U tom tekstu, Šenon je opisao dve opšte strategije za izbor poteza:

- A:** „Minimaks“ procedurom vrši se pretraživanje stabla igre sa određenom funkcijom evaluacije i ocenjivanje legalnih poteza; bira se potez sa najboljom ocenom (videti poglavljje 5.4.2).
- B:** Potez se bira na osnovu trenutne pozicije/situacije u igri i na osnovu odgovarajuće, unapred pripremljene tabele.

Pristup zasnovan na Šenonovoj **A** strategiji naziva se „sistemskim“ ili „dubinskim pretraživanjem“ a i pristupom „gruba sila plus jednostavna vrednosna funkcija“. Ako bi se pretraživanje stabla igre vršilo do završnih stanja, efektivno bi bili ispitivani svi mogući tokovi nastavka partije i mogao bi da bude izabran zaista najbolji potez. Međutim, to (a ponekad čak ni pretraživanje stabla igre do dubine od svega nekoliko poteza) za netrivijalne igre nije praktično ostvarivo. Zbog toga, efikasna primena Šenonove **A** strategije svodi se na pretraživanje stabla igre do relativno male dubine algoritmima koji su usmereni heuristikama i uz dobro osmišljenu, ali jednostavnu funkciju evaluacije za ocenu nezavršnih pozicija igre (umesto jednostavne „tovrednosne“ funkcije za ocenu završnih pozicija). Ovakvim pristupom gubi se svojstvo po kojem se pretraživanjem dobija zaista najbolji potez, a obim pretraživanja i različitim izračunavanja ostaje, najčešće, i dalje veoma veliki. Precizniji opisi pojmova stabla igre, funkcije evaluacije i algoritama minimaks tipa dati su u poglavljju 5.4.

Šenonova strategija **B** tipa zasniva se na jednostavnoj, unapred pripremljenoj tabeli koja zamenjuje izračunavanje u toku izvršavanja. U ovom pristupu, znanje o igri ne nalazi se u programu koji igra, već u programu koji je

tabelu generisao. Tabela ima dve kolone: u jednoj su moguće pozicije/stanja igre, a u drugoj preporučeni (ponekad optimalni) potezi. Jedan od „klasičnih“ primera ovog pristupa je program za igranje šahovske završnice *kralj i kraljica protiv kralja i topa* koji je 1977. godine kreirao Kenet Tompson (Kenneth Thompson, tvorac operativnog sistema UNIX). Tabela koju je koristio program sadržavala je sve moguće pozicije i optimalne poteze za sve te pozicije (pri čemu se pod optimalnim potezom za igrača koji ima kralja i kraljicu smatra potez koji vodi pobedi u najmanjem broju poteza, a za slabijeg – potez koji maksimalno odlaže poraz). Tabela je imala oko tri miliona vrsta i program koji se na njoj zasnivao bio je, naravno, nepogrešiv. Tabela je kreirana korišćenjem *retrogradne analize*: za optimalnu igru igrača koji ima kraljicu, prepostavimo da je to beli, najpre se prepoznaju i označavaju pozicije u kojima je protivnik matiran (mat u nula poteza belog); dalje, prepoznaje se i označava svaka pozicija i u tabelu se upisuje odgovarajući potez nakon kojeg, ma šta da odigra protivnik, postoji mat u nula poteza (za takve pozicije važi da postoji mat u jednom potezu belog), dalje, prepoznaje se i označava svaka pozicija i u tabelu se upisuje odgovarajući potez nakon kojeg, ma šta da odigra protivnik, postoji mat u najviše jednom potezu belog (za takve pozicije važi da postoji mat u najviše dva poteza belog), ... dalje, prepoznaje se i označava svaka pozicija i u tabelu se upisuje odgovarajući potez nakon kojeg, ma šta da odigra protivnik, postoji mat u najviše  $d$  poteza belog (za takve pozicije važi da postoji mat u najviše  $d + 1$  poteza)... Ovaj postupak nastavlja se dok god postoji neka pozicija u kojoj je na potezu beli, a kojoj nije pridružen optimalni potez. Korišćenjem istog pristupa, na moskovskom univerzitetu su 2012. godine kreirane *Lomonosov* tabele optimalnih poteza za sve šahovske završnice sa najviše sedam figura na tabli. Tabela sadrži više od 500 triliona pozicija (pri čemu se u tabeli ne čuvaju pozicije koje se mogu dobiti od drugih pozicija simetrijama i rotacijama).

Šenonova strategija **A** u procesu izbora poteza zahteva malo memorije i mnogo izračunavanja, a strategija **B** malo izračunavanja i mnogo memorije. Na toj skali odnosa količine podataka koji se koriste i obima izračunavanja, čovekov način zaključivanja je između ovih krajnosti i bitno se od njih razlikuje po svojoj prirodi.

U dvadeset prvom veku, na popularnosti dobija i treća opšta strategija za igranje strateških igara zasnovana na Monte Karlo pretrazi i velikom broju simulacija kojima se, u sadejstvu sa mašinskim učenjem, ocenjuju mogući potezi u nekoj poziciji.

## 5.2 Legalni potezi i stablo igre

Pravila konkretne igre definišu *legalna stanja* (tj. legalne pozicije) i *legalne poteze* za svaku legalnu poziciju.<sup>1</sup> Za svaku legalnu poziciju može se efektivno odrediti skup legalnih poteza. Neke legalne pozicije mogu biti *početne pozicije* a neke *završne*. U nekim igrama, legalni potez može biti i *dalje*, u situaciji kada igrač koji je na redu nema na raspolaganju legalnih poteza i preskače svoj red (takve situacije ne postoje u šahu, ali postoje, na primer, u igri reversi).<sup>2</sup>

Prostor stanja igre može se opisati grafom čiji su čvorovi legalne pozicije, a grane legalni potezi. Taj graf, graf prostora stanja, je usmeren jer nije nužno da postoje potezi u oba smera koji povezuju dva stanja (na primer, igri u šah pešak može da se kreće samo napred, ne i nazad). *Stablo igre* je stablo u čijim su čvorovima legalne pozicije i za svaki čvor njegova deca su sve pozicije do kojih se iz tog čvora može doći po jednim legalnim potezom. Od korena do bilo kog lista naizmenično se, dakle, smenjuju grane koje odgovaraju potezima prvog i drugog igrača. *Potpuno stablo igre* je stablo igre u čijem je korenu početna pozicija igre, a svi listovi su završne pozicije igre i svakom listu pridružen je ishod – pobeda prvog igrača, nerešeno ili pobeda drugog igrača. Potpuno stablo igre ima onoliko listova koliko data igra ima različitih mogućih tokova. Taj broj je kod većine igara (čak i kod veoma jednostavnih) ogroman i onemogućava kompletno pretraživanje u cilju izbora poteza. Na slici 5.1 ilustrovani je deo potpunog stabla igre za igru iks-oks.

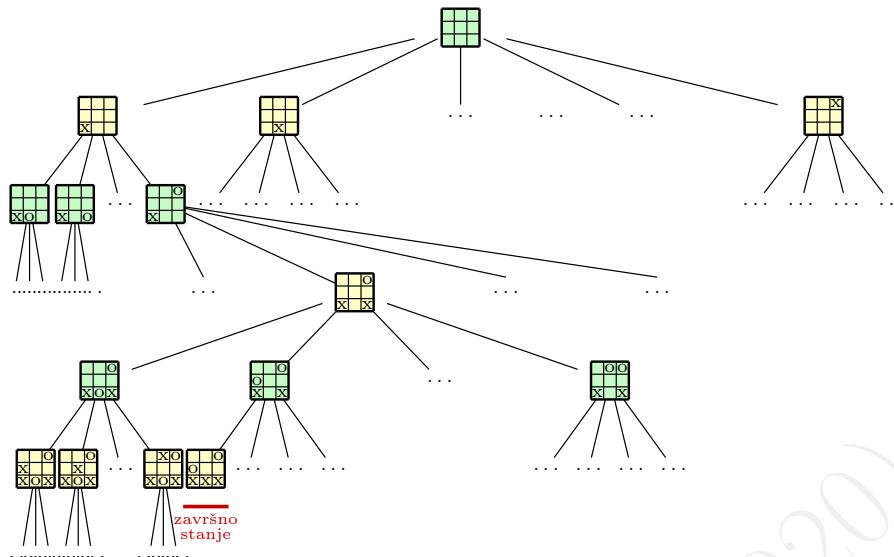
## 5.3 Otvaranje

U strateškim igrama, umesto da se na samom početku igre program upusti u proces pretrage, obično se koriste *knjige otvaranja* (eng. *opening book*) koje su zasnovane na ljudskom iskustvu i koje sadrže informacije o poznatim i kvalitetnim potezima koji se često javljaju u otvaranju.<sup>3</sup> Knjiga otvaranja koristi se, ne zahtevajući mnogo vremena za izvršavanje, dok god je to moguće (u šahu, na primer, obično za prvih desetak poteza) sa očekivanjem da se dođe do bolje pozicije nego da je program koristio druga znanja. Kada više nije moguće koristiti knjigu otvaranja, prelazi se na druge strategije izbora poteza.

<sup>1</sup> Nekad se ono što odigra jedan igrač smatra *polupotezom* (eng. *ply*), a par takvih uzastopnih polupoteza smatra se *potezom* (eng. *move*). U ovom tekstu će se ipak smatrati da je potez ono što odigra jedan igrač.

<sup>2</sup> U implementacijama, struktura koja opisuje potez treba da sadrži informacije dovoljne da bi potez bio odigran, ali i da bi mogao da bude vraćen, tj. da iz dobijene pozicije može da se rekonstruiše polazna pozicija.

<sup>3</sup> U evropskoj istoriji šaha, otvaranja se sistematično proučavaju već više od četiristotine godina.



Slika 5.1: Deo potpunog stabla igre za igru iks-oks.

Ukoliko protivnik odigra neki neuobičajen potez koji ne postoji u knjizi otvaranja, program je primoran da promeni pristup i da počne da vrši pretragu nastavka stabla igre. Takav neuobičajeni potez može, međutim, često da bude rizičniji za igrača koji ga je odigrao nego za program koji u daljem toku može da kompenzuje rano napuštanje knjige otvaranja i iskoristi slabosti protivnika.

Ukoliko za neku poziciju postoji u knjizi otvaranja više mogućih poteza, izbor može da se načini po verovatnoćama koje oslikavaju kvalitet tih poteza. Pomenuti pristup može da se realizuje, na primer, na sledeći način: neka je, na osnovu knjige otvaranja, u nekom trenutku na raspaganju  $n$  poteza. Svakom od njih neka je pridružena neka empirijska nenegativna ocena kvaliteta  $m_i$ ,  $1 \leq i \leq n$  (na primer, na osnovu ishoda svih partija koje uključuju taj potez ili na osnovu broja takvih partija u knjizi, a kao težnja da se što duže ostane u poznatim varijantama). Što je neki potez bolji, to je njegova ocena veća. Tada se, u toj poziciji,  $i$ -ti potez ( $1 \leq i \leq n$ ) bira sa verovatnoćom

$$p_i = \frac{m_i}{\sum_{j=1}^n m_j}.$$

Na taj način izbegava se determinističko ponašanje programa u otvaranju: bolji potezi (u smislu neke procene) biraju se češće, ali ne uvek. Ocene  $m_i$  mogu se tokom vremena i korigovati.

Knjiga otvaranja može biti statička (sadržati određen, konačan broj varijanti u svakom potezu i informacije o potezima samo do o dredene dubine) ili se proširivati tokom samog izvršavanja programa.

## 5.4 Središnjica

Savremeni programi za strateške igre u središnjici najčešće koriste Šenonovu **A** strategiju — koriste dubinsko pretraživanje stabla igre sa pogodnom funkcijom evaluacije. Očekuje se da su potezi dobijeni na ovaj način dobro dobro (ali ne nužno i zaista najbolji mogući). Funkcija evaluacije izračunava se samo za čvorove na nekoj izabranoj dubini, a ostalim čvorovima ocena se određuje na osnovu ocena njihovih potomaka. Pored kvalitetne funkcije evaluacije, od ključne važnosti su algoritmi koji se koriste za pretraživanje stabla igre vođeni raznovrsnim heuristikama. Pretraživanjem stabla igre nastaje *stablo pretraživanja* ili *stablo pretrage*. Za konkretnu poziciju, stablo pretrage obično čini samo mali deo kompletног stabla igre, tj. algoritam posećuje samo mali deo skupa pozicija u stablu igre. Algoritam je efikasniji što je taj deo manji. Kao i u drugim vrstama pretrage, u procesu traženja poteza, stablo pretrage ne kreira se eksplicitno, kao struktura u memoriji računara, već samo implicitno.

### 5.4.1 Funkcija evaluacije i statička ocena pozicije

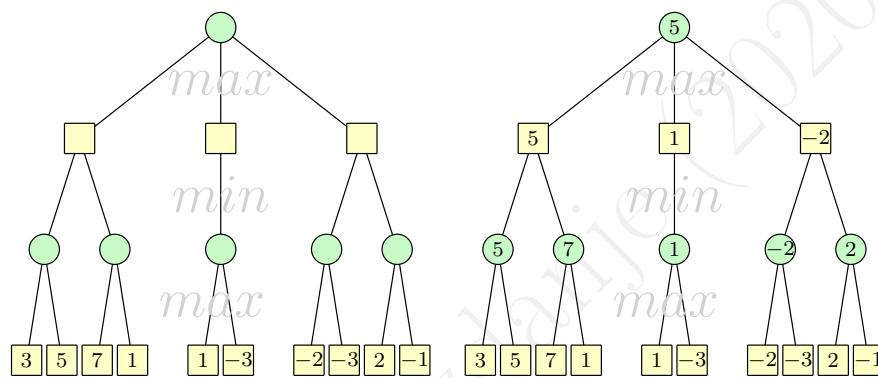
*Funkcija evaluacije* (eng. *evaluation function*) dodeljuje pozicijama, tj. čvorovima stabla igre, neke ocene na osnovu kojih one mogu da se porede po kvalitetu. Ocena pozicije formira se u skladu sa specifičnostima konkretnе igre, pri čemu se ne ispituju ni pozicije iz kojih se došlo u tu poziciju niti mogući nastavci, te se zato za ovu ocenu kaže da je *statička ocena pozicije*. Gotovo sve znanje o igri koje se koristi u središnjici partije

sadržano je u funkciji evaluacije i u velikoj meri od nje zavisi kvalitet igre programa. Potrebno je da ona koristi što više relevantnih informacija, ali, s druge strane, kako se izračunava mnogo puta, potrebno je da bude što jednostavnija. Funkcija evaluacije obično preslikava skup svih mogućih pozicija u simetričan segment  $[-M, M]$ . Tada se vrednost  $M$  dodeljuje samo završnim čvorovima u kojima je pobednik prvi igrač, a vrednost  $-M$  samo završnim čvorovima u kojima je pobednik drugi igrač.<sup>4</sup>

U igrama nulte sume, smisao funkcije evaluacije za protivnike u igri za dva igrača je suprotan — ono što je najbolje stanje za jednog igrača najlošije je za drugog i obratno. Dakle, funkcija evaluacije za simetrične pozicije (za zamenjene uloge igrača) treba da daje vrednosti koje se razlikuju samo po znaku.

Najjednostavnija je tzv. trovrednosna funkcija evaluacije: ona se primenjuje samo na završne pozicije igre i ima samo tri različite vrednosti — za pobedu prvog, za pobedu drugog igrača i za nerešen ishod (na primer, 1, -1 i 0). Trovrednosna funkcija zahteva pretraživanje stabla igre do završnih čvorova, pa je, zbog velike dubine pretraživanja, ova funkcija za većinu igara praktično neupotrebljiva.

U šahu funkcija evaluacije obično uključuje vrednost ukupnog „materijala“, pokretljivost figura, pešačku strukturu, rokade i slično. Beloj kraljici, na primer, može da bude pridružena vrednost 100, topu 50, lovcu i konju 30, a pešaku 10.



Slika 5.2: Ilustracija primene algoritma Minimaks.

#### 5.4.2 Algoritam Minimaks

Algoritam Minimaks ključni je element Šenonove A strategije i on je u osnovi skoro svih algoritama za izbor poteza sistematičnim pretraživanjem stabla igre. Ovaj algoritam prvi je opisao Fon Nojman još 1928. godine.

Minimaks algoritam pretraživanjem stabla igre za igrača koji je na potezu određuje najbolji mogući potez u datoj situaciji — pri čemu se pod „najboljim“ podrazumeva najbolji za zadati čvor, zadatu dubinu pretraživanja i za izabranu funkciju evaluacije. Pretpostavimo da funkcija evaluacije za igrača koji je na potezu ima pozitivan smisao, tj. bolji je potez ako obezbeđuje veću vrednost funkcije. Funkcijom evaluacije ocene se dodeljuju čvorovima na zadatoj, fiksnoj dubini (ti čvorovi ne moraju da predstavljaju završna stanja igre). Dalji postupak je rekurzivan: kao ocena čvoru dodeljuje se minimum ocena njegove dece ako je u tom čvoru na potezu protivnik, a maksimum ocena njegove dece, u suprotnom (slika 5.2). Ocena početnog čvora je maksimum ocena njegove dece i bira se potez kojem odgovara taj maksimum. Dakle, algoritam karakteriše minimizovanje ocene kada je na potezu protivnik i maksimizovanje ocene kada je na potezu sâm igrač, pa otuda i ime algoritma. Postupak naizmeničnog minimizovanja i maksimizovanja zovemo i *minimaxizacija*. Algoritam Minimaks dat je na slici 5.3 (algoritam se može jednostavno optimizovati eliminisanjem ponovljenih funkcijskih poziva; algoritam je prikazan u postojećem obliku zbog jednostavnosti).

**Primer 5.1.** Na slici 5.4 prikazan je primer primene algoritma Minimaks na šah (pojednostavljenu verziju na tabli  $4 \times 4$ ). Kao najbolji potez za koren čvor bira se prvi potez naveden u sledećem redu — potez koji vodi u mat u dva poteza.

Algoritam Minimaks (kao i ostali algoritmi zasnovani na minimaksizaciji) vrši izbor poteza samo na osnovu vrednosti koje su pridružene čvorovima na maksimalnoj dubini pretraživanja. To znači da se ne ispituju potezi koji dalje slede (a to ispitivanje često bi promenilo odluku o izabranom potezu). Dodatno, kada je neki potez

<sup>4</sup>Funkcije evaluacije obično se, efikasnosti radi, implementiraju kao celobrojne funkcije.

**Algoritam:** Minimaks

**Ulaz:** Funkcija evaluacije  $f$ , pozicija, dozvoljena dubina pretraživanja  $d$

**Izlaz:** Potez

- 1:  $v := \text{Max}(f, \text{pozicija}, d)$
- 2: vrati potez kojem odgovara vrednost  $v$

**Algoritam:** Max

**Ulaz:** Funkcija evaluacije  $f$ , pozicija, dozvoljena dubina pretraživanja  $d$

**Izlaz:** Vrednost pozicije

- 1: **ako** je pozicija završna ili je  $d$  jednako 0 **onda**
- 2:       vrati  $f(\text{pozicija})$
- 3:  $v := -\infty$
- 4: **za** svaku poziciju  $s$  do koje se može doći u jednom potezu **radi**
- 5:       **ako**  $\text{Min}(f, s, d - 1) > v$  **onda**
- 6:            $v := \text{Min}(f, s, d - 1)$
- 7: vrati  $v$

**Algoritam:** Min

**Ulaz:** Funkcija evaluacije  $f$ , pozicija, dozvoljena dubina pretraživanja  $d$

**Izlaz:** Vrednost pozicije

- 1: **ako** je pozicija završna ili je  $d$  jednako 0 **onda**
- 2:       vrati  $f(\text{pozicija})$
- 3:  $v := +\infty$
- 4: **za** svaku poziciju  $s$  do koje se može doći u jednom potezu **radi**
- 5:       **ako**  $\text{Max}(f, s, d - 1) < v$  **onda**
- 6:            $v := \text{Max}(f, s, d - 1)$
- 7: vrati  $v$

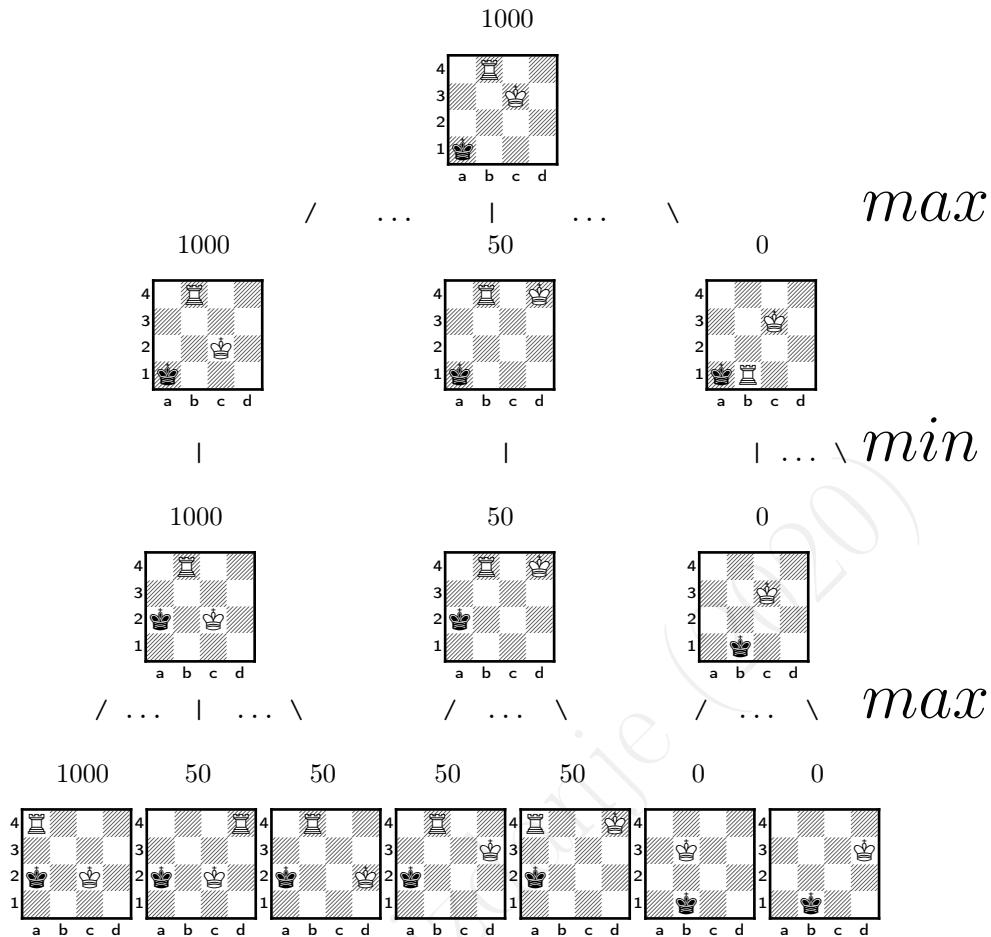
Slika 5.3: Algoritam Minimaks.

odabran na osnovu čvorova na nekoj dubini i odigran, informacija o tome ne koristi se u procesu izbora narednog poteza (na primer, ako je u šahu neki potez izabran jer odgovara čvoru dubine tri u kojem se protivniku daje šah, u sledećem potezu pretraživanje kreće iznova i često neće biti izabran potez koji vodi do šaha protivniku, sada u dva poteza). Dakle, pri pretraživanju stabla igre razmatraju se („vide se“) samo čvorovi na nekoj fiksnoj dubini a ne oni posle njih. Za ovakvo ponašanje često se kaže da ima *efekat horizonta* (eng. *horizon effect*).

#### 5.4.3 Algoritam Alfa-beta

Algoritam Alfa-beta (ili  $\alpha - \beta$ ) otkriven je sredinom dvadesetog veka nezavisno od strane nekoliko istraživača. Semjuel (Arthur Samuel) sa jedne i Edvards (Daniel Edwards), Hart (Timothy Hart) i Levin (Michael Levin) sa druge strane, formulisali su nezavisno ranu verziju algoritma sredinom pedesetih godina. Makarti je slične ideje predstavio 1956. godine, tokom znamenite konferencije u Darmutu. Brudno (Alexander Brudno) je, takođe nezavisno, otkrio Alfa-beta algoritam i objavio ga 1963. godine.

Alfa-beta algoritam zasnovan je na tzv. alfa i beta odsecanju stabla igre i predstavlja heuristikama ubrzan algoritam Minimaks. Osnovni postupak ocenjivanja čvorova je minimaks tipa: funkcijom evaluacije ocenjuju se samo čvorovi na nekoj odabranoj dubini, a zatim se rekursivnim postupkom (minimaksizacijom) ocenjuju čvorovi prethodnici. Postupak „alfa odsecanje“ biće opisan prepostavljući da funkcija evaluacije za igrača  $A$  koji je na potezu ima pozitivan smisao (bolje su veće ocene). Neka je ocenjeno  $n - 1$  od njegovih legalnih poteza,

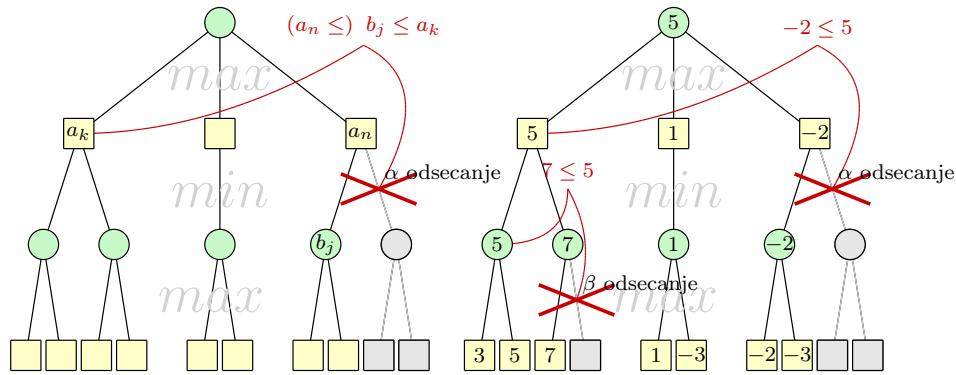


Slika 5.4: Ilustracija rada algoritma Minimaks.

neka su dobijene ocene  $a_1, a_2, \dots, a_{n-1}$  i neka je  $a_k$  najveća od njih. Razmatramo  $n$ -ti legalni potez, neka je  $v$  čvor u koji taj potez vodi i neka je  $a_n$  njegova ocena koja tek treba da bude određena kao minimum ocena dece čvora  $v$  (slika 5.5). Nakon tog poteza, protivnik (igrac  $B$ ) ima više mogućnosti i traži se ona sa najmanjom ocenom. Za ocenu  $b_i$  bilo kog deteta čvora  $v$  važi da je ona veća od ili jednaka zajedničkom minimumu  $a_n$ , tj. važi  $a_n \leq b_i$ . Ako se u pretraživanju čvora  $v$  dođe do čvora sa ocenom  $b_j$  za koju važi  $b_j \leq a_k$ , sigurno je da je i ocena  $a_n$  čvora  $v$  manja od ili jednaka oceni  $a_k$ , jer važi  $a_n \leq b_j \leq a_k$ . Kako se u početnom čvoru traži maksimum ocena mogućih poteza, vrednost  $a_n$ , dakle, ne može da utiče na ocenu početnog čvora. Zato se dalje pretraživanje poteza protivnika u čvoru  $v$  može prekinuti bez uticaja na rezultat pretraživanja — može da se izvrši „alfa odsecanje stabla“ (slika 5.5). „Beta odsecanje“ potpuno je analogno i primenjuje se na čvorove u kojima je na potezu protivnik. Naravno, s obzirom na smisao funkcije evaluacije, maksimumi pominjani u „alfa odsecanju“ zamenjuju se minimumima i obratno (slika 5.5). Algoritam Alfa-beta prikazan je na slici 5.6. U algoritmu, u funkciji Min, vrednost  $\beta$  ažurira se kao tekući minimum. U istoj funkciji, vrednost  $\alpha$  je tekući maksimum roditeljskog čvora. Ako je tekuća ocena  $v$  čvora manja ili jednaka od  $\alpha$ , onda se vrši odsecanje, tj. prekida izvršavanje ove funkcije i vraća  $v$  (ovo radi naredba ako  $v \leq \alpha$  vrati  $v$ ). Naredba ako  $v < \beta$  onda  $\beta := v$  ažurira vrednost  $\beta$  na ovom nivou pretrage. Funkcija Max funkcioniše analogno.

Kako je stablo igre obično ogromno, ubrzavanje Minimaks algoritma heuristikama „alfa-odsecanje“ i „beta-odsecanje“ ima izuzetan značaj. Posebno je važna činjenica da i Alfa-beta algoritam nalazi zaista najbolji mogući potez za zadati čvor i zadatu dubinu pretraživanja, što znači da heuristike koje se primenjuju ne narušavaju tu osobinu algoritma Minimaks.

**Primer 5.2.** Na slici 5.7 prikazan je primer primene algoritma Alfa-beta na šah (pojednostavljenu verziju na tabli  $4 \times 4$ ). Sa  $X$  su označeni delovi stabla igre kod kojih je došlo do odsecanja.



Slika 5.5: Ilustracija primene algoritma Alfa-beta.

Ukoliko se u svakom čvoru potezi ispituju od najlošijeg ka najboljem (u kontekstu tekućeg čvora)<sup>5</sup>, tada nema nijednog alfa ili beta odsecanja, pa se Alfa-beta algoritam svodi na algoritam Minimaks. S druge strane, najviše alfa i beta odsecanja ima kada se najpre ispituju potezi najbolji u kontekstu tekućeg čvora (za korišćenu funkciju evaluacije i za zadatu dubinu) i tada algoritam Alfa-beta daje najbolji efekat (tj. ispituje najmanji broj čvorova stabla). Naravno, nije moguće unapred znati koji je potez najbolji u datom čvoru, ali se i dobrim procenama (izborom jednog od boljih poteza) postižu dobri efekti. Upravo na toj ideji zasnivaju se i neke varijacije Alfa-beta algoritma.

#### 5.4.4 Heuristika kiler

Tokom pretraživanja stabla igre, u svakom čvoru raspoloživi legalni potezi mogu se razmatrati u bilo kom poretku. Međutim, neki poredak može biti bolji od drugog, u smislu da omogućava veći broj alfa i beta odsecanja i time efikasniju pretragu. Heuristika *kiler* (eng. *killer*) ima za cilj da se u mnogim čvorovima razmatra najpre najbolji (ili makar veoma dobar) potez za taj čvor, kako bi bilo što više alfa i beta odsecanja. I ova heuristika je opšta i ne koristi specifična znanja o igri.

Neka se tokom pretraživanja stabla igre algoritmom Alfa-beta ocenjuje neki čvor, prvi na dubini  $d$  i neka je  $p$  pronađeni najbolji potez za taj čvor. Taj potez postaje *kiler potez* za dubinu  $d$ . Kada se sledeći put najde na neki čvor dubine  $d$ , ispitivanje poteza počinje od kiler poteza za tu dubinu.<sup>6</sup> Ukoliko se kasnije pokaže da je za taj nivo bolji neki drugi potez, onda on postaje kiler potez za dubinu  $d$ . Opisana heuristika primenjuje se za sve dubine do maksimalne dubine pretraživanja, tj. za svaku dubinu ažurira se po jedan kiler potez.

Razmotrimo primer ilustrovan na slici 5.8: gornje stablo prikazuje pretraživanje algoritmom Alfa-beta bez kiler heuristike (nije bilo alfa i beta odsecanja), a donje stablo sa primenom kiler heuristike. Tokom ocenjivanja prvog čvora na dubini 1, ocenjivana su njegova deca i kao najbolji pronađen je potez  $p$  (kojem odgovara ocena  $-1$ ). Taj potez postaje *kiler potez* za dubinu 1. U naredna dva čvora na dubini 1, ispitivanje poteza počinje od tog istog poteza (a preostali legalni potezi uređeni su postojećim uređenjem). U oba slučaja dolazi do značajnog broja odsecanja. U četvrtom čvoru na dubini 1 tekući kiler potez  $p$  nije legalan, te ispitivanje poteza počinje od prvog sledećeg legalnog poteza (u nekom postojećem uređenju).

Kiler heuristika zasniva se na sledećem rasuđivanju: ukoliko je u jednoj grani stabla na dubini  $d$  najbolji potez  $p$ , ima izgleda da je on najbolji ili veoma dobar (ako je legalan) i u drugim čvorovima na istoj dubini. Ilustrujmo to na primeru šaha: neka igraču koji je na redu preti mat u sledećem potezu i neka nijedan njegov potez ne može da otkloni tu pretnju. Pretraživanjem stabla, u čvoru u kojem je na potezu protivnik, otkriva se matni potez i on postaje kiler potez (za dubinu 1). Pri daljem pretraživanju stabla, na dubini 1 najpre se ispituje taj potez i kako on vodi pobedu protivnika, alfa odsecanje čini nepotrebnim dalje ispitivanje poteza u tom čvoru. Time se broj čvorova stabla koje u ovakvoj situaciji treba ispitati drastično smanjuje.

Alfa-beta algoritam proširen kiler heuristikom zovemo Alfa-beta/kiler algoritam. Primenom kiler heuristike ne menja se rezultat Alfa-beta algoritma (za istu funkciju evaluacije i istu dubinu pretraživanja) u smislu da se dobija potez sa istom ocenom (ne nužno i isti potez) kao primenom algoritama Alfa-beta ili Minimaks. Pri tome, takav potez dobija se najčešće sa bitno manjim brojem ispitanih čvorova stabla.

<sup>5</sup>Da li je neki potez dobar zavisi od igrača iz čije se perspektive potez razmatra: ono što je dobro za jednog igrača nije za drugog i obratno. Kada se kaže „potez je dobar u kontekstu tekućeg čvora“, to znači da je dobar iz perspektive igrača koji je na potezu u tom čvoru.

<sup>6</sup>Ukoliko u nekom čvoru tekući kiler potez nije legalan, ispitivanje poteza počinje od prvog sledećeg (u postojećem uređenju) poteza.

**Algoritam:** Alfa-beta

**Ulaz:** Funkcija evaluacije  $f$ , pozicija, dozvoljena dubina pretraživanja  $d$

**Izlaz:** Potez

- 1:  $v = \text{Max}(f, \text{pozicija}, -\infty, +\infty, d)$
- 2: vrati potez kojem odgovara vrednost  $v$

**Algoritam:** Max

**Ulaz:** Funkcija evaluacije  $f$ , pozicija, alfa vrednost  $\alpha$ , beta vrednost  $\beta$ , dozvoljena dubina pretraživanja  $d$

**Izlaz:** Vrednost pozicije

- 1: **ako** je pozicija završna ili je  $d$  jednako 0 **onda**
- 2:     vrati  $f(\text{pozicija})$
- 3:  $v := -\infty$
- 4: **za** svaku poziciju  $s$  do koje se može doći u jednom potezu **radi**
- 5:     **ako**  $\text{Min}(f, s, \alpha, \beta, d - 1) > v$  **onda**
- 6:          $v := \text{Min}(f, s, \alpha, \beta, d - 1)$
- 7:     **ako**  $v \geq \beta$  **onda**
- 8:         vrati  $v$
- 9:     **ako**  $v > \alpha$  **onda**
- 10:         $\alpha := v$
- 11: vrati  $v$

**Algoritam:** Min

**Ulaz:** Funkcija evaluacije  $f$ , pozicija, alfa vrednost  $\alpha$ , beta vrednost  $\beta$ , dozvoljena dubina pretraživanja  $d$

**Izlaz:** Vrednost pozicije

- 1: **ako** je pozicija završna ili je  $d$  jednako 0 **onda**
- 2:     vrati  $f(\text{pozicija})$
- 3:  $v := +\infty$
- 4: **za** svaku poziciju  $s$  do koje se može doći u jednom potezu **radi**
- 5:     **ako**  $\text{Max}(f, s, \alpha, \beta, d - 1) < v$  **onda**
- 6:          $v := \text{Max}(f, s, \alpha, \beta, d - 1)$
- 7:     **ako**  $v \leq \alpha$  **onda**
- 8:         vrati  $v$
- 9:     **ako**  $v < \beta$  **onda**
- 10:         $\beta := v$
- 11: vrati  $v$

Slika 5.6: Algoritam Alfa-beta.

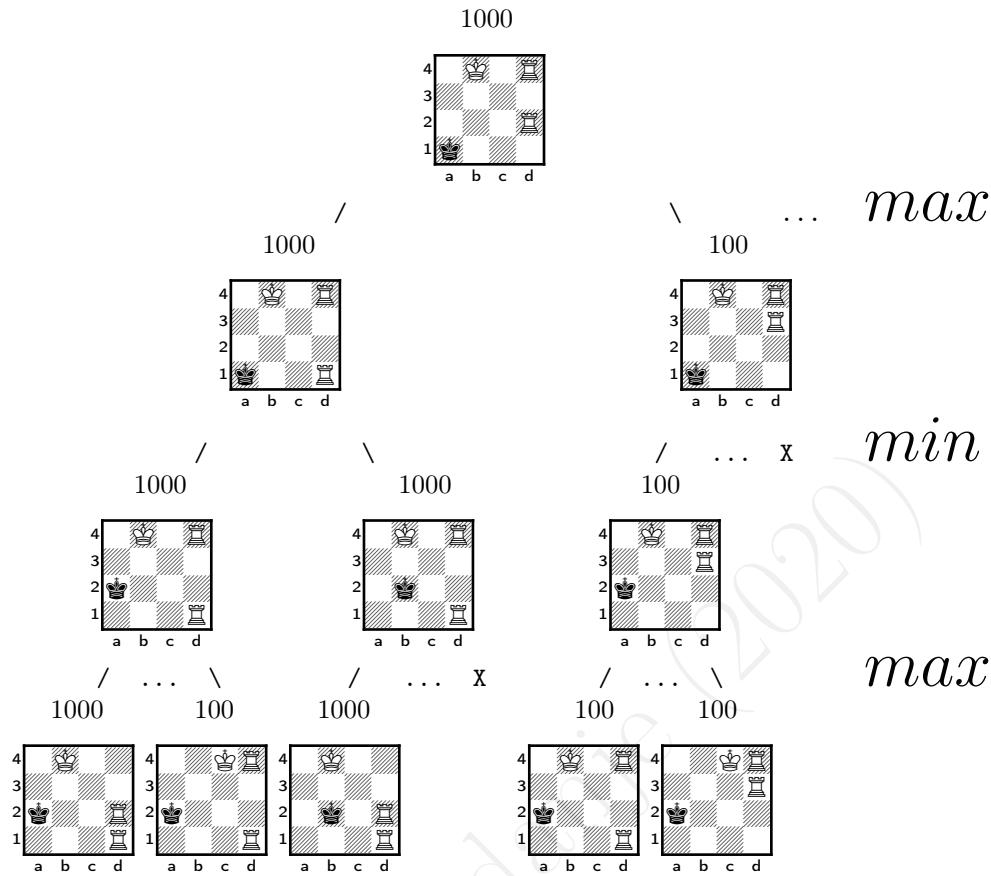
### 5.4.5 Iterativni Alfa-beta/kiler algoritam

U osnovnoj verziji kiler heuristike, kiler potez ne postoji na početnom nivou stabla igre, a upravo taj nivo bi mogao da omogući najveći broj odsecanja. Jedno rešenje za ovaj i još nekoliko drugih problema nudi iterativni Alfa-beta/kiler algoritam.

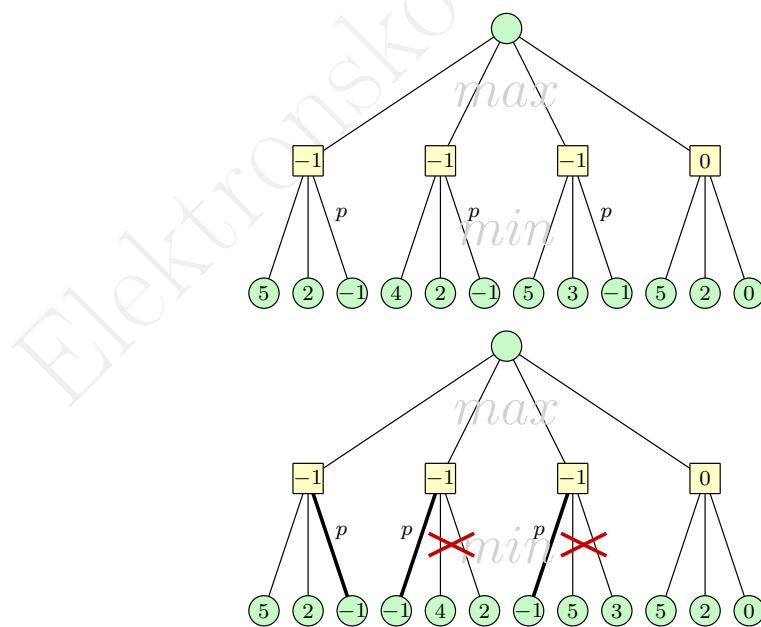
Za zadatu dubinu  $d_{max}$ , iterativni Alfa-beta/kiler algoritam realizuje se na sledeći način: algoritam Alfa-beta/kiler primenjuje se za zadati čvor u iteracijama: za dubinu 1, pa za dubinu 2, i tako dalje, sve do zadate maksimalne dubine  $d_{max}$ . Kad god se završi jedna iteracija, najbolji pronađeni potez postaje kiler potez za početni čvor (za nivo 0) i u sledećoj iteraciji ispitivanje svih legalnih poteza kreće od njega. Finalno, kao najbolji potez za zadati čvor bira se onaj dobijen završnom primenom Alfa-beta/kiler algoritma — primenom za dubinu  $d_{max}$ .

Ako bi se iterativni Alfa-beta/kiler algoritam primenio na stablo prikazano na slici 5.8, u drugoj iteraciji (a možda već i u prvoj) za kiler potez za nivo 0 bio bi odabran četvrti legalni potez (onaj koji vodi oceni 0). U sledećoj iteraciji, ispitivanja poteza na početnom nivou kretaće od tog poteza.

Efekti ovog algoritma su, u svakoj iteraciji, slični efektima Alfa-beta/kiler algoritma, s tim što u iterativnom



Slika 5.7: Ilustracija rada algoritma Alfa-beta.



Slika 5.8: Ilustracija rada algoritma Alfa-beta bez (gore) i sa heuristikom kiler (dole).

algoritmu postoji i kiler potez za početni čvor. Ima izgleda da je u svakoj iteraciji taj kiler potez bolje odabran i da daje sve bolje i bolje rezultate (veći broj alfa i beta odsecanja). Druga dobra osobina iterativnog algoritma je to što u slučaju prekida pretraživanja, praktično u svakom trenutku, ima smisleni rezultat kao najbolji pronađeni potez za neku kompletno završenu iteraciju (videti poglavlje 5.4.7). Ono što izgleda kao manja algoritma —

višestruko pretraživanje nekih čvorova — ne utiče bitno na performanse algoritma. Naime, u odnosu na vreme utrošeno za završnu iteraciju, vreme utrošeno na sve prethodne iteracije praktično je zanemarljivo. Pored toga, s obzirom na (najčešće) dobro odabran kiler potez za početni čvor, završna iteracija obično će zahtevati ispitivanje znatno manjeg broja čvorova od običnog algoritma Alfa-beta/kiler za istu dubinu.

Primenom iterativnog Alfa-beta/kiler algoritma dobija se potez sa najboljom ocenom za datu funkciju evaluacije i datu dubinu pretraživanja, isto kao i primenom algoritama Minimaks, Alfa-beta i Alfa-beta/kiler (možda ne isti potez, ali potez sa istom minimaks ocenom).

#### 5.4.6 Stabilno pretraživanje

Nedostatak pristupa u kojem se pretraživanje stabla igre vrši do fiksne dubine je u tome što funkcija evaluacije, koja se primjenjuje na čvorove na najvećoj dubini, ne razmatra moguće nastavke za pozicije na najvećoj dubini. Te ocene, ma koliko funkcija evaluacije bila dobra, mogu da budu varljive i da vode lošem izboru poteza (u šahu se, na primer, može izabrati potez zbog nekog, naizgled dobrog, završnog čvora u kojem se zarobljava protivnikov top, ali se ne zna da nakon toga može da bude izgubljena kraljica ili da sledi mat). Zbog toga se primjenjuje „stabilno pretraživanje“ (eng. *quiscence searching*): vrši se pretraživanje do neke fiksne dubine, ali se pretraživanje nastavlja i dalje ukoliko je, po nekom kriterijumu, završni čvor „nestabilan“. Maksimalna dubina dodatnog pretraživanja takođe treba da bude ograničena. Stabilno pretraživanje može se primenjivati u kombinaciji sa svakom od ranije opisanih tehnika.

Kriterijumi stabilnosti poteza određuju se u skladu sa specifičnostima konkretnе igre. U šahu, na primer, pozicija se može smatrati stabilnom ukoliko igrač koji je na potezu nije pod šahom, ukoliko ne „visi“ nijedna figura i ukoliko ne preti neposredno izvođenje nekog protivnikovog pešaka. Savremeni programi za šah često pretražuju stablo igre do dubine desetak poteza sa dodatnim, stabilnim pretraživanjem do dubine dvadesetak poteza.

#### 5.4.7 Prekidi i vremenska ograničenja

Vremenska ograničenja su važna u programiranju strateških igara: potrebno je da program izabere smislen potez i ukoliko se pretraživanje stabla igre prekine pre nego što se izvrši kompletan algoritam. Prekidi mogu biti izazvani akcijom korisnika ili ograničenjima vremena raspoloživog za jedan potez ili za celu partiju<sup>7</sup>. Kod do sada opisanih algoritama, ako algoritam nije kompletno izvršen (i, na primer, ispitani su samo neki, povoljni odgovori protivnika), odabrani potez može biti veoma loš. Izuzetak je iterativni Alfa-beta/kiler algoritam, jer praktično u svakom trenutku (prva iteracija traje zanemarljivo) ima neku kompletno završenu iteraciju, te će biti izbegnuti makar najjednostavniji previdi.

#### 5.4.8 Svojstva algoritama Minimaks i Alfa-beta

**Definicija 5.1** (Faktor grananja). *Neka je  $A$  deterministički algoritam za pretraživanje uniformnog stabla igre stepena  $b$  (svi njegovi čvorovi osim listova imaju tačno po  $b$  dece), dubine  $d$  i sa listovima kojima su pridružene vrednosti po raspodeli  $F$ . Ako je sa  $I_A(d, b, F)$  označen očekivani broj završnih čvorova koje algoritam  $A$  ispituje, tada vrednost*

$$R_A(b, F) = \lim_{d \rightarrow \infty} [I_A(d, b, F)]^{1/d}$$

zovemo faktor grananja algoritma  $A$ .

Faktor grananja je ključna karakteristika algoritama za pretraživanje stabla igre jer govori o očekivanom broju završnih čvorova koje nekim algoritmom treba ispitati. Naime, ukoliko je  $R$  faktor grananja nekog algoritma, onda je očekivani broj ispitanih čvorova za dubinu pretraživanja  $d$  jednak  $R^d$ . Na primer, za šahovsku središnjicu procenjuje se da je faktor grananja između 35 i 38, a za igru go oko 250.

Ako se uniformno stablo stepena  $b$  i dubine  $d$  pretražuje Minimaks algoritmom, biće ispitano  $b^d$  završnih čvorova stabla, pa je faktor grananja algoritma Minimaks za svaku raspodelu  $F$  očigledno jednak  $b$ :

$$R_{\text{Minimaks}}(b, F) = b.$$

Ukoliko se u svakom čvoru stabla igre ispituju potezi od najlošijeg ka najboljem, onda, očigledno, nema nikakvih odsecanja i algoritam Alfa-beta se ponaša kao algoritam Minimaks. S druge strane, algoritam Alfa-beta

<sup>7</sup>Ukoliko je ograničeno vreme raspoloživo za celu partiju, program mora i da ga ekonomično deli na procenjeni broj poteza.

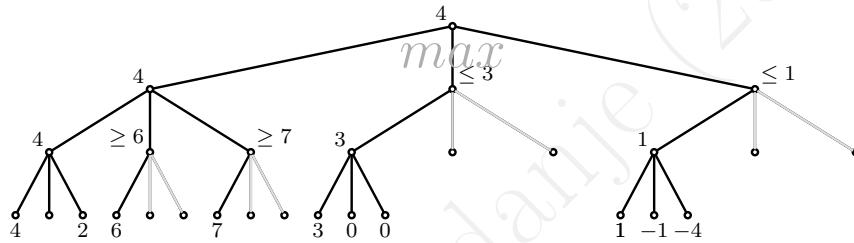
ponaša se najbolje ukoliko se u svakom čvoru najpre ispituje najbolji potez za taj čvor i o tome govori naredna teorema.

**Teorema 5.1.** *Ukoliko je stablo igre uniformno (svaki čvor koji nije list ima tačno  $b$  dece) i ukoliko se u svakom čvoru najpre ispituje najbolji potez za taj čvor, onda algoritam Alfa-beta, primjenjen do fiksne dubine  $d$  ispituje  $O(b^{d/2})$  listova.*

**Dokaz:** Tokom primene algoritma Alfa-beta, nekim čvorovima se izračunava egzaktna minimaks ocena za zadatu dubinu, a nekim čvorovima se ocena računa samo onoliko koliko je neophodno za odsecanja. Na primer, na slici 5.9, za čvorove sa ocenom 4, ova ocena je egzaktna (ista ocena bila bi izračunata i da se uopšte ne primenjuju odsecanja), a čvoru sa ocenom  $\geq 7$ , ocena 7 nije nužno egzaktna (možda je veća), jer je primenjeno odsecanje i nisu ispitani svi njegovi potomci.

Neka  $T(d)$  označava broj listova koje treba ispitati za određivanje egzaktne ocene čvora na nivou  $d$  (za zadato stablo), pri čemu je u ovom konkretnom kontekstu pogodno da se nivoi broje od listova – njima odgovara nivo 0.

Neka  $L(d)$  označava broj listova koje treba ispitati za određivanje (samo) ograničenja ocene čvora na nivou  $d$  dovoljnog da obezbedi odsecanja u nastavku primene algoritma.



Slika 5.9: Ilustracija za broj listova potreban za izračunavanje egzaktne ocene čvora i procene ocene čvora.

Važi  $T(0) = 1$  i  $L(0) = 1$ . Kako se u svakom čvoru najpre ispituje najbolji potez, onda važi (slika 5.9):

$$T(d) = T(d-1) + (b-1)L(d-1)$$

$$L(d) = T(d-1)$$

odakle sledi

$$T(d) = T(d-1) + (b-1)T(d-2)$$

Karakteristična jednačina ove rekurentne veze je  $t^2 - t - (b-1) = 0$  i njena rešenja su

$$t_1 = \frac{(1-\sqrt{1+4(b-1)})}{2} = 1/2 - \sqrt{b-3}/4$$

$$t_2 = \frac{(1+\sqrt{1+4(b-1)})}{2} = 1/2 + \sqrt{b-3}/4$$

Dakle, važi

$$T(d) = \lambda_1 \cdot \left(1/2 - \sqrt{b-3}/4\right)^d + \lambda_2 \cdot \left(1/2 + \sqrt{b-3}/4\right)^d$$

$$\text{i } T(d) = O(\sqrt{b}^d) \text{ tj. } T(d) = O(b^{d/2})$$

□

Navedena teorema govori da algoritam Alfa-beta ispituje barem  $O(b^{d/2})$  listova, tj. njegov faktor granaanja je reda većeg ili jednakog  $O(b^{1/2})$ . Kiler heuristika u iterativnoj verziji često postiže dobar poredak poteza u svakom čvoru, pa ponašanje reda  $O(b^{d/2})$  nije nerealno očekivati. Ovaj rezultat govori i da je (uz dobar poredak poteza), Alfa-beta algoritmom moguće birati poteze analizirajući stablo do dva puta veće dubine nego primenom Minimaks algoritma. Naglasimo da mnoge igre ne ispunjavaju uslov teoreme da je stablo igre uniformno, ali ona ipak daje neku procenu korisnu i u tim slučajevima.

Problem određivanja faktora granaanja Alfa-beta algoritma za funkciju  $F$  sa ravnomernom raspodelom a bez pretpostavke da se najpre ispituje najbolji potez mnogo je složeniji. Odgovor na pitanje o faktoru granaanja Alfa-beta algoritma daje sledeća teorema.<sup>8</sup>

<sup>8</sup>Zapravo, preciznije tvrđenje je da se vrednost  $R_{\alpha-\beta}(b, F)$  može ograničiti odozgo sa  $\xi_b/(1-\xi_b)$ , gde je  $\xi_b$  pozitivno rešenje jednačine  $x^b + x - 1 = 0$ . Vrednost  $\xi_b/(1-\xi_b)$  se za  $b$  takvo da je  $b \leq 1000$  može aproksimirati sa  $0.925 \cdot b^{0.747}$ , što se dalje može ograničiti sa  $b^{3/4}$ .

**Teorema 5.2.** Za faktor grananja Alfa-beta algoritma za stablo stepena  $b$  (koji nije mnogo veliki, tj. za koji je  $b \leq 1000$ ) i za ravnomernu raspodelu  $F$  važi:

$$R_{\alpha-\beta}(b, F) = O(b^{3/4})$$

Važno je i sledeće srođno tvrđenje.

**Teorema 5.3.** Algoritam Alfa-beta je asimptotski optimalan algoritam za pretraživanje stabla igre.

Navedeno tvrđenje znači da ne postoji algoritam za pretraživanje stabla igre koji, u opštem slučaju, asimptotski ispituje manje završnih čvorova nego algoritam Alfa-beta. Modifikacije algoritma Alfa-beta opisane u prethodnom delu veoma često u praksi daju bolje rezultate nego osnovni algoritam. Ipak, faktori grananja ovih algoritama ne razlikuju se od faktora grananja Alfa-beta algoritma i u opštem slučaju oni ne garantuju asimptotski manje ispitanih završnih čvorova nego algoritam Alfa-beta.

## 5.5 Završnica

U igramama kao što je šah, završnica iziskuje specifične tehnike. Naime, pristupi koji se zasnivaju na dubinskom pretraživanju ne daju dobre rezultate u završnici jer kvalitetna igra iziskuje jako veliku dubinu pretraživanja. Problem završnice još je teži ako se postavi zahtev za korektnom ili optimalnom igrom/taktikom<sup>9</sup>. Ukoliko se, tehnikama koje se koriste u središnjici, sistematsko pretraživanje vrši do završnih čvorova time se obezbeđuje optimalna strategija (naravno, za većinu igara to je u praksi nemoguće izvesti). Optimalnu igru u šahovskoj završnici praktično je nemoguće obezbediti (za ubičajena vremenska ograničenja), jer su moguće završnice u kojoj igrač ima dobijenu poziciju, ali ne može da matira protivnika u manje od, na primer, četrdeset poteza<sup>10</sup>, pa takva završnica za optimalnu igru zahteva preveliku dubinu pretraživanja. Slični problemi važe i za korektnu igru.

U nastavku su opisani neki od pristupa koji se koriste u šahovskim završnicama.

**Skupovi pozicija kao klase ekvivalencija.** Bramerov (Max Brammer) opšti algoritam (1975) za završnicu izgleda ovako:

- (a) generiši skup svih legalnih poteza — skup  $Q$ ;
- (b) odaberi najbolje ocenjeni element skupa  $Q$  — element  $q$ ;
- (c) odigraj potez  $q$ .

Ključni korak algoritma (korak (b)) zasnovan je na sledećoj ideji: neka je skup  $Q^*$  skup svih legalnih pozicija u igri i neka je svaka od tih pozicija svrstana u tačno jedan od podskupova koji razlažu  $Q^*$  (to razlaganje definiše se u skladu sa prirodnom konkretnom igre tako da istom skupu pripadaju suštinski slične pozicije – na primer, u šahu, sve pozicije „kralj i pešak protiv kralja“; svakom od tih skupova (odnosno *klasa ekvivalencije*) pridružena je jedinstvena ocena i jedinstvena funkcija evaluacije. Ocenjivanje koje se pominje u koraku (b) Bramerovog algoritma svodi se na sabiranje ocene klase pozicija i ocene pozicije koja toj klasi pripada. Na taj način ocenjuju se svi legalni potezi iz datog čvora i to bez ikakvog pretraživanja preko dubine 1. Kao najbolji bira se potez kojem odgovara najveća zbirna ocena. Opisani algoritam ima brojne varijacije (uključujući varijacije upoređivanja elemenata iz različitih klasa ekvivalencije, provere izabranog poteza pretraživanjem u dubinu itd.).

Opisani algoritam poređenje poteza vrši po ključnim parametrima za konkretnu završnicu (na primer, u šahu, po rastojanju između dva kralja), a ne po opštim kriterijumima koji se koriste u središnjici. Algoritam zahteva kompleksno definisanje mnogih relevantnih klasa pozicija u završnici i pratećih funkcija evaluacije.

<sup>9</sup>Za taktiku kažemo da je *korektna* ukoliko u dobijenoj poziciji sigurno vodi do pobede i ukoliko pri (teorijskoj) remi-poziciji sigurno vodi bar remiju. Za taktiku kažemo da je *optimalna* ukoliko u dobijenoj poziciji vodi pobedi u najmanjem broju poteza, odnosno ukoliko u izgubljenoj poziciji poraz maksimalno odlaže. Očigledno, ako je taktika optimalna, onda je i korektna, ali ne važi obratno.

<sup>10</sup>Lomonosov tabele pokazale su da postoje pozicije u kojima beli dobija, ali ne nužno u manje od (čak) 545 poteza.

**Mali saveti.** Jedan od pristupa koji se primenjuje u šahovskim završnicama je i pristup *malih saveta* (eng. *advice texts*). Navedimo, kao ilustraciju, jedan „mali savet“ za završnicu „kralj i top protiv kralja“ (autora Ivana Bratka).

1. „mat“: pokušaj da matiraš protivnika u dva poteza;
2. „stezanje“: ako to nije moguće, pokušaj da topom smanjiš prostor na tabli dostupan protivničkom kralju;
3. „približavanje“: ako to nije moguće, pronađi način da svog kralja približiš protivničkom;
4. „zadržavanje“: ako to nije moguće, pronađi potez koji задрžava trenutno stanje u smislu (2) i (3) (tj. odaberite „potez čekanja“);
5. „razdvajanje“: ako to nije moguće, pronađi potez kojim se dobija pozicija u kojoj top razdvaja dva kralja, bilo vertikalno ili horizontalno.

Nedostatak pristupa ilustrovanog navedenim primerom je u tome što iziskuje posebne „male savete“ za sve suštinski različite završnice. Pored toga, za sve tipove završnica nije jednostavno (ili nije moguće) napraviti koncizan i efikasan „mali savet“. Prethodnih godina za generisanje ovakvih „malih saveta“ koriste se i tehnike mašinskog učenja.

## 5.6 Monte Karlo pretraga stabla igre

Heuristička pretraga Monte Karlo primenjuje se još od četrdesetih godina prošlog veka u mnogim oblastima, a od 2006. godine često i u pretrazi stabala, kada se naziva *Monte Karlo pretraga stabla* (eng. *Monte Carlo tree search*). To je metod koji u obilasku stabla pretrage koristi slučajno uzorkovanje („semplovanje“). Naime, u situacijama kada nije moguće sprovesti sistematičnu pretragu grafa prostora stanja, simulacijom odigravanja partija obilazi se samo jedan deo odgovarajućeg stabla pretrage. Time se aproksimira njegov obilazak i donose zaključci na osnovu nepotpunih informacija.

Monte Karlo pretraga koristi se decenijama i u strateškim igramama, ali se do najznamenitijih rezultata došlo tek u prethodnih nekoliko godina. Programi AlphaGo i AlphaZero (kompanije DeepMind), koji koriste ovu vrstu pretrage u sadejstvu sa neuronskim mrežama (poglavlje 11.5), pobedili su 2017. godine vodeće svetske igrače i programe u igramu go i šah. Mehanizam koji je u daljem tekstu pojednostavljen opisan, program AlphaZero koristio je za samotreniranje, na nekoliko hiljada procesora, tokom perioda od dvadeset četiri sata. Ishod je superiorna pobeda nad jednim od najboljih šahovskih programa u tom trenutku (Stockfish 8) od 64:36 i osvojena znanja o mnogim otvaranjima za koja su ljudima bili potrebne vekovi. Uprkos ovim uspesima, postoji i uverenje da će najbolji programi za strateške igre u budućnosti (vodeći računa i o raspoloživim resursima) kombinovati Monte Karlo pretragu i neuronske mreže sa algoritmima minimaks tipa opisanim ranije.

Monte Karlo pretraga koristi se u strateškim igramama bez ikakvog domenskog znanja, tj. iskustvenog znanja o konkretnoj igri i jedino što je potrebno to je poznавanje pravila igre, tj. skupa legalnih poteza, završnih stanja i mogućih ishoda igre (na primer: pobeda, remi i poraz). Nepotpuno stablo igre formira se u iteracijama, postepenim popunjavanjem novim čvorovima. U izabranim čvorovima odigravaju se partie simulacijama – do kraja partie, slučajnim odigravanjem poteza, bez ikakvog netrivijalnog rasuđivanja. Ovakvim simulacijama postepeno se profinjuju ocene čvorova stabla (tj. pozicija) i aproksimiraju njihove stvarne vrednosti. Taj postupak nastavlja se sve dok to dozvoljavaju raspoloživi resursi (računarsko vreme i memorija). Svaka iteracija opisanog postupka ima sledeće faze:

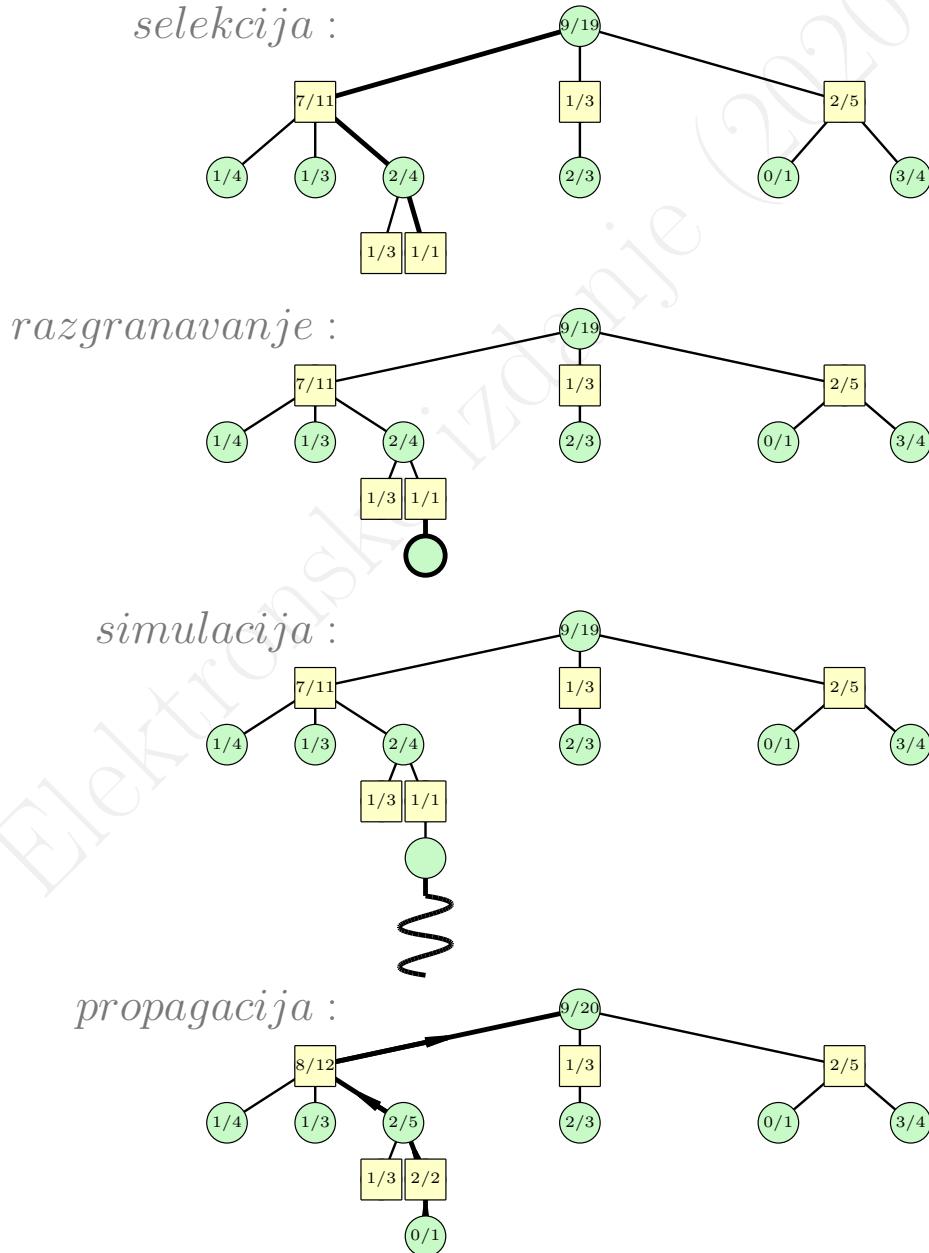
- selekcija: bira se čvor  $M$  koji nije završno stanje igre i iz kojeg se može jednim potezom dobiti pozicija  $m$  koja još uvek nije posećena;
- razgranavanje: čvor sa pozicijom  $m$  pridodaje se stablu i pridružuje mu se ocena 0/0;
- simulacija: pokreće se simulacija nastavka partie iz čvora  $m$  (do završnog stanja) i određuje se ocena te simulacije (na primer 1, ako je u simulaciji pobedio igrač koji je na potezu u čvoru  $m$ );
- propagacija: ažuriraju se ocene svih čvorova od  $m$  do korena novodobijenim rezultatom.

U svakoj iteraciji, dakle, dodaje se po jedan ocenjeni čvor. Simulacija može da se ne sprovodi samo do završnih stanja, već i do nekih drugih predefinisanih stanja (u kojima se jasno procenjuje ishod partie). Dođatno, simulacije mogu da budu obogaćene nekim znanjima kako ne bi bile potpuno slučajne. Umesto jedne simulacije iz novog čvora, može da se izvrši više njih. Ukupan broj simulacija može se smanjiti korišćenjem neuronskih mreža, čime se povećava prostor za temeljniju analizu poteza koji obećavaju više. Korišćenjem neuronskih mreža dobija se i mogućnost generalizacije na stanja koja uopšte nisu viđena.

Načine na koje se bira čvor  $M$  u okviru selekcije zovemo *politikama selekcije*. Politika selekcije često se definiše za jedan nivo stabla i primenjuje rekurzivno, počevši od korena ka listovima. Postoje mnoge politike selekcije. Najjednostavnija je da se čvor za razgranavanje uvek bira slučajno. Još jedna jednostavna politika selekcije je da se sprovodi jednak broj simulacija za svaki čvor na istom nivou stabla. Jedna naprednija politika selekcije zasniva se na ocenama čvorova jednakim

$$\frac{w}{n}$$

gde je  $n$  broj simulacija,  $w$  zbir ishoda svih simulacija iz tog čvora: na primer, vrednosti 1 za pobedu igrača koji je na potezu, 0.5 za remi i 0 za izgubljenu partiju. Primetimo da je, za razliku od minimaks ocena, smisao ocena  $w$  na svim nivoima isti („veće ocene su bolje“). Primetimo i sledeće: ako je ocena nekog čvora  $w/n$ , a  $w_i/n_i$   $i = 1, \dots, k$  su ocene njegove dece, onda važi  $n = \sum_{i=1}^k n_i$  i  $w = n - \sum_{i=1}^k w_i$  i taj odnos se održava kada se ažuriraju ocene čvorova tokom propagacije. Ovakva ocena  $w/n$  aproksimira u toj poziciji verovatnoću pobeđe igrača koji je na potezu. Selekcija se sprovodi tako što se od korena ka listovima uvek bira čvor koji ima najbolju ocenu. Time se obezbeđuje da se bolji potezi detaljnije analiziraju.



Slika 5.10: Ilustracija primene Monte Karlo pretrage stabla igre.

**Primer 5.3.** Na slici 5.10 ilustrovana je jedna iteracija primene Monte Karlo pretrage stabla igre. Neka su mogući ishodi igre  $-1, 0$  i  $1$ . U čvorovima su ocene  $w/n$ , gde je  $n$  broj simulacija,  $w$  zbirni ishod svih simulacija iz tog čvora. U koraku selekcije bira se čvor koji ima ocenu  $1/1$ , on se razgranava – dodaje se jedno njegovo dete i tom detetu se pridružuje ocena  $0/0$ . Zatim se vrši simulacija igre od tog deteta i dobija se da igru gubi igrač koji igra u toj poziciji. Ocena deteta se ažurira na  $0/1$ . Kako gubi igrač koji igra u poziciji u čvoru-detetu, to znači da dobija igrač koji igra u čvoru-roditelju, te se ocena  $1/1$  ažurira na  $2/2$  (što znači da su dobijene dve partije od dve simulacije). Dobijena ocena se, zatim, analogno propagira do samog korena stabla.

Korišćenje ocena čvorova  $\frac{w}{n}$  obezbeđuje da se bolji potezi detaljnije analiziraju, ali potrebno je obezbediti i dodatno analiziranje čvorova sa malim brojem simulacija kako ne bi došlo do previda nekog veoma dobrog poteza. To rade još naprednije politike selekcije koje koriste ocene poput:

$$\frac{w}{n} + \sqrt{\frac{c \ln N}{n}}$$

gde je  $N$  ukupan broj simulacija iz roditeljskog čvora, a  $c$  pogodan parametar koji se bira empirijski (obično je jednak 2).

Stablo koje kreira Monte Karlo metod ne razvija se nužno simetrično. Time se obezbeđuje njegova efikasnost i praktična upotrebljivost, ali u nekim situacijama to može da bude i loše, jer u ograničenom vremenu može da se desi da se metod fokusira na neku lošu granu, previdevši postojanje neke mnogo bolje.

Složenost Monte Karlo pretrage proporcionalna je broju dozvoljenih simulacija. I nakon kratkog utrošenog vremena, metod ima nekakve upotrebljive ocene za raspoložive poteze.

Način rada Monte Karlo pretrage stabla igre suštinski je drugačiji od načina rada algoritama Minimaks i Alfa-beta. Ipak, postoji snažna veza između njih. Pretpostavimo da se algoritam Minimaks primenjuje do listova stabla igre, na primer, do ishoda  $-1, 0, 1$  i da su minimaksizacijom ocenjeni svi čvorovi stabla. Neka su te ocene transformisane u ocene  $0, 1/2, 1$  na sledeći način: neka se na *max* nivoima stabla ocene  $x$  preslikavaju u ocene  $(x+1)/2$ , a na *min* nivoima stabla u ocene  $(-x+1)/2$ . Ocene pozicija zasnovane na Monte Karlo pretrazi, sa opisanom naprednom selekcijom, konvergiraju upravo tim novodobijenim ocenama.



## Glava 6

# Genetski algoritmi

Heuristike koje se koriste u rešavanju problema pretrage dizajnirane su za konkretni problem, imajući u vidu njegove specifičnosti. Heuristika dizajnirana za jedan problem često je potpuno neupotrebljiva za drugi problem i rešavanje svakog novog problema korišćenjem heuristika može da bude veoma zahtevno. S druge strane, *metaheuristike* ili *metaheurističke metode* su metode koje opisuju opšte strategije pretrage za rešavanje optimizacionih problema i formulisane su nezavisno od konkretnog problema. U svom opštem obliku, metaheuristike ne koriste specifičnosti nijednog konkretnog problema i mogu se koristiti za rešavanje široke klase problema. Međutim, iako su metaheuristike opšte metode, one mogu biti prilagođene (kombinacijom internih parametara) specifičnom problemu koji se rešava. Metaheuristike obično razmatraju samo mali uzorak skupa svih mogućih rešenja i obično ne garantuju pronalaženje najboljeg mogućeg rešenja. Međutim, rešenja koja daju metaheuristike često mogu biti dovoljno dobra, posebno u situacijama kada nije raspoloživa ili nije praktično upotrebljiva odgovarajuća egzaktna metoda (koja garantuje pronalaženje najboljeg mogućeg rešenja).

Genetski algoritmi pripadaju široj grupi metaheurističkih algoritama globalne optimizacije ili pretrage koji koriste tehnike inspirisane biologijom. Genetski algoritmi koriste pojmove kao što su selekcija, ukrštanje, nasleđivanje, mutacija, itd. U prirodi, evolucija je proces u kojem jedinke koje su najbolje prilagođene okolini preživljavaju i ostavljaju potomstvo, koje je najčešće isto tako ili bolje prilagođeno okolini. Svaka ćelija svakog živog organizma sadrži hromozome. Svaki hromozom sadrži skup gena — blokove DNK. Svaki gen određuje neku osobinu organizma. Familija gena često se naziva genotip, a familija osobina fenotip. Reprodukcija organizama uključuje kombinovanje gena roditelja i, pored toga, male količine mutacije. Jedinka može biti manje ili više prilagođena okolini. Jedinka koja je bolje prilagođena okolini u kojoj živi ima veću verovatnoću preživljavanja i ostavljanja potomstva, a time i prenošenja svog genetskog materijala. Genetski materijal prilagođenih jedinki uglavnom opstaje, dok genetski materijal neprilagođenih jedinki uglavnom nestaje kroz generacije. Dakle, evolucijski procesi u prirodi su, u određenom smislu, optimizacioni procesi — procesi u kojima se kroz generacije optimizuje genetski materijal (tj. osobine organizama) tako da bude što bolje prilagođen okolini.

Genetski algoritmi mogu se koristiti za nalaženje tačnog ili približnog rešenja nekog problema optimizacije ili pretrage. Mada je još pedesetih godina dvadesetog veka bilo računarskih simulacija zasnovanih na evoluciji, smatra se da je moderne genetske algoritme uveo Džon Holand (John Holland) sedamdesetih godina dvadesetog veka, a postali su popularni kasnih osamdesetih godina. Tokom prethodnih tridesetak godina ostvaren je veliki napredak u razvoju genetskih algoritama. Genetski algoritmi se uspešno primenjuju na širokom skupu problema, često NP-teških ili još težih problema, za koje ne postoji efikasna rešenja. Genetski algoritmi imaju uspešne primene u ekonomiji, tehničkim, bioinformaticim, hemiji, fizici itd. Genetski algoritmi uspešno se primenjuju u mnogim optimizacionim problemima u kojima postoji i više lokalnih ekstremuma. Popularnost genetskih algoritama potiče iz njihove uspešnosti, ali i jednostavnosti. Naime, ideje na kojima su genetski algoritmi zasnovani su jednostavne za razumevanje i implementiranje, a daju opšti sistem pretrage primenljiv na veliki broj problema. Pored toga, i u situacijama kada ne nalaze globalne ekstremume, rešenja koja daju su često dovoljno dobra.

Uporedno sa nalaženjem brojnih novih primena i unapređivanjem algoritma, razvijaju se i teorijske osnove genetskih algoritama, ali još uvek sa ograničenim uspesima. Na primer, iako često nalaze globalne ekstremume, genetski algoritmi ne pružaju informaciju o tome da li je u pitanju globalni ili lokalni ekstremum, niti o tome sa kolikom greškom je određeno rešenje.

## 6.1 Opšti genetski algoritam

Genetski algoritmi implementiraju se kao računarska simulacija u kojoj populacija apstraktno opisanih jedinki koje su kandidati za rešenje problema treba da se približava boljim rešenjima. Reprezentacija jedinke

naziva se *hromozomom* ili *genotipom*. Cilj je naći vrednost za koju zadata *funkcija cilja* dostiže svoj ekstremum ili vrednost koja je dovoljno blizu ekstremuma i rešenje problema može biti numerička vrednost, matematička funkcija, put u grafu itd. Potencijalna rešenja, tj. jedinke obično su predstavljene nizovima nula i jedinica, ali su moguće i druge reprezentacije za probleme u kojima binarna reprezentacija nije pogodna. Postupak se odvija kroz generacije. Početnu generaciju obično čine slučajno generisane jedinke, ali ona može da sadrži i jedinke koje su (grubi) rezultat neke druge, jeftinije optimizacione metode.

Obično u svakoj generaciji postoji isti broj jedinki i za svaku od njih računa se njen kvalitet (koji odgovara prilagođenosti okolini). Funkcija koja pridružuje te vrednosti jedinkama naziva se *funkcija prilagođenosti* ili *funkcija kvaliteta*. Ova funkcija ima ključnu ulogu u algoritmu. Ona može ali ne mora da bude jednaka funkciji cilja.

Iz jedne generacije se, na osnovu vrednosti funkcije prilagođenosti, kroz proces *selekcije* biraju jedinke koje će biti iskorišćene za stvaranje novih jedinki (potomstva). One kvalitetnije biraju se sa većom verovatnoćom. Nad izabranim jedinkama primenjuju se genetski operatori *ukrštanja*<sup>1</sup> i tako se dobijaju nove jedinke. Ukrštanjem se od dve jedinke dobija nova (ili dve nove) sa genetskim materijalom koji je dobijen neposredno od roditelja, tj. od polaznih jedinki. Operatorom *mutacije* može da se modifikuje deo polazne jedinke (i ona oponaša mutacije koje se u prirodi javljaju pod uticajem spoljnih faktora). U svakoj generaciji, dakle, zbog rekombinacije gena mogu da se javljaju brojne sličnosti i različitosti između jedinki te generacije i njihovih potomaka.

*Politika zamene generacija* određuje kako se od postojećih jedinki i njihovog potomstva kreira nova generacija. Neke jedinke u novoj generaciji mogu biti bolje, neke mogu biti i lošije od jedinki iz prethodne generacije, ali se očekuje da se prosečna prilagođenost popravlja. Tako dobijena nova generacija koristi se za sledeću iteraciju algoritma.

Postupak se zaustavlja kada je dostignut zadati broj generacija, kada je dostignut željeni nivo kvaliteta populacije (na primer, prilagođenost najprilagođenije jedinke) ili kada je ispunjen neki drugi uslov. Ukoliko je dostignut zadati broj generacija, nema nikakvih garancija da tekuća najkvalitetnija jedinka ima zadovoljavajuću vrednost funkcije cilja.

Genetski algoritmi se, kao i mnogi drugi algoritmi, dizajniraju za rešavanje neke klase instanci, a ne pojedinačnih instanci. Na primer, algoritam za rešavanje kvadratne jednačine može da reši bilo koju kvadratnu jednačinu, ali je za njegovu primenu potrebno zadati koeficijente koji je u potpunosti određuju. Slično, genetski algoritam za pronalaženje rasporeda časova treba da bude dizajniran tako da rešava različite instance tog problema — za različite škole sa različitim brojem nastavnika, odeljenja, učionica i termina i sa različitim specifičnim zahtevima. To znači da funkcija cilja može da bude definisana tek kad su poznati svi podaci koji precizno zadaju problem. Zbog toga se može smatrati da ulaz za opšti algoritam čini opis problema na osnovu kojeg tek treba definisati funkciju cilja, ali i brojna podešavanja algoritma (tj. vrednosti njegovih parametara) pogodna za konkretan problem. Međutim, upravo definisanje funkcije prilagođenosti i izbor pogodnih parametara često čine najteži deo primene genetskih algoritama, tj. primena opšteg genetskog algoritma je obično samo mali deo potrebnog truda.

Opšti genetski algoritam prikazan je na slici 6.1.

### Algoritam: Opšti genetski algoritam

**Ulaz:** Podaci koji određuju funkciju cilja i podešavanja algoritma

**Izlaz:** Najkvalitetnija jedinka u tekućoj populaciji

- 1: generiši početnu populaciju jedinki;
- 2: izračunaj prilagođenost svake jedinke u populaciji;
- 3: **ponavljam**
- 4: izaberi iz populacije skup jedinki za reprodukciju;
- 5: primenom operatora ukrštanja i mutacije kreiraj nove jedinke (i računaj njihovu prilagođenost);
- 6: na osnovu starih i novih jedinki, kreiraj novu generaciju;
- 7: **dok nije ispunjen** uslov zaustavljanja
- 8: vrati najkvalitetniju jedinku u poslednjoj populaciji

Slika 6.1: Opšti genetski algoritam.

Pored podataka koji određuju funkciju cilja, da bi navedeni opšti algoritam bio upotrebljiv potrebno je

<sup>1</sup>Ovaj termin nije sasvim u skladu sa značenjem koje ima u biologiji.

izabrati i podešavanja algoritma za konkretni problem – definisati reprezentaciju jedinki, funkciju prilagođenosti, politiku selekcije, politiku zamene generacija, itd.

## 6.2 Komponente genetskog algoritma

Svaki genetski algoritam ima nekoliko komponenti koje moraju biti zadate, kao što je reprezentacija jedinki, proces selekcije, politika zamene generacija itd.

### 6.2.1 Reprezentacija jedinki

Jedinke mogu biti predstavljene raznovrsnim strukturama podataka, na primer, nizovima binarnih cifara, stablima, matricama i drugim. Neophodno je da izabrana reprezentacija može da opiše moguće rešenje razmatranog problema i da se nad njom definišu genetski operatori (ukrštanje i mutacija). Takođe, odabrana reprezentacija bitno utiče na performanse algoritma. Poželjno je da genetski operatori budu definisani tako da se njima ne dobijaju jedinke koje ne predstavljaju moguća rešenja (na primer, nelegalni putevi u grafu), jer bi one narušavale performanse algoritma. Međutim, nekada se koriste i takvi operatori, ali se onda moraju definisati mehanizmi popravljanja jedinki, tako da odgovaraju mogućim rešenjima.

Najčešće korišćena reprezentacija jedinki je u vidu nizova binarnih cifara (analogno nizovima bitova u digitalnim računarima). Svaki deo hromozoma, tj. svaku cifru u takvoj reprezentaciji, zovemo *gen*. Dublja priroda binarne reprezentacije zavisi od konkretnog problema. Na primer, ako je dužina hromozoma  $n$  binarnih cifara (tj. bitova) i ako je prostor mogućih rešenja interval realnih brojeva  $[a, b]$ , onda je potrebno uspostaviti vezu (koja, naravno, nije bijektivna) između nizova  $n$  binarnih cifara i realnih brojeva iz datog intervala. Tako će binarna reprezentacija  $\underbrace{000\dots0}_n$  odgovarati broju  $a$ , a binarna reprezentacija  $\underbrace{111\dots1}_n$  broju  $b$ . Broju  $x$  sa binarnom reprezentacijom između  $\underbrace{000\dots0}_n$  i  $\underbrace{111\dots1}_n$  odgovara realni broj

$$a + \frac{x}{2^n - 1}(b - a)$$

S druge strane, realnom broju  $x$  iz intervala  $[a, b]$  pridružujemo niz binarnih cifara koji predstavlja binarnu reprezentaciju broja

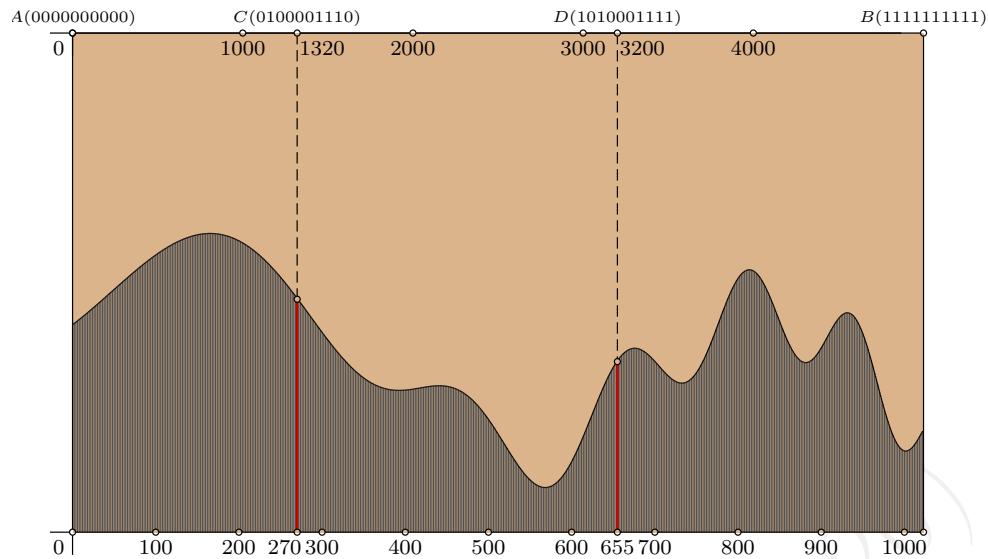
$$\left[ \frac{x - a}{b - a} (2^n - 1) \right].$$

**Primer 6.1.** Prepostavimo da naftnu platformu treba postaviti na pogodnom mestu na putu između tačaka  $A$  i  $B$ , koji je dužine 5000m. Lokacija platforme je pogodnija ukoliko na tom mestu postoje veće rezerve nafte. U biranju lokacije platforme moguće je meriti postojeće rezerve nafte na bilo kojoj tački između  $A$  i  $B$ . Moguća rešenja mogu se predstaviti nizovima bitova dužine 10, tj. brojevima od 0 do 1023. Tački  $A$  tada odgovara broj 0 i reprezentacija 0000000000, a tački  $B$  broj 1023 i reprezentacija 1111111111. Tački  $C$  na rastojanju 1320m od tačke  $A$  odgovara vrednost  $1023 \cdot (1320/5000) \approx 270$  i reprezentacija 0100001110, a tački  $D$  na rastojanju 3200m od tačke  $A$  odgovara vrednost  $1023 \cdot (3200/5000) \approx 655$  i reprezentacija 1010001111. Za vrednost funkcije prilagođenosti jedne tačke može se uzeti rezerva nafte izmerena u toj tački (slika 6.2).

### 6.2.2 Funkcija prilagođenosti

Funkcija prilagođenosti daje ocenu kvaliteta jedinke. Što je vrednost funkcije prilagođenosti za neku jedinku veća, to će biti veća i verovatnoća da se ta jedinka koristi za generisanje sledeće generacije. Očekuje se da, kroz generacije, ukupna prilagođenost bude sve bolja i bolja. Pogodan izbor funkcije prilagođenosti od izuzetne je važnosti za efikasnost algoritma.

Funkcija prilagođenosti treba da je definisana za sve moguće jedinke i da se relativno brzo izračunava. Sem ovih, ne postoje nikakvi opšti uslovi koje funkcija prilagođenosti treba da zadovoljava (na primer, da je diferencijabilna), mada je algoritam često efikasniji za funkcije koje zadovoljavaju neke specifične uslove. Često je pogodno da funkcija prilagođenosti bude nenegativna (na primer, zbog jednostavnijeg sprovođenja procesa selekcije, videti poglavlje 6.2.4).



Slika 6.2: Reprezentacija jedinki u problemu lokacije naftne platforme.

**Primer 6.2.** Potrebno je odrediti maksimum funkcije  $f(x)$  na intervalu  $[a, b]$ . Funkcija je definisana za sve elemente datog intervala, ali nije nužno ni neprekidna, ni diferencijabilna. Genetski algoritam moguće je primeniti tako da se za funkciju prilagođenosti koristi upravo funkcija  $f$ , a da se za reprezentaciju koristi binarna reprezentacija (na način opisan u poglavљу 6.2.1). Ukoliko je potrebno odrediti minimum funkcije  $f(x)$ , onda bi za funkciju prilagođenosti mogla da se koristi funkcija  $-f$ .

**Primer 6.3.** Zadatak je napraviti raspored časova za neki fakultet. Potrebno je da raspored zadovoljava uslove koji se odnose na studentske grupe, na nastavnike, na raspoložive sale, itd. Genetski algoritam moguće je primeniti tako da se za funkciju prilagođenosti koristi broj zadovoljenih uslova. Ukoliko su neki uslovi važniji od drugih, u funkciji prilagođenosti oni mogu da imaju veće težinske faktore.

Ukoliko je, na primer, zadatak odrediti maksimum neke eksplicitno zadate funkcije  $f$ , onda je prirođan izbor za funkciju prilagođenosti sâma ta funkcija  $f$ . Funkcija cilja i funkcija prilagođenosti, međutim, ne moraju da budu iste. Na primer, funkcija prilagođenosti može jedinkama čija vrednost funkcije cilja prelazi neku granicu pridruživati vrednost 1, a ostalima vrednost 0. Tako se može pojednostaviti implementacija algoritma, ali se ovakve odluke moraju donositi oprezno kako ne bi došlo do smanjenja raznolikosti populacije. U nekim problemima, funkcija prilagođenosti je samo jednostavno transformisana funkcija cilja: na primer, ako se traži maksimum neke funkcije  $f$  koja ima i pozitivne i negativne vrednosti, za funkciju prilagođenosti, kako bi bila nenegativna, može se uzeti funkcija cilja uvećana za  $C$ , gde je  $C$  konstanta dovoljno velika da je taj zbir uvek nenegativan. Ponekad je i skoro nemoguće da funkcija bude identična funkciji cilja. Naime, funkcija cilja ne mora uvek biti eksplicitno zadata nekom matematičkom reprezentacijom, već samo implicitno ili nekakvим manje formalnim opisom.

### 6.2.3 Inicijalizacija

Populaciju jedinki jedne generacije, ukoliko se koristi binarna reprezentacija, čini skup nizova binarnih cifara. U toku rešavanja jednog problema, obično sve generacije imaju isti broj jedinki. Taj broj, veličina populacije, je parametar algoritma i on je obično nekoliko desetina, stotina ili, eventualno, nekoliko hiljada.

Proces inicijalizacije, tj. proces generisanja početne populacije, često je jednostavan. Najčešće se početna populacija generiše slučajno (tako da pokriva čitav prostor pretrage). Ukoliko se koristi binarna reprezentacija, jedinke početne generacije mogu se generisati kao slučajni broevi u intervalu  $[0, 2^n - 1]$ , gde je  $n$  dužina hromozoma u izabranoj reprezentaciji. Dodatno, u početnu populaciju mogu biti dodate neke specifične jedinke (na primer, iz delova prostora pretrage za koje se veruje da sadrži optimalna rešenja) ili čitava početna populacija može biti generisana koristeći neki drugi optimizacioni metod. U nekim problemima može da postoji ograničenje nad potencijalnim rešenjima, tj. jedinkama i njega onda treba uzeti u obzir pri generisanju slučajnih jedinki.

Slično, i u kasnijim fazama algoritma treba voditi računa o neispravnim jedinkama, koje su se pojavile u populaciji, a po formi ne ispunjavaju uslove koje potencijalna rešenja moraju da ispune. Takve jedinke obično se koriguju unapred definisanim postupcima.

#### 6.2.4 Selekcija

Selekcija obezbeđuje čuvanje i prenošenje dobrih osobina populacije (tj. dobrog genetskog materijala) na sledeću generaciju. U svakoj generaciji, deo jedinki se izdvaja za reprodukciju i generisanje nove generacije. Izdvajanje jedinki koje će učestovavati u reprodukciji zasniva se na funkciji prilagođenosti i, generalno, prilagođenje jedinke imaju veću verovatnoću da imaju potomstvo. U najjednostavnijim pristupima biraju se jedinke sa najvećom vrednošću funkcije prilagođenosti. U drugim pristupima, jedinke se biraju slučajno, ali sa verovatnoćama koje su izvedene iz prilagođenosti, pri čemu je moguće da budu izabrane i neke lošije prilagođene jedinke (to može da pomogne u održavanju genetske raznolikosti i, dalje, u sprečavanju prerane konvergencije ka nekom lokalnom optimumu). Najpopularnije strategije selekcije su ruletska i turnirska selekcija.

**Ruletska selekcija** Ruletska selekcija (eng. *roulette wheel selection*) je proces selekcije u kojem veće šanse da učestvuju u reprodukciji imaju prilagođenije jedinke.

Ako je  $f(i)$  vrednost funkcije prilagođenosti za jedinku  $i$ , a  $N$  broj jedinki u populaciji, verovatnoća da će jedinka  $i$  biti izabrana da učestvuje u reprodukciji jednaka je

$$p_i = \frac{f(i)}{\sum_{j=1}^N f(j)}$$

Naziv ruletske selekcije potiče od analogije koja se može napraviti sa ruletom. Ukoliko polja ruleta imaju širine proporcionalne verovatnoćama jedinki populacije, onda je proces biranja  $m$  jedinki za reprodukciju analogn odigravanju  $m$  partija ruleta.

**Primer 6.4.** Prepostavimo da populacija ima osam jedinki:  $a, b, c, d, e, f, g, h$  i da su njihove prilagođenosti redom  $0.10, 0.30, 0.06, 0.10, 0.40, 0.24, 0.60, 0.20$ . Ukupna prilagođenost generacije jednaka je  $2.00$ . Sledеća tabela prikazuje verovatnoće izbora jedinki u ruletskoj selekciji:

jedinka	$a$	$b$	$c$	$d$	$e$	$f$	$g$	$h$
prilagođenost	0.10	0.30	0.06	0.10	0.40	0.24	0.60	0.20
verovatnoća izbora	0.05	0.15	0.03	0.05	0.20	0.12	0.30	0.10

Slika 6.3 ilustruje, u formi ruleta, verovatnoće izbora koje su pridružene jedinkama.

U opisanom pristupu, prepostavlja se da je funkcija prilagođenosti definisana tako da ima samo nenegativne vrednosti.

U ruletskoj selekciji moguće je da jedna jedinka više puta bude izabrana da učestvuje u reprodukciji. Prevelik broj ponavljanja istih jedinki loše utiče na performanse algoritma.

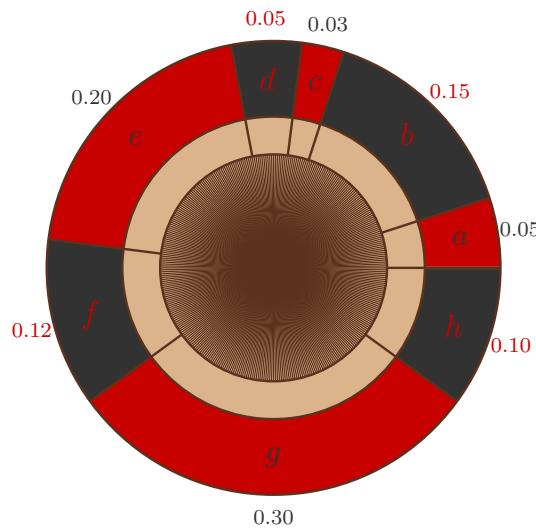
**Turnirska selekcija** U turnirskoj selekciji (eng. *tournament selection*), jedinke „odigravaju turnire“ u kojima veće šanse za pobedu (tj. za prelazak u narednu generaciju) imaju one sa boljom prilagođenošću.

Za jedan turnir bira se slučajno  $k$  jedinki iz populacije, gde je  $k$  parametar procesa turnirske selekcije. Nakon toga, u jednoj varijanti turnirske selekcije, pobednikom se smatra jedinka sa najvećom prilagođenošću. U drugoj varijanti, izabranih  $k$  jedinki sortira se po vrednosti funkcije prilagođenosti i  $i$ -ta jedinka u tako sortiranom nizu bira se sa verovatnoćom<sup>2</sup>  $p(1 - p)^{i-1}$ , gde je verovatnoća  $p$  drugi parametar procesa turnirske selekcije. Postoje i mnoge druge varijante turnirske selekcije.

Ukoliko se u procesu selekcije koristi veća veličina turnira, onda nekvalitetne jedinke imaju manje šanse da budu izabrane. Selekcija sa veličinom turnira 1 ekvivalentna je slučajnoj selekciji. U determinističkoj turnirskoj selekciji ( $p = 1$ ) bira se najbolja jedinka u svakom turniru.

Jedinkama koje su jednom izabrane može se zabraniti učestvovanje u daljim turnirima.

<sup>2</sup> Zbir ovih verovatnoća za svaku konačnu vrednost  $k$  manji je od 1. Zato se one mogu skalirati tako da im zbir bude jednak 1 ili neki turniri mogu da ne daju pobednika.

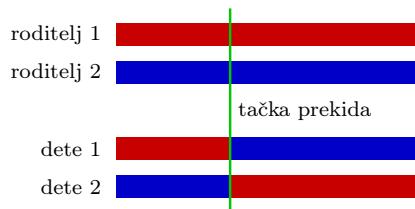


Slika 6.3: Ilustracija ruletske selekcije.

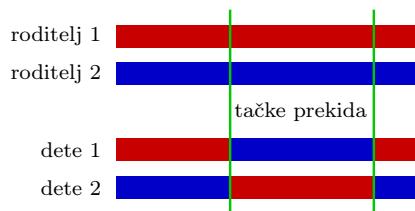
### 6.2.5 Reprodukcija

U procesu reprodukcije učestvuju jedinke koje su izabrane u procesu selekcije: biraju se dve različite i ukrštaju sa zadatom verovatnoćom (obično između 0.6 i 0.9). Dve jedinke koje učestvuju u ukrštanju (eng. *crossover*) nazivaju se roditelji. Rezultat ukrštanja je jedna nova jedinka ili dve nove jedinke koje nazivamo decom ili neposrednim potomcima. Očekivano je da deca nasleđuju osobine roditelja, pa je otuda i nada da deca mogu da imaju prilagođenost kao svoji roditelji ili bolju (ako su nekako objedinila različite osobine koje daju dobru prilagođenost). Dobijeni potomci imaju šansu da uđu u sledeću generaciju, u skladu sa politikom zamene generacija.

Postoji više jednostavnih varijanti ukrštanja kada se koristi binarna reprezentacija. U jednoj varijanti (*višepoziciono ukrštanje*) dovoljno je izabratи tačke ukrštanja i prekombinovati nizove binarnih cifara — jedno deo deo od jedne tačke prekida do sledeće nasleđuje od jednog roditelja, a naredni deo od drugog. Ukrštanje može koristiti proizvoljan broj tačaka prekida (s tim da je manji od dužine hromozoma). Slike 6.4 i 6.5 ilustruju ukrštanje sa jednom (jednopoziciono ukrštanje) i sa dve tačke ukrštanja (dvopoziciono ukrštanje) za binarnu reprezentaciju. Tačke prekida biraju se slučajno iz skupa svih mogućih tačaka prekida.



Slika 6.4: Jednopoziciono ukrštanje.



Slika 6.5: Dvopoziciono ukrštanje.

Uniformno ukrštanje daje dva deteta. Kod ovog ukrštanja svaka binarna cifra prvog roditelja se sa verovatnoćom  $p$  prenosi na prvo dete i sa verovatnoćom  $1 - p$  na drugo dete (pri čemu dete koje nije izabrano nasleđuje binarnu cifru drugog roditelja). Verovatnoća  $p$  je obično jednaka 0.5, ali može biti i drugačija.

### 6.2.6 Mutacija

Mutacija se primenjuje nakon procesa ukrštanja. To je operator koji sa određenom (obično veoma malom) verovatnoćom menja jedan deo jedinke na određeni način. Na primer, u binarnoj reprezentaciji mutacija menja jedan ili više slučajno odabralih gena. Od jedne jedinke dobija se jedna nova jedinka. Verovatnoća da će neka binarna cifra neke jedinke populacije biti promenjena je parametar algoritma i određuje se eksperimentalno (a obično je manja od 1%).

Uloga mutacija u genetskim algoritmima je da spreči da jedinke u populaciji postanu sviše slične i da pomogne u obnavljanju izgubljenog genetskog materijala. Na primer, ukoliko u jednoj generaciji sve jedinke imaju istu vrednost jednog gena, onda taj gen samo ukrštanjem nikada ne bi mogao da se promeni. Kontrolisano podsticanje genetske raznolikosti mutacijom često omogućava izbegavanje lokalnih ekstremuma. Mutacije, naime, omogućavaju razmatranje novih delova prostora pretrage u nadi da će se naići na globalni ekstremum. Dovoljno je da se jedna jedinka približi globalnom (ili nekom lokalnom) ekstremumu, pa da za nekoliko generacija sve jedinke budu u tom delu prostora pretrage.

Ukoliko je verovatnoća mutacije velika, onda usmeravanje pretrage postaje preslabo i ona počinje da liči na slučajnu pretragu. Ukoliko je verovatnoća mutacije jednaka nuli, onda uopšte nema mutacije i algoritam će verovatno brzo dospeti do nekog lokalnog ekstremuma.

### 6.2.7 Politika zamene generacije

Politika zamene generacije opisuje kako se od tekuće generacije dobija nova. Osnovna podela po ovom kriterijumu je na *generacijske* genetske algoritme (eng. *generational genetic algorithm*) i genetske algoritme *stabilnog stanja* (eng. *steady state genetic algorithm*).

U slučaju generacijskih genetskih algoritama, nova generacija dobija se tako što se selekcijom bira dovoljno jedinki iz tekuće generacije da se napravi cela nova generacija. Izabrane jedinke se ukrštaju i mutiraju i tako dobijena generacija zamenjuje staru.

U slučaju genetskih algoritama stabilnog stanja, čim se izabere par roditelja, vrše se ukrštanje i mutacija i umetanje potomaka u populaciju u skladu sa nekom politikom zamene. Postoje raznovrsne politike zamene a neke od njih su:

- *zamena najgorih*, prema kojoj dobijeni potomci zamenjuju najmanje prilagođene jedinke u populaciji;
- *nasumična zamena*, prema kojoj dobijeni potomci zamenjuju nasumično izabrane jedinke iz populacije;
- *takmičenje roditelja i potomaka*, prema kojoj dobijeni potomci zamenjuju svoje roditelje ukoliko su od njih bolji;
- *turnirska zamena*, prema kojoj se jedinka koju dobijeni potomci zamenjuju bira istim mehanizmom kao kod turnirske selekcije, s tim što se umesto najbolje prilagođenih jedinki biraju najgore.

Pored navedenih, za genetske algoritme stabilnog stanja, postoje i druge strategije zamene.

*Elitizam* je (opcionala) strategija u okviru zamene generacije kojom se nekoliko najboljih jedinki (možda samo jedna) u generaciji štite od eliminisanja ili bilo kakvih izmena i takve prenose u sledeću generaciju. Ovim se eliminiše opasnost da se neka posebno kvalitetna jedinka izgubi tokom evolucionog procesa. Elitizam može da se koristi i u generacijskim politikama i u politikama stabilnog stanja.

### 6.2.8 Zaustavljanje

Genetski algoritam se izvršava, tj. evolucijski proces stvaranja novih generacija se ponavlja, sve dok nije zadovoljen neki uslov zaustavljanja. Najčešće se koriste sledeći uslovi zaustavljanja:

- pronađeno je rešenje koje zadovoljava unapred zadati kriterijum;
- dostignut je zadati broj generacija;
- funkcija prilagođenosti izračunata je zadati broj puta;
- vrednost prilagođenosti najbolje jedinke se tokom određenog broja generacija nije popravila;
- kombinacija nekoliko uslova.

### 6.3 Svojstva genetskih algoritama

Genetski algoritmi imaju širok domen i uspešno se primenjuju na velikom broju optimizacionih problema, često onih koji su NP-teški ili još teži. S druge strane, još uvek nema mnogo teorijskih rezultata koji govore o svojstvima genetskih algoritama, o kvalitetu rešenja koja daju, pa čak ni o tome zašto su genetski algoritmi uspešni. U daljem tekstu, biće reči o nekim dobrim i lošim stranama genetskih algoritama.

**Ciljna funkcija.** Ciljna funkcija može biti potpuno proizvoljna i ne mora da zadovoljava nikakve uslove (na primer, da bude neprekidna ili diferencijabilna). Međutim, u primenama u veštačkoj inteligenciji (na primer, u kretanju robota), ciljna funkcija često nije zadata eksplisitno već implicitno, kroz veći broj kriterijuma.

**Reprezentacija jedinki, funkcija prilagođenosti i operatori.** Pogodan izbor reprezentacije jedinki, funkcije prilagođenosti i operatora ukrštanja obično je ključan za performanse algoritma (brzina dolaženja do rešenja i kvalitet rešenja). Ipak, za mnoge optimizacione probleme nije lako konstruisati pogodnu funkciju prilagođenosti jer se obično ne može unapred oceniti da li je nešto rešenje ili nije. U prvoj fazi rešavanja, reprezentacija jedinki, funkcija prilagođenosti i operatori se prilagođavaju problemu, a onda se vrši i prilagođavanje parametara algoritma, kao i dodatno fino podešavanje procesa rešavanja.

**Parametri algoritma.** Pogodan izbor operatora ukrštanja i parametara genetskog algoritma (veličina populacije, verovatnoća ukrštanja, verovatnoća mutacije, itd) veoma je važan za njegove performanse. S druge strane, upravo velika sloboda u izboru parametara istovremeno je i pretnja da mogu da budu korišćeni parametri koji daju loše performanse. Optimizovanje parametara genetskog algoritma je kompleksan problem koji se najčešće rešava izvođenjem eksperimenta – probnih rešavanja. Za izbor pogodnih parametara često se koriste sâmi genetski algoritmi. Parametri genetskog algoritma ne moraju biti fiksirani, već mogu da se menjaju i prilagođavaju tokom rada. Na primer, ukoliko su tekuće jedinke raznolike, onda se može povećati verovatnoća ukrštanja, a smanjiti mutacija, a ukoliko su slične, onda se može uraditi obratno, kako bi se povećale šanse za bekstvo jedinki iz lokalnog optimuma.

**Domen genetskih algoritama.** Genetski algoritmi primenljivi su na veoma širok skup problema. Ipak, za uspešno rešavanje konkretnih problema potrebno je napraviti mnogo dobrih izbora (na primer, za funkciju prilagođenosti i za parametre).

**Kvalitet rešenja.** Genetski algoritam ne daje garanciju da je pronađeno rešenje globalni optimum. Štaviše, genetski algoritmi često imaju tendenciju da idu ka lokalnim optimumima, pošto je pronađenje globalnog optimuma teško. Međutim, i ako nije nađeno rešenje koje je globalni optimum, često je rešenje koje je nađeno dovoljno dobro. Dodatno, kao rezultat algoritma može se ponuditi neki skup najboljih pronađenih jedinki, što je često veoma pogodno. Takvo ponašanje je zadovoljavajuće, posebno u problemima za koje ne postoje tehnikе koje garantuju pronađenje optimalnog rešenja.

**Zahtevani resursi.** Genetski algoritmi se jednostavno implementiraju. Ipak, za najbolje rezultate često je potrebno implementaciju prilagoditi konkretnom problemu. Iako su algoritmi i implementacije obično jednostavni, izvršavanje genetskih algoritama često je veoma vremenski i memorijski zahtevno. Genetski algoritmi mogu se pogodno i efikasno paralelizovati.

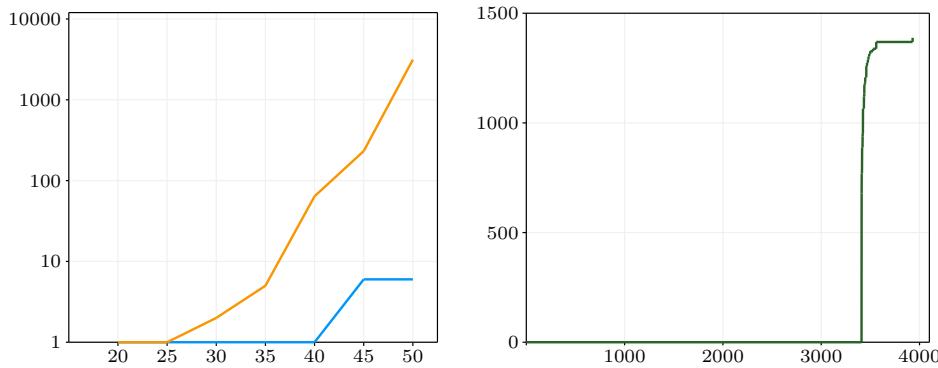
### 6.4 Primeri primena genetskih algoritama

U ovom poglavlju biće data tri konkretna, jednostavna, ali ilustrativna primera primene genetskih algoritama.

#### 6.4.1 Rešavanje problema ranca

Genetski algoritmi često se koriste za rešavanje teških kombinatornih problema. Jedan takav problem je dobro poznati problem ranaca: dat je skup predmeta, svaki sa nekom poznatom vrednošću i poznatom masom; treba pronaći podskup predmeta koje treba staviti u dati ranac, tako da njihova masa ne pređe nosivost ranca a da ukupna vrednost predmeta u rancu bude maksimalna moguća. Ovde ćemo razmatrati varijantu 0-1 problema ranca u kojoj se predmeti mogu samo u celosti staviti u ranac (nije ih moguće deliti na delove). Ovaj problem je NP-težak, što znači da sa povećanjem dimenzije problema egzaktni algoritmi brzo postaju praktično neupotrebljivi, te je kao takav zanimljiv za primenu genetskih algoritama.

Prepostavlja se postojanje  $n$  predmeta, vrednosti  $v_1, \dots, v_n$  i masa  $m_1, \dots, m_n$ . Na raspolažanju je ranac nosivosti  $M$ . Ako su  $x_1, \dots, x_n$  binarne promenljive koje označavaju da li je odgovarajući predmet stavljen u



Slika 6.6: Na levoj slici prikazano je vreme izvršavanja egzaktnog algoritma (narandžasto) i genetskog algoritma (plavo) u odnosu na broj predmeta u problemu ranca. Vremenska skala je logaritamska. Na desnoj slici prikazana je vrednost najboljeg rešenja kroz generacije do dostizanja optimalnog rešenja za dimenziju 50.

ranac ili ne i  $x$  binarni vektor čije su one koordinate, problem ranca može se definisati kao sledeći problem maksimizacije:

$$\max_{\mathbf{x} \in \{0,1\}^n} \sum_{i=1}^n v_i x_i$$

$$\text{tako da } \sum_{i=1}^n m_i x_i \leq M$$

U slučaju ovog problema, lako je definisati osnovne elemente genetskog algoritma. Za reprezentaciju rešenja, odnosno hromozom, prirodno se nameće sâm binarni vektor  $\mathbf{x}$ . Za funkciju prilagođenosti može se uzeti ukupna vrednost izabranih predmeta ukoliko njihova ukupna masa ne premašuje nosivost ranca, a 0 u suprotnom. Za selekciju ćemo izabrati turnirsku, iako su mogući i drugačiji izbori. Za ukrštanje je moguće izabrati bilo koji od prethodno opisanih načina. U nastavku se pretpostavlja da se radi o jednopozicionom ukrštanju. Mutacija se takođe vrši na uobičajen način. Politika zamene populacije je generacijska.

Kako bismo demonstrirali upotrebljivost genetskog algoritma, izvršili smo sledeći eksperiment. Nosivost ranca je fiksirana na 500. Razmatrano je nekoliko instanci problema ranca, koje su generisane nasumice: za svaku dimenziju problema  $n$ , vrednosti predmeta i njihove mase izabrane su nasumice iz intervala  $[0, 100]$ .

Genetski algoritam konfigurisan je tako da je veličina populacije jednaka 1000, verovatnoća ukrštanja 0.8, a da je verovatnoća mutacije 0.01. Jedinke su inicijalizovane tako što su im svi bitovi nasumično generisani sa verovatnoćom 0.5 za oba ishoda.

Optimalno rešenje za svaku dimenziju pronađeno je jednostavnim egzaktnim algoritmom (BFS sa bektrekingom), pri čemu je mereno i njegovo vreme izvršavanja. Za genetski algoritam mereno je vreme izvršavanja do pronalaženja optimalnog rešenja, u čemu je genetski uspeo u svakom od eksperimenata. Primetimo da za razliku od egzaktnog algoritma, genetski algoritam ne pruža garancije da će pronaći optimalno rešenje niti može da prepozna da je dostigao optimalno rešenje (ako se to desi), te nema načina da se zaustavi tačno u tom trenutku kao u ovom eksperimentu (kada je optimalno rešenje bilo poznato unapred). Ipak, u ovom eksperimentu mereno je vreme do pronalaženja optimalnog rešenja, jer pruža neku informaciju o kvalitetu algoritma. Na slici 6.6 (levo), prikazana su pomenuta vremena u zavisnosti od dimenzije problema. Očigledno, u slučaju instanci problema veće dimenzije, genetski algoritam izvršava se drastično brže nego egzaktni algoritam. Konkretno, za dimenziju 50, egzaktni algoritam izvršava se 3141 sekundu, a genetski algoritam nalazi optimalno rešenje za svega 6 sekundi. Na istoj slici (desno) prikazan je i rast vrednosti najbolje pronađene jedinke kroz generacije za dimenziju 50. Grafik pokazuje da je većina vremena protekla pre pojavljivanja prve jedinke koja zadovoljava uslov nosivosti. Od njene pojave, optimizacija napreduje vrlo brzo. Ovo ponašanje lako je razumljivo. Za problem dimenzije 50, očekivana masa izbora koji definiše jedan nasumično generisani hromozom je 1250 (u proseku, polovina svih predmeta biće stavljen u ranac, a prosek njihovih masa je 50), što je značajno iznad ograničenja od 500, tako da nije verovatno da će u polaznoj populaciji biti jedinki koje zadovoljavaju ograničenje ukupne mase. S druge strane, u slučaju problema dimenzije 20, očekivana masa je 500, što odgovara ograničenju, pa je očekivano da će biti podjednako jedinki koje imaju i manju i veću masu. Promena inicijalizacije smanjenjem verovatnoće generisanja jedinica čini da optimizacija kreće mnogo ranije i u slučaju problema dimenzije 50.

	Životna navika/stanje	1	2	3	4	5	6	7
1	Fizička aktivnost	1.00	-0.59	0.52	-0.68	-0.83	-0.80	-0.28
2	Pušenje		1.00	-0.57	0.68	0.60	0.65	0.53
3	Zdrava ishrana			1.00	-0.65	-0.28	-0.35	-0.23
4	Gojaznost				1.00	0.67	0.71	0.46
5	Dijabetes					1.00	0.87	0.48
6	Visok krvni pritisak						1.00	0.57
7	Povišen holesterol							1.00

Tabela 6.1: Tabela korelacije životnih navika i zdravstvenih stanja.

Nepoznata	$c_1$	$c_2$	$c_3$	$c_4$	$c_5$	$c_6$	$c_7$
Vrednost iz intervala $[0, 255]$	187	61	217	78	46	38	21
Ugao	$262^\circ$	$85^\circ$	$305^\circ$	$109^\circ$	$64^\circ$	$53^\circ$	$29^\circ$

Tabela 6.2: Jedinka koja predstavlja najbolju vizualizaciju dobijenu genetskim algoritmom. Za svaku promenljivu  $c_i$ , data je njena vrednost u hromozomu u rasponu od 0 do 255 i odgovarajući ugao na krugu.

#### 6.4.2 Vizualizacija životnih navika i zdravstvenih stanja

Čak i uz duboku pažnju, razumevanje obimnih podataka često je teško i zamorno. Zbog toga se traga za pogodnim, sažetim načinima prezentovanja podataka. Na primer, matrica korelacije prikazuje zavisnost između različitih vrednosti: svako polje matrice sadrži vrednost *koeficijenta korelacije* između dve promenljive. Nećemo ulaziti u detalje ovog pojma i zadržaćemo se samo na njegovim osnovnim osobinama: koeficijent korelacije ima vrednosti između -1 i 1. Vrednost 1 znači da postoji savršena pozitivna zavisnost, vrednost -1 da postoji savršena negativna zavisnost, a vrednost 0 da nema zavisnosti.<sup>3</sup> Na primer, prisustvo gripe i povišena temperatura su visoko pozitivno korelisani. Ishodi u međusobnim susretima dva košarkaška tima savršeno su negativno korelisani (ako je jedan pobedio, drugi je izgubio). Među svim osobama rođenim 1980, visina i mesec rođenja verovatno nisu korelisani.

Tabela 6.1 prikazuje korelisanost nekih životnih navika i zdravstvenih stanja (podaci su stvari i odnose se na veliki uzorak stanovnika Sjednjnjenih Američkih Država). Na primer, navika vežbanja pozitivno je korelisana sa navikom zdrave ishrane – koeficijent korelacije jednak je 0.52.

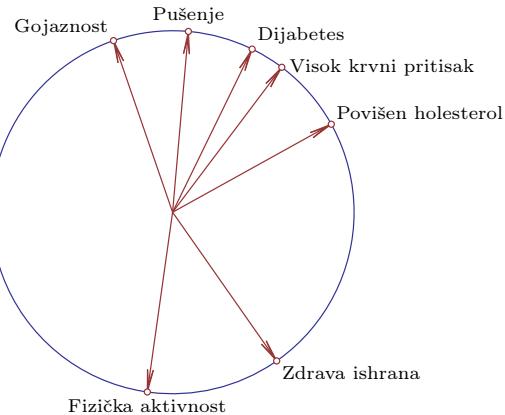
Navedena tabela sažeto prikazuje mnoštvo podataka koji se odnose na veliki broj osoba. Ali može se otići i korak dalje i, uz izvesna pojednostavljinjanja, vizualizovati njen sadržaj. Tako će suština tabele i njene glavne poruke biti još lakši za razumevanje. Sadržaj matrice korelacije može se vizualizovati na sledeći način: svakoj promenljivoj biće dodeljena jedna tačka na krugu, tako da promenljive koje su jače pozitivno korelisane budu međusobno što bliže, a promenljive koje su jače negativno korelisane budu međusobno što dalje. U konkretnom slučaju, time se ilustruje koje životne navike obično idu jedna sa drugom, a koje ne.

Precizirajmo sada način na koji to treba uraditi. Neka se razmatra  $n$  životnih navika, odnosno stanja i neka su koeficijenti korelacije jednaki  $x_{i,j}$ , za  $1 \leq i, j \leq n$ . Neka je  $c_i$ , za  $1 \leq i \leq n$  pozicija pridružena promenljivoj  $i$  na krugu (to su nepoznate veličine koje tražimo) i neka su njene moguće vrednosti od  $0^\circ$  do  $360^\circ$ . Neka je  $\mathbf{c}$  vektor tih veličina. Smatraćemo da je bliskost na krugu dve promenljive jednaka kosinusu ugla između njih. Ukoliko su dve tačke na krugu jednake, onda je njihova bliskost jednaka  $\cos 0^\circ = 1$ , a ukoliko su dijametralno suprotne onda je njihova bliskost jednaka  $\cos 180^\circ = -1$ . Takva mera upravo odgovara koeficijentu korelacije, pa je naš cilj da promenljive budu raspoređene na krugu tako da bliskost dve promenljive što bolje odgovara njihovom koeficijentu korelacije. Preciznije, problem se može definisati kao sledeći problem minimizacije:

$$\min_{\mathbf{c} \in [0, 360]^n} \sum_{i,j=1}^n (\cos(c_i - c_j) - x_{i,j})^2$$

Ovaj problem može se aproksimativno rešiti primenom genetskih algoritama. U konkretnom primeru, postoji 7 promenljivih. Svaka može imati vrednosti od  $0^\circ$  do  $360^\circ$ , ali taj interval možemo diskretizovati i zameniti intervalom celih brojeva  $[0, 255]$  (pri čemu se  $0^\circ$  preslikava u 0, levi kraj intervala, a  $360^\circ$  u 255, desni kraj intervala). Dakle, svaku od traženih vrednosti možemo predstaviti pomoću 8 bitova, pa će hromozom imati ukupno  $7 \times 8 = 56$  bitova.

<sup>3</sup>Preciznije bi bilo govoriti o linearnoj zavisnosti.



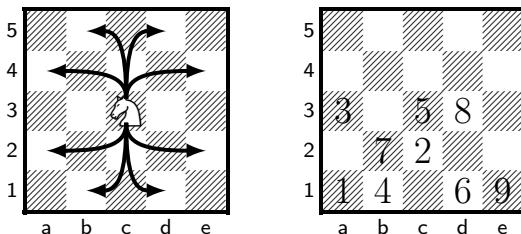
Slika 6.7: Vizualizacija međusobnih zavisnosti životnih navika i stanja.

Genetski algoritam koristi populaciju veličine 100, verovatnoća ukrštanja je 0.6, a verovatnoća mutacije 0.01. Korišćena je ruletska selekcija i generacijska politika zamene populacije. Jedinke su inicijalizovane generisanjem po 7 slučajnih brojeva u intervalu [0, 255]. Postupak optimizacije trajao je 10000 iteracija, a najbolje rešenje sa vrednošću ciljne funkcije 1.23 dobijeno je u 5558. iteraciji. Najbolja jedinka data je u tabeli 6.2 i ilustrovana slikom 6.7. Kako bismo se intuitivno uverili da ovakva ilustracija odražava informacije iz tabele korelacija, uočimo da su sve pozitivno korelisane promenljive raspoređene pod uglom manjim od  $90^\circ$ , a negativno korelisane pod uglom većim od  $90^\circ$ , osim zdrave ishrane i povиšenog holesterola koji na slici deluju nekorelisan. Ipak, zanimljivo je da to i jeste najslabija korelacija u tabeli.

Primetimo da je u ovom problemu diskretizacijom smanjena šansa da algoritam nađe optimalno rešenje. Dobijeno rešenje odražava informacije iz tabele korelacija ali, naravno, ne egzaktno. To nije ni bio cilj, već samo intuitivno razumljiva vizualizacija.

#### 6.4.3 Obilazak table skakačem

Problem obilaska table skakačem je problem pronalaženja putanje skakača na šahovskoj tabli  $n \times n$ , takve da skakač poseti što veći broj različitih polja ali nijedno polje dvaput ili više puta.<sup>4</sup> U daljem razmatranju, prepostavimo dimenzije table  $5 \times 5$  i da je skakač na početku u donjem levom uglu. Na slici 6.8, prikazana je jedna putanja skakača od osam poteza koja se ne može nastaviti.



Slika 6.8: Kretanje skakača (levo) i jedna putanja skakača od osam poteza koja se ne može nastaviti (desno).

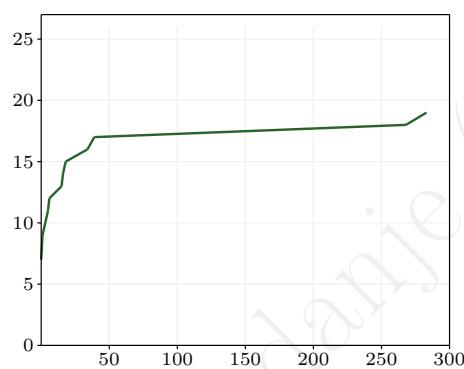
Za svako od 25 polja treba odrediti polje na koje je najbolje da skakač pređe. U zavisnosti od polja na kojem je, skakač može preći na dva do osam drugih polja, pa se izbor narednog polja uvek može kodirati pomoću tri binarne cifre, tj pomoću tri bita. Skakač može napraviti najviše 24 poteza. Stoga, za potrebe primene genetskog algoritma, hromozom se može sastojati iz  $24 \times 3 = 72$  bita, pri čemu svaka trojka odgovara jednom polju table i označava jedan od osam mogućih poteza sa tog polja. Očigledno, za neka polja neki od poteza koje hromozom može predstaviti neće biti legalna, ali ih svejedno dopuštamo.

<sup>4</sup>Strože postavljen problem je problem pronalaženja putanje skakača na šahovskoj tabli kojom skakač po jednom posećuje svako polje table.

Za funkciju cilja i funkciju prilagođenosti prirodno se nameće broj skokova koje skakač može da izvede u skladu sa evaluiranim hromozomom, do skoka van table ili do skoka na već posećeno polje.

Koristi se generacijski genetski algoritam – u svakom koraku bira se dovoljno jedinki da se generiše nova populacija, a potom se vrše ukrštanja i mutacije. Selekcija je jednostavna ruletska, a ukrštanje sa jednom tačkom prekida. Koristi se populacija od 3000 jedinki, verovatnoća ukrštanja 0.8, verovatnoća mutacije po hromozomu 0.05, a maksimalan broj iteracija je 1000.

Kako ponašanje genetskog algoritma može značajno zavisiti od polazne populacije koja se slučajno generiše, može se desiti da se u različitim pokretanjima dobije različit kvalitet rešenja. Kako bi se stekla bolja slika, rešavanje je bilo vršeno 10 puta. Potom su izračunati prosečan kvalitet najboljeg rešenja i prosečna generacija u kojoj je ono nađeno. Prosečna dužina pronađene putanje je 19.1. Prosečan broj iteracija koje su bile potrebne za dostizanje najboljeg pronađenog rešenja je 326.7. Na slici 6.9, prikazana je zavisnost dužine pređene putanje za najbolju jedinku u odnosu na broj iteracija za jedno izvršavanje genetskog algoritma. Nije pronađena putanja koja obilazi sva polja. Razlog za to je taj problem previše težak za pravolinjski pristup koji je upotrebljen. Za njegovo puno rešavanje potrebne su dodatne, napredne tehnike. Ovaj primer ilustruje i kako genetski algoritmi u nekim problemima ne dovode nužno do optimalnog rešenja.



Slika 6.9: Zavisnost kvaliteta najbolje jedinke u populaciji od broja generacija genetskog algoritma.

*Deo II*

---

## **Automatsko rasuđivanje**

---

Elektronsko izdanie (2020)



## Rešavanje problema korišćenjem automatskog rasuđivanja

Automatsko rasuđivanje (ili automatsko rezonovanje), bavi se razvojem algoritama i programa koji mogu da rasuđuju automatski, koristeći metode matematičko-deduktivnog zaključivanja u nekim konkretnim logičkim okvirima. Mnoge podoblasti veštacke inteligencije oslanjaju se na matematičku logiku i na automatsko rasuđivanje. Ove tehnike koristimo kada je potrebno netrivijalno rigorozno zaključivanje. Tu spadaju i problemi u okviru kojih je potrebno dokazati da neko tvrđenje važi za sve moguće objekte koji zadovoljavaju neke preduslove, kao i problemi u okviru kojih je potrebno dokazati da postoje objekti za koje neko tvrđenje važi.

Logičkih okvira ima mnogo i pojedinačno su pogodni za opisivanje raznovrsnih teorija i praktičnih problema: iskazna logika, logika prvog reda, logika višeg reda, fazi logika, modalna logika, temporalna logika, deontičke logike, itd. Zbog ograničenosti prostora, mi ćemo se u nastavku baviti samo iskaznom logikom i logikom prvog reda.

8	2	7	1	5	4	3	9	6
9	6	5	3	2	7	1	4	8
3	4	1	6	8	9	7	5	2
5	9	3	4	6	8	2	7	1
4	7	2	5	1	3	6	8	9
6	1	8	9	7	2	4	3	5
7	8	6	2	3	5	9	1	4
1	5	4	7	9	6	8	2	3
2	3	9	8	4	1	5	6	7

Slika 7.1: Sudoku.

**Primer 7.1.** Popularna igra „sudoku“ (za jednog igrača) igra se na sledeći način. Data je tabela koja se sastoji od devet kolona i devet vrsta. Polja su dodatno grupisana u 9 kvadrata od po 9 polja. Na početku neka polja sadrže brojeve i zadatak je popuniti čitavu tabelu tako da važe sledeći uslovi:

- Svako polje sadrži neki broj između 1 i 9;
- Svaka vrsta sadrži sve brojeve između 1 i 9;
- Svaka kolona sadrži sve brojeve između 1 i 9;
- Svaki mali kvadrat sadrži sve brojeve između 1 i 9.

Iz navedenih uslova jednostavno sledi da ni u jednoj koloni, vrsti niti malom kvadratu ne postoji broj koji se pojavljuje dva puta. Slika 7.1 prikazuje jedan rešeni sudoku (krupnijim ciframa prikazani su unapred zadati brojevi, sitnjim brojevi koji čine rešenje).

Kako svako polje ima jednu od vrednosti iz konačnog skupa, ovaj zadatak, u principu, može biti rešen sistematičnim ispitivanjem svih mogućnosti. Ipak, broj svih mogućnosti toliko je veliki (maksimalno  $9^{81}$ ), da to nije praktično primenljivo rešenje ni za računar, a kamoli za čoveka. Zato rešavanje sudokua zahteva netrivijalno rasuđivanje. U toku procesa rešavanja, za neka polja može se jednoznačno odrediti vrednost, tj. može se zaključiti, dokazati da nema drugih mogućnosti. Slično, za neka polja broj mogućnosti može da se svede samo na dve. Takvim zaključivanjem prostor pretrage značajno se smanjuje i uspešno rešavanje postaje moguće. Sredstva automatskog rasuđivanja mogu da se koriste za donošenje pojedinačnih odluka u fazi rešavanja ili za rešavanje celokupnog problema.

**Primer 7.2.** Pretpostavimo da je pitanje da li može doći do greške u izvršavanju narednog koda:

```
int f(int y)
{
    int i, x=1;
    for (int i=0; i<y; i++)
        x += 2y;
    return 1000 / x;
}
```

Jedina naredba u kojoj može doći do greške u fazi izvršavanja je naredba `return`, u okviru koje se računa vrednost izraza primenom operacije deljenja. Do greške može doći ako je `x` jednak 0. Može se dokazati da `x` ni u kom slučaju ne može biti jednak 0 i to zahteva neko netrivijalno rasuđivanje. Postoje algoritmi koji omogućavaju automatsko dokazivanje ovakvih tvrdjenja.

**Primeri primena automatskog rasuđivanja.** Tehnike automatskog rasuđivanja koriste se u razvoju matematičkih teorija, u problemima rasporedivanja, u verifikaciji softvera i hardvera i drugde.

Raspored utakmica za špansku fudbalsku ligu, više godina pravljen je korišćenjem namenskih programa za rasuđivanje u iskaznoj logici – SAT rešavača. I mnogi drugi problemi raspoređivanja rešavani su na sličan način.

Godine 1996. program Otter/EQP zasnovan na metodu rezolucije, nakon osam dana rada, automatski je dokazao hipotezu otvorenu oko pedeset godina ranije i do tada nedokazanu: svaka Robinsova algebra je Bulova. Tu vest preneli su tada gotovo svi svetski mediji kao vest o značajnoj prekretnici – polje matematičkih otkrića, rezervisano do tada samo za ljude, počeli su da osvajaju i računari.

Formalne metode su oblast računarstva koja se bavi specifikovanjem, razvojem i verifikacijom softverskih i hardverskih sistema. U okviru ovih metoda, analize zasnovane na matematičkoj logici imaju zadatak da obezbede pouzdanost sistema koji se razvijaju. Ključnu ulogu u formalnim metodama imaju logika i automatsko rasuđivanje. Sistem kontrola leta u Ujedinjenom Kraljevstvu i linije pariskog metroa bez vozača, na primer, razvijeni su i verifikovani formalnim metodama.

Namenski programi za rasuđivanje u specijalizovanim teorijama kao što je, na primer, linearna aritmetika, zovu se SMT rešavači i intenzivno se koriste u verifikaciji softvera i hardvera – testiranju i dokazivanju njegove ispravnosti, ali i u sintezi programa. Jedan od najznačajnijih rešavača je z3, kompanije Microsoft.

**Faze rešavanja problema korišćenjem automatskog rasuđivanja.** Tök rešavanja problema korišćenjem automatskog rasuđivanja obično obuhvata sledeće osnovne faze:

- modelovanje problema;
- rešavanje problema opisanog u matematičkim terminima;
- interpretiranje i analiza rešenja.

Navedene faze slične su fazama rešavanja u drugim podoblastima veštačke inteligencije. O specifičnostima će biti reči u daljem tekstu.

## 7.1 Modelovanje problema

Modelovanje problema predstavlja formulisanje problema precizno, u matematičkim terminima, korišćenjem pogodnog logičkog okvira.

Potrebno je najpre utvrditi vrstu osnovnih objekata i nepoznatih veličina u problemu. Na primer, to mogu biti jednostavni iskazi, (poznati i nepoznati) celi brojevi i slično. Onda treba opisati uslove koji moraju da važe za objekte opisane u problemu.

U nekom logičkom okviru ne mogu da se izraze neki uslovi, pa u fazi modelovanja biramo i pogodni logički okvir, na primer – iskaznu logiku (glava 8) ili logiku prvog reda (glava 9). Dalje, u okviru kao što je logika prvog reda, mogu se opisati teorije pogodne za opisivanje raznovrsnih struktura i njihovih svojstava. Postoje, na primer, teorije koje opisuju realne brojeve, teorije koje opisuju cele brojeve fiksne širine i tako dalje.

**Primer 7.3.** Šef protokola na jednom dvoru treba da organizuje bal za predstavnike ambasada. Kralj traži da na bal bude pozvan Peru ili da ne bude pozvan Katar (Qatar). Kraljica zahteva da budu pozvani Katar ili Rumunija (ili i Katar i Rumunija). Princ zahteva da ne bude pozvan Peru ili da ne bude pozvana Rumunija (ili da ne budu pozvani ni Peru ni Rumunija). Da li je moguće organizovati bal tako da su zadovoljeni zahtevi svih članova kraljevske porodice?

Navedeni problem potrebno je najpre modelovati na neki precizan način. Iskaz „na bal će doći ambasador Peru“ označićemo sa  $p$ , iskaz „na bal će doći ambasador Katara“ označićemo sa  $q$ , a iskaz „na bal će doći ambasador Rumunije“ sa  $r$ . Uslov koji postavlja kralj, onda glasi „važi  $p$  ili ne važi  $q$ “ ili kraće zapisano „ $p$  ili ne  $q$ “. Uslov koji postavlja kraljica glasi „ $q$  ili  $r$ “. Uslov koji postavlja princ glasi „ne  $p$  ili ne  $r$ “.

Sva navedena ograničenja, svi ovi iskazi, zajedno čine novi, komplikovaniji iskaz koji možemo da zapišemo na sledeći način:

$$\text{„}(p \text{ ili ne } q) \text{ i } (q \text{ ili } r) \text{ i } (\text{ne } p \text{ ili ne } r)\text{“}$$

Ovaj složeni iskaz predstavlja precizan zapis problema. Potrebno je proveriti da li polazni iskazi  $p$ ,  $q$  i  $r$  mogu da imaju konkretnе vrednosti tačno ili netačno takve da složeni iskaz ima vrednost tačno. Da bi se taj problem rešio potrebno je precizno definisati i na koji način se složenim iskazima pridružuje vrednost tačno ili netačno ukoliko je poznato koje vrednosti su pridružene polaznim iskazima.

**Primer 7.4.** U primeru 7.1, osnovni objekti mogu biti iskazi koji opisuju sadržaj polja. Jedan takav iskaz je „polje (1,2) sadrži vrednost 5“. Na taj način, možemo uvesti  $81 \times 9$  iskaza od kojih će u konačnom rešenju neki biti tačni a neki netačni. Odnosi između ovakvih iskaza mogu se opisati uslovima poput „ako polje (2,3) sadrži vrednost 4, onda polje (2,4) ne sadrži vrednost 4“. Kao polazne pretpostavke treba uvesti iskaze o zadatim poljima, na primer „polje (1,9) sadrži vrednost 2“.

S druge strane, u ovom problemu osnovnim objektima možemo smatrati cele brojeve koji čine vrednosti polja tabele. Na primer, promenljiva  $x_{1,2}$  može da označava vrednost polja (1,2). Moraju da važe uslovi poput  $1 \leq x_{1,2} \leq 9$  i uslovi poput „ako važi  $x_{2,3} = 4$ , onda važi  $x_{2,4} \neq 4$ “. Polazne pretpostavke daju uslove o zadatim poljima, na primer  $x_{1,9} = 2$ .

**Primer 7.5.** U primeru 7.2, programske promenljive možemo modelovati celim brojevima ili brojevima fiksne širine, takozvanim bitvektorima.

Neka programska promenljiva  $\mathbf{x}$  bude modelovana promenljivom  $x$  koja ima celobrojne vrednosti. Precizije, kako se vrednosti programske promenljive  $\mathbf{x}$  menjaju tokom izvršavanja programa, njoj neće odgovarati jedna promenljiva  $x$ , nego niz promenljivih  $x_0, x_1, x_2, x_3, \dots$ . Slično, neka programskoj promenljivoj  $y$  odgovara promenljiva  $\mathbf{y}$  koja ima celobrojne vrednosti ( $y$  se ne menja tokom izvršavanja programa, pa možemo da je modelujemo jednom promenljivom). Važe sledeći uslovi: „ $x_0 = 1$ “, „ $x_{i+1} = x_i + 2y$ “, za svako  $i \geq 0$ . Ukoliko za modelovanje koristimo cele brojeve, a želimo da rad datog programa modelujemo precizno, u navedenim uslovima trebalo bi dodati računanje po modulu (koje odgovara izračunavanju na računaru).

Odsustvo greške u navedenom programu modelujemo uslovom „izraz  $1000 / \mathbf{x}$  je u datom programu uvek definisan“, tj. uslovom „promenljiva  $\mathbf{x}$  nikad nema vrednost 0“. U terminima uvedenih promenljivih ovaj uslov glasi: „ $x_i = 0$ “ ne važi ni za jednu vrednost  $i \geq 0$ .

**Primer 7.6.** Date su sledeće tvrdnje: „svaki čovek je smrtan“, „Sokrat je čovek“. Pitanje je da li se iz ovih tvrdnji može utvrditi da je tačno i „Sokrat je smrtan“.

Tvrdnju da je  $x$  čovek zapišimo kao „ $x$  je čovek“ a tvrdnju da je  $x$  smrtan zapišimo kao „ $x$  je smrtan“. Tvrdnju „svaki čovek je smrtan“ zapišimo „za svaku  $x$  važi: ako ( $x$  je čovek) onda ( $x$  je smrtan)“. Zadato tvrdjenje onda (pomalo rogobatno u odnosu na svakodnevni jezik) glasi: „ako (za svaku  $x$  važi: ako ( $x$  je

čovek) onda ( $x$  je smrtan)) i (Sokrat je čovek) onda (Sokrat je smrtan)“.

Ključna razlika u odnosu na primer 7.3 je to što navedeni iskazi (pa i složene tvrdnje) zavise od nekog parametra  $x$ . To sugerije da će logički okvir potreban za rešavanje ovog problema morati da bude izražajniji od logičkog okvira za primer 7.3.

## 7.2 Rešavanje problema opisanog u matematičkim terminima

U fazi rešavanja, traži se rešenje matematički formulisanog problema, korišćenjem pogodnih metoda zaključivanja. Metode za automatsko rasuđivanje (za logički formulisane probleme) razvijaju se uspešno već više od šezdeset godina.

Ispitivanje da li, pod nekim uslovima, složeni iskaz iz primera 7.3 može biti tačan, može se sprovesti tako što bi bile ispitane vrednosti složenog iskaza za sve moguće vrednosti pridružene iskazima  $p$ ,  $q$  i  $r$ . Tih iskaza ima tri, za svaki postoje dve mogućnosti – tačno i netačno, pa ukupno ima  $2^3 = 8$  mogućnosti koje treba ispitati. Slično, ako se za rešavanje sudokua koristi prvi stil modelovanja iz primera 7.4, onda postoji  $2^{81 \cdot 9} = 2^{729}$  mogućnosti koje treba ispitati. Međutim, umesto takvog naivnog pristupa, u praksi se koriste pristupi koji slične analize sprovode daleko efikasnije. Na primer, ako smo pretpostavili da je  $p$  tačno, onda je čitav uslov „ $p$  ili ne  $q$ “ tačan i nema potrebe da razmatramo mogućnosti za  $q$ . Slično, ako smo pretpostavili da je  $p$  netačno, onda je i čitav uslov „ $p$  i  $q$ “ netačan, nezavisno od vrednosti  $q$ .

U primeru 7.2, potrebno je dokazati da iz uslova „ $x_0 = 1$ “ i „ $x_{i+1} = x_i + 2y$ “, za svako  $i \geq 0$  sledi da „ $x_i = 0^n$ “ ne važi ni za jednu vrednost  $i \geq 0$ . Navedeno tvrđenje možemo dokazati za bilo koju konkretnu vrednost  $i$ , višestrukom primenom tvrđenja „ako je  $x_i$  neparan broj, onda je i  $x_{i+1} = x_i + 2y$  neparan broj“. To se može dokazati u teoriji koja opisuje cele brojeve ili cele brojeve fiksne širine (i sabiranje nad njima). Međutim, ukoliko želimo da dokažemo da „ $x_i = 0^n$ “ ne važi ni za jednu vrednost  $i \geq 0$ , onda naš logički okvir mora da uključuje i princip matematičke indukcije. Postoje algoritmi za automatsko rezonovanje i za takve logičke okvire.

Ispitivanje da li je tačno tvrđenje iz primera 7.6 može se sprovesti na sledeći način: pošto za svako  $x$  važi „ako ( $x$  je čovek) onda ( $x$  je smrtan)“, važi i kada je  $x$  upravo Sokrat, tj. važi „ako (Sokrat je čovek) onda (Sokrat je smrtan)“. Odatle i iz „Sokrat je čovek“, primenom pravila *modus ponens* (iz  $A$  i  $A \Rightarrow B$  sledi  $B$ ) sledi „Sokrat je smrtan“, pa važi dato tvrđenje.

Navedeni procesi zaključivanja opisani su neformalno i grubo, ali služe kao motivacija za stroga pravila zaključivanja u iskaznoj logici i logici prvog reda i njihovo automatizovanje.

## 7.3 Interpretiranje i analiza rešenja

Dobijeno rešenje matematički formulisanog problema potrebno je formulisati u terminima početnog problema i potrebno je razumeti svojstva dobijenog rešenja. Na primer, ako se rešavanjem primera 7.3 dobije da  $p$  mora biti tačno, to znači da na bal treba da dođe ambasador Perua.

Zaključci koji se dobijaju procesom rešavanja su nesumnjivi, oni ne mogu biti pogrešni (sem ako je neispravan program koji sprovodi rezonovanje). U tom smislu nikakva analiza, niti evaluacija tačnosti dobijenih zaključaka nije potrebna. Međutim, često su potrebni dodatna analiza i objašnjenje dobijenog rešenja u zavisnosti od izabranog modelovanja. Na primer, dva različita načina modelovanja opisana u primeru 7.5 nisu ekvivalentna: zaista, kod celih brojeva nema prekoračenja, a kod celih brojeva fiksne širine – ima. Rasuđivanje u domenu celih brojeva može da bude efikasnije, ali je rasuđivanje u domenu brojeva fiksne širine preciznije i vernije opisuje programske promenljive. Dakle, ukoliko smo se odlučili za modelovanje koje omogućava efikasno automatsko zaključivanje, ali ne modeluje zadati problem verno, treba utvrditi u kojim slučajevima ima odstupanja u potrebnim zaključcima. Dodatno, u fazi analize treba utvrditi da li konkretna instanca problema potпадa pod ta odstupanja, tj. da li rezultat faze rešavanja može da se neposredno odnosi na polazni zadatak.

Pored navedenog, nekad je potrebno tumačiti i neuspeh faze rešavanja. Ako neki sistem ne uspe da dokaže tvrđenje  $S$ , u nekim slučajevima to može da znači da  $S$  zaista nije tačno, a u nekim slučajevima može da bude posledica činjenice da korišćeni sistem nije u stanju da dokaže sva moguća tačna tvrđenja (dakle, moguće je da je  $S$  tačno, ali naš sistem to ne može da dokaže). Nekada se ne može razlučiti (čak i ako poznajemo algoritam rasuđivanja) između ta dva: nekada ako sistem prijavi da nije dokazao  $S$ , to još ne znači da  $S$  nije tačno.

Najvažnija opšta svojstva koje algoritmi za rasuđivanje mogu da imaju su sledeća:

**Potpunost** je svojstvo koje kaže da je algoritam u stanju da dokaže svako tvrđenje iz svog domena koje je tačno (pre ili kasnije, nekada je za to potrebno veoma dugo vreme).

**Saglasnost** je svojstvo koje kaže da ako algoritam tvrdi da je neko tvrđenje (iz njegovog domena) tačno, onda ono zaiste jeste tačno.

Potpunost je svakako poželjna osobina, a saglasnost je neophodna (kako bi zaključci koje donose algoritmi bili nesumnjivi).

Moć nekih algoritama logičkog rasuđivanja ograničena je sâmom prirodom problema koji se razmatraju. Za *problem odlučivanja*, tj. za problem koji zahteva odgovor „da“ ili „ne“, kažemo da je *odlučiv* ako postoji algoritam koji može da reši svaku njegovu instancu, a algoritam sa takvim svojstvom zovemo *procedura odlučivanja*. Na primer, neki problem raspoređivanja je *odlučiv* ako postoji algoritam koji za svaku instancu tog problema može da utvrdi da li traženi raspored postoji ili ne. I mnogi problemi rasuđivanja su problemi odlučivanja, pa se ove definicije odnose i na njih. Na primer, odlučiv je problem SAT – problem ispitivanja iskazne zadovoljivosti. Mnogi važni problemi logičkog rasuđivanja nisu odlučivi, tj. oni su *neodlučivi*. Za takve probleme moramo biti spremni da nećemo moći da dobijemo rešenje za proizvoljnu ulaznu instancu. Moguće je razviti samo algoritam koji primenom nekakvih heuristika može da reši neke instance problema. Postoje i *poluodlučivi problemi*: oni nisu odlučivi ali za njih mogu da postoje algoritmi koji uspešno mogu da reše problem za sve instance za koje je odgovor „da“. Takav algoritam zovemo *procedura poluodlučivanja*. Za instance za koje je ispravan odgovor „ne“, procedura poluodlučivanja vraća odgovor „ne“ ili se ne zaustavlja. Poluodlučiv je, na primer, halting problem – problem ispitivanja da li se dati program zaustavlja za datu ulaznu vrednost. I mnogi važni problemi logičkog rasuđivanja su poluodlučivi. Na primer, poluodlučiv je problem ispitivanja valjanosti u logici prvog reda.

Pored navedenih pitanja, za algoritme rasuđivanja, kao i za skoro sve druge vrste algoritama važna su pitanja vremenske i prostorne složenosti (njegoreg i prosečnog slučaja).



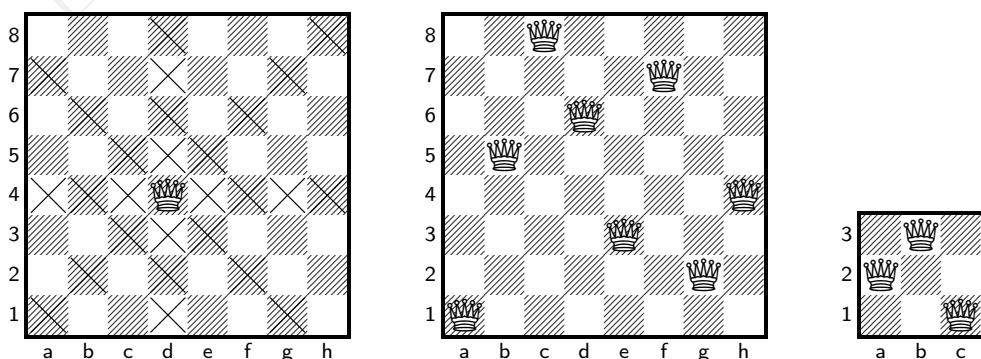
## Glava 8

# Automatsko rasuđivanje u iskaznoj logici

U iskaznoj logici (eng. *propositional logic*) razmatraju se iskazi, tvrdnje, tvrđenja (na primer „Ana je zubar“). Iskazi mogu biti kombinovani u složenije iskaze logičkim veznicima (na primer „Ana je zubar i Ana živi u Beogradu“). Iskazna logika ima svoju sintaksu (koja opisuje njen jezik) i svoju semantiku (koja definiše istinitosnu vrednost iskaza). Centralni problemi u iskaznoj logici su ispitivanje da li je data iskazna formula *valjana* (*tautologija*) tj. da li je tačna za sve moguće istinitosne vrednosti elementarnih iskaza od kojih je sačinjena (na primer tvrđenje „Ana je zubar ili Ana nije zubar“ je uvek tačno – i ako je tvrđenje „Ana je zubar“ tačno i ako je netačno), kao i ispitivanje da li je data iskazna formula *zadovoljiva*, tj. da li je tačna za neke istinitosne vrednosti elementarnih iskaza od kojih je sačinjena (na primer tvrđenje „Ana je zubar i Ana živi u Beogradu“ može mada ne mora uvek biti tačno). Problem ispitivanja zadovoljivosti formule u KNF obliku poznat je kao problem SAT i on je centralni predstavnik klase NP-kompletnih problema. Postoji više metoda za ispitivanje valjanosti i zadovoljivosti.

U algoritmima za logičko zaključivanje često je neki korak zaključivanja moguće sprovesti na više različitih načina, ali nije precizirano koji od tih načina treba da se izabere. Naime, bez obzira na načinjeni izbor, izvedeni zaključci su uvek ispravni. Međutim, neki putevi do istog zaključka mogu da budu znatno kraći od drugih i tada je proces automatskog rasuđivanja znatno efikasniji. Ovo pokazuje da je i u logičkom rasuđivanju jedan od centralnih problema problem usmeravanja pretrage.

Iskazna logika dovoljno je izražajna za opisivanje raznovrsnih problema, uključujući mnoge praktične probleme, kao što su, na primer, problemi raspoređivanja ili dizajniranje kombinatornih kola. Iskazna logika pogodna je posebno za opisivanje problema nad konačnim domenima. Naime, svaki objekat koji može imati konačan broj stanja može se opisati konačnim brojem iskaznih promenljivih: ako je broj mogućih stanja  $2^n$ , onda je dovoljno koristiti  $n$  iskaznih promenljivih. Svi brojevi zapisani u računaru zapisani su bitovima, pa se i svi ti brojevi mogu modelovati iskaznim promenljivim: koliko bitova, toliko iskaznih promenljivih. Sabiranje celih brojeva (kao i mnoge druge operacije) onda može da se opiše u terminima iskazne logike. Slično važi i za mnoge druge vrste podataka i mnoge vrste problema. Sa tako velikom izražajnom snagom i velikim brojem raznolikih primena, iskazna logika i rešavači za iskaznu logiku često se smatraju „švajcarskim nožićem“ savremenog računarstva, a posebno – veštačke inteligencije.



Slika 8.1: Kretanje dame na tabli  $8 \times 8$  (levo), jedno rešenje za problem osam dama (sredina), jedno raspoređivanje za problem tri dame koji nije rešenje (desno).

**Primer 8.1.** Razmotrimo, za ilustraciju rešavanja primenom logike, problem „n dama”, opisan ukratko u primeru 3.4 i detaljnije razmatran u poglavlju 8.4.3. Cilj je rasporediti n dama na šahovskoj tabli dimenzija  $n \times n$  tako da se nikoje dve dame ne napadaju. Na slici ?? (levo i sredina) prikazano je kretanje dame i jedno rešenje problema za  $n = 8$ .

Jednostavnosti radi, u nastavku ćemo razmatrati problem dimenzije 3, za koji je jedno neispravno raspoređivanje tri dame (raspoređivanje koje ne čini rešenje) prikazano na slici ?? (desno). Uslovi koje ispravno raspoređivanje treba da zadovolji su:

- na jednom od polja  $a_1, a_2, a_3$  nalazi se dama.
- na jednom od polja  $b_1, b_2, b_3$  nalazi se dama.
- na jednom od polja  $c_1, c_2, c_3$  nalazi se dama.
- ako je neka dama na polju  $a_1$ , onda na polju  $a_2$  ne može da bude dama.
- ako je neka dama na polju  $a_1$ , onda na polju  $a_3$  ne može da bude dama.
- ako je neka dama na polju  $a_2$ , onda na polju  $a_1$  ne može da bude dama.
- ako je neka dama na polju  $a_2$ , onda na polju  $a_3$  ne može da bude dama.
- ako je neka dama na polju  $a_3$ , onda na polju  $a_1$  ne može da bude dama.
- ako je neka dama na polju  $a_3$ , onda na polju  $a_2$  ne može da bude dama.
- ...
- ako je neka dama na polju  $a_2$ , onda na polju  $b_3$  ne može da bude dama.
- ako je neka dama na polju  $b_3$ , onda na polju  $a_2$  ne može da bude dama.
- ako je neka dama na polju  $b_1$ , onda na polju  $c_2$  ne može da bude dama.
- ako je neka dama na polju  $c_2$ , onda na polju  $b_1$  ne može da bude dama.

Navedeni uslovi zavise od iskaza oblika „na polju ?? nalazi se dama“. Označimo sa  $p_{a1}$  iskaz „na polju  $a_1$  nalazi se dama“, sa  $p_{a2}$  iskaz „na polju  $a_2$  nalazi se dama“, ..., sa  $p_{c3}$  iskaz „na polju  $c_3$  nalazi se dama“. Onda navedeni uslovi mogu da se zapisu kraće:

- $p_{a1} \text{ ili } p_{a2} \text{ ili } p_{a3}$ .
- $p_{b1} \text{ ili } p_{b2} \text{ ili } p_{b3}$ .
- $p_{c1} \text{ ili } p_{c2} \text{ ili } p_{c3}$ .
- ako je  $p_{a1}$ , onda nije  $p_{a2}$ .
- ako je  $p_{a1}$ , onda nije  $p_{a3}$ .
- ako je  $p_{a2}$ , onda nije  $p_{a1}$ .
- ako je  $p_{a2}$ , onda nije  $p_{a3}$ .
- ako je  $p_{a3}$ , onda nije  $p_{a1}$ .
- ako je  $p_{a3}$ , onda nije  $p_{a2}$ .
- ...
- ako je  $p_{a2}$ , onda nije  $p_{b3}$ .
- ako je  $p_{b3}$ , onda nije  $p_{a2}$ .
- ako je  $p_{b1}$ , onda nije  $p_{c2}$ .

- ako je  $p_{c2}$ , onda nije  $p_{b1}$ .

Ovim su, od jednostavnih iskaza, konstruisani složeniji. Skup svih navedenih složenih uslova čini još složeniji iskaz – iskaz koji sadrži sve uslove zadatka. Sintaksa iskazne logike govori o pravilima po kojim se od elementarnih iskaza mogu konstruisati složeniji, to jest, o pravilima za konstruisanje ispravnih iskaznih formula.

Svaki od jednostavnih iskaza kao što je  $p_{a1}$  može biti tačan ili netačan. U zavisnosti od toga, može se odrediti istinitosna vrednost složenijih iskaza. Na primer, ako je  $p_{a1}$  tačno, a  $p_{a2}$  netačno, onda je tačno i „ako je  $p_{a1}$ , onda nije  $p_{a2}$ “. Semantika iskazne logike govori o tome kako se složenim iskazima (to jest, iskaznim formulama) određuje istinitosna vrednost na osnovu istinitosnih vrednosti elementarnih iskaza. Pošto je u fazi rešavanja problema vrednost iskaza kao što je  $p_{a1}$  nepoznata i pošto on može biti tačan ili netačan,  $p_{a1}$  ćemo zvati i iskazna promenljiva.

Sâmo rešavanje ovako modelovanog problema „n dama“ može se svesti na rešavanje sledećeg problema: odrediti istinitosne vrednosti elementarnih iskaza  $p_{a1}, p_{a2}, \dots, p_{c3}$  takve da svi navedeni uslovi budu ispunjeni (to jest, da svi odgovarajući iskazi imaju istinitosnu vrednost tačno). Može se razmatrati pitanje da li takve istinitosne vrednosti uopšte postoje, to jest da li početni problem uopšte ima rešenja ili pitanje pronalaženja svih rešenja. Pitanje da li uopšte postoji rešenje može se rešiti razmatranjem svih mogućih varijacija vrednosti za  $p_{a1}, p_{a2}, \dots, p_{c3}$ . Takvih varijacija ima  $2^9 = 512$  i razmatranje svih je naporno i nepraktično. Postoje i metode koje ne razmatraju sve mogućnosti i postojanje rešenja mogu obično da ispitaju znatno efikasnije.

U kontekstu navedenih ograničenja, mogu se izvesti i neki zaključci. Na primer, ako važi  $p_{a1}$  onda važi  $p_{b2}$  ili  $p_{b3}$ . Ovakvi zaključci mogu se dobiti različitim pristupima, a mogu se koristiti za ubrzavanje traganja za rešenjem.

## 8.1 Sintaksa i semantika iskazne logike

Sintaksa iskazne logike govori o njenom jeziku – o skupu njenih (ispravno formiranih) formula i ne razmatra njihovo značenje i njihove (moguće) istinitosne vrednosti. Značenje iskaznih formula, na osnovu izabranog značenja elementarnih iskaza od kojih je sačinjena, uvodi semantika iskazne logike.

### 8.1.1 Sintaksa iskazne logike

Skup iskaznih formula obično se definiše za fiksiran, prebrojiv skup *iskaznih promenljivih* (*iskaznih varijabli*) ili *iskaznih slova*  $P$ , dve logičke konstante –  $\top$  (te) i  $\perp$  (nete), kao i konačan skup osnovnih logičkih (tj. bulovskih) veznika: unarnog – *negacija* i binarnih – *konjunkcija*, *disjunkcija*, *implikacija*, *ekvivalencija*.

**Definicija 8.1** (Skup iskaznih formula).

- Iskazne promenljive (elementi skupa  $P$ ) i logičke konstante su iskazne formule;
- Ako su  $A$  i  $B$  iskazne formule, onda su iskazne formule i objekti dobijeni kombinovanjem ovih formula logičkim veznicima;
- Iskazne formule mogu biti dobijene samo na navedene načine.

U navedenoj definiciji (u duhu *apstraktne sintakse*) ne govori se o tome kako se zapisuju ili čitaju iskazne formule, već samo o tome kako se grade na apstraktan način (možemo da ih zamislimo u vidu stabla). Način na koji se logički veznici i iskazne formule zapisuju može se zadati konkretnom sintaksom – u vidu niza simbola. Uobičajeno je da se negacija zapisuje kao  $\neg$ , konjunkcija kao  $\wedge$ , disjunkcija kao  $\vee$ , implikacija kao  $\Rightarrow$  i ekvivalencija kao  $\Leftrightarrow$ . U takvom konkretnom zapisu – zapisu u vidu konkretnih nizova simbola, ako su  $A$  i  $B$  iskazne formule, onda su iskazne formule i  $(\neg A)$ ,  $(A \wedge B)$ ,  $(A \vee B)$ ,  $(A \Rightarrow B)$  i  $(A \Leftrightarrow B)$ . Na primer, zapis  $(A \wedge \top)$  čitamo „ $A$  i te“. U ovakovom zapisu, neophodno je koristiti zagrade kako bi se izbegla višesmislenost. Da bi se izbeglo korišćenje velikog broja zagrada, obično se izostavljaju spoljne zagrade i podrazumeva se sledeći prioritet veznika (od višeg ka nižem):  $\neg$ ,  $\wedge$ ,  $\vee$ ,  $\Rightarrow$ ,  $\Leftrightarrow$ .

Elemente skupa  $P$  i logičke konstante zovemo *atomičkim iskaznim formulama*. Literal je iskazna formula koja je ili atomička iskazna formula ili negacija atomičke iskazne formule.

Ako su dve iskazne formule  $A$  i  $B$  identične (tj. ako su, zapisane u konkretnoj sintaksi, jednake kao nizovi simbola), onda to zapisujemo  $A = B$ , a inače pišemo  $A \neq B$ .

Elementi skupa  $P$  obično se označavaju malim latiničnim slovima (eventualno sa indeksima). Izkazne formule obično se označavaju velikim latiničnim slovima (eventualno sa indeksima). Skupovi iskaznih formula obično se označavaju velikim slovima grčkog alfabetu (eventualno sa indeksima).

**Primer 8.2.** *Uslovi iz primera 8.1 mogli bi da se zapišu kao iskazne formule nad skupom iskaznih promenljivih  $\{p_{a1}, p_{a2}, \dots, p_{c3}\}$ . Na primer, uslov „na jednom od polja  $a_1, a_2, a_3$  nalazi se dama”, tj. uslov „ $p_{a1} \text{ ili } p_{a2} \text{ ili } p_{a3}$ “ može da se zapiše kao  $p_{a1} \vee p_{a2} \vee p_{a3}$ . Analogno zapisujemo i preostale uslove:*

- $p_{b1} \vee p_{b2} \vee p_{b3}$ .
- $p_{c1} \vee p_{c2} \vee p_{c3}$ .
- $p_{a1} \Rightarrow \neg p_{a2}$ .
- $p_{a1} \Rightarrow \neg p_{a3}$ .
- $p_{a2} \Rightarrow \neg p_{a1}$ .
- $\dots$
- $p_{c2} \Rightarrow \neg p_{b1}$ .

Ako iskazne formule zamišljamo u vidu stabla (ili ih predstavljamo u računarskom programu u vidu stabla), onda svakom podstablu stabla koje odgovara nekoj iskaznoj formuli takođe odgovara iskazna formula. Sve te formule zovemo *potformulama* (eng. *subformulae*) formule koja odgovara korenu stabla. Skup potformula jedne formule može se, preciznije, definisati na sledeći način.

**Definicija 8.2** (Skup potformula). *Skup potformula formule A definisan je na sledeći način:*

- svaka iskazna formula A je potformula sâma sebi;
- ako je A jednako  $\neg B$ , onda je svaka potformula formule B istovremeno i potformula formule A. Ako je A jednako  $B \wedge C$ ,  $B \vee C$ ,  $B \Rightarrow C$  ili  $B \Leftrightarrow C$ , onda je svaka potformula formule B i svaka potformula formule C istovremeno i potformula formule A;
- Potformule formule A su samo formule opisane prethodnim stavkama.

**Primer 8.3.** *Skup potformula formule  $(p \Rightarrow q) \vee r$  je  $\{p, q, r, p \Rightarrow q, (p \Rightarrow q) \vee r\}$ .*

Na različite načine može se definisati *složenost* iskaznih formula. Na primer, složenost može biti definisana kao dubina stabla koje odgovara formuli ili kao broj logičkih vezika koje formula sadrži.

Često je potrebno jedan deo iskazne formule zameniti nekom drugom iskaznom formulom. Pojam zamene precizno uvodi naredna definicija.

**Definicija 8.3** (Zamena). *Rezultat zamene (supstitucije) svih pojavljivanja iskazne formule C u iskaznoj formuli A iskaznom formulom D označavamo sa  $A[C \mapsto D]$ . Ta zamena definiše se na sledeći način:*

- ako za iskazne formule A i C važi  $A = C$ , onda je  $A[C \mapsto D]$  jednako D;
- ako za iskazne formule A i C važi  $A \neq C$  i A je atomička iskazna formula, onda je  $A[C \mapsto D]$  jednako A;
- ako za iskazne formule A, B i C važi  $A \neq C$  i  $A = (\neg B)$ , onda je  $A[C \mapsto D] = \neg(B[C \mapsto D])$ ;
- ako za iskazne formule A,  $B_1$ ,  $B_2$  i C važi  $A \neq C$  i formula A jednaka je  $(B_1 \wedge B_2)$ ,  $(B_1 \vee B_2)$ ,  $(B_1 \Rightarrow B_2)$  ili  $(B_1 \Leftrightarrow B_2)$ , onda je formula  $A[C \mapsto D]$  jednaka (redom)  $(B_1[C \mapsto D]) \wedge (B_2[C \mapsto D])$ ,  $(B_1[C \mapsto D]) \vee (B_2[C \mapsto D])$ ,  $(B_1[C \mapsto D]) \Rightarrow (B_2[C \mapsto D])$  ili  $(B_1[C \mapsto D]) \Leftrightarrow (B_2[C \mapsto D])$ .

**Primer 8.4.**  $((p \Rightarrow q) \vee r)[r \mapsto p] = (p \Rightarrow q) \vee p$   
 $((p \Rightarrow q) \vee r)[p \Rightarrow q \mapsto \neg p \vee q] = (\neg p \vee q) \vee r$   
 $(p \wedge (q \Rightarrow r))[q \mapsto r] = p \wedge (r \Rightarrow r)$   
 $((p \Rightarrow q) \vee r)[s \mapsto p] = ((p \Rightarrow q) \vee r)$   
 $((p \vee q) \wedge (p \vee r))[p \mapsto s] = ((s \vee q) \wedge (s \vee r))$

Ako iskazne formule predstavimo u računarskom programu u vidu stabla, onda opisanu zamenu možemo opisati rekurzivnom funkcijom (čiji su bazni slučajevi opisani prvim dvema stavkama definicije). Ta funkcija zamenjuje jedno podstablo formule (zapravo — sva podstabla koja odgovaraju istoj formuli) nekim drugim zadatim stablom.

### 8.1.2 Semantika iskazne logike

Semantika iskazne logike govori o *istinitosnim vrednostima* iskaznih formula. Istinitosna vrednost iskazne formule može biti 0 ili 1 (intuitivno — *netačno* i *tačno*).<sup>1</sup> Istinitosne vrednosti formula iskazne logike definišu se u skladu sa uobičajenim, svakodnevnim rasuđivanjem. Zbog toga, definicija semantike iskazne logike može da deluje i suvišno, ali ona je potrebna da bi baratanje formulama moglo da se opiše na precizan način (pogodan, na primer, za računarsku obradu).

Formula  $\perp$  ima istinitosnu vrednost 0, a formula  $\top$  ima istinitosnu vrednost 1. Istinitosna vrednost složenih (neatomičkih) formula zavisi samo od istinitosne vrednosti njenih potformula. Dakle, u krajnjoj instanci, istinitosna vrednost formule zavisi (samo) od istinitosnih vrednosti iskaznih promenljivih koje se u njoj pojavljuju. Funkcije koje pridružuju istinitosnu vrednost promenljivim (tj. funkcije  $v$  iz  $P$  u  $\{0, 1\}$ ) zovemo *valuacijama* (eng. *valuation, assignment*). *Interpretacija* (eng. *interpretation*) je proširenje valuacije: svaka valuacija  $v$  određuje jednu funkciju  $I_v$  koju zovemo *interpretacijom za valuaciju*  $v$  i koja pridružuje (jedinstvene) istinitosne vrednosti formulama (tj. preslikava skup iskaznih formula u skup  $\{0, 1\}$ ).

**Definicija 8.4** (Interpretacija). *Interpretacija  $I_v$  za valuaciju  $v$  definiše se na sledeći način:*

- $I_v(\top) = 1$  i  $I_v(\perp) = 0$ ;
- $I_v(p) = v(p)$ , za svaki element  $p$  skupa  $P$ ;
- $I_v(\neg A) = \begin{cases} 1, & \text{ako je } I_v(A) = 0 \\ 0, & \text{inače} \end{cases}$
- $I_v(A \wedge B) = \begin{cases} 1, & \text{ako je } I_v(A) = 1 \text{ i } I_v(B) = 1 \\ 0, & \text{inače} \end{cases}$
- $I_v(A \vee B) = \begin{cases} 0, & \text{ako je } I_v(A) = 0 \text{ i } I_v(B) = 0 \\ 1, & \text{inače} \end{cases}$
- $I_v(A \Rightarrow B) = \begin{cases} 0, & \text{ako je } I_v(A) = 1 \text{ i } I_v(B) = 0 \\ 1, & \text{inače} \end{cases}$
- $I_v(A \Leftrightarrow B) = \begin{cases} 1, & \text{ako je } I_v(A) = I_v(B) \\ 0, & \text{inače} \end{cases}$

Vrednost  $I_v(A)$  zovemo *istinitosnom vrednošću* iskazne formule  $A$  u interpretaciji  $I_v$  (ili u valuaciji  $v$ ). Ako za valuaciju  $v$  važi  $I_v(A) = 1$ , onda se za formulu  $A$  kaže da je *tačna* u interpretaciji  $I_v$ , a inače da je *netačna* u interpretaciji  $I_v$ .

**Primer 8.5.** Tvrđenje „Ana je zubar i Ana živi u Beogradu“ može da se opiše kao konjunkcija  $p \wedge q$ , gde iskazna promenljiva  $p$  odgovara elementarnom iskazu „Ana je zubar“, a iskazna promenljiva  $q$  odgovara elementarnom iskazu „Ana živi u Beogradu“. Može biti da je Ana zubar i može biti da Ana nije zubar, tj. istinitosna vrednost promenljive  $p$  može biti 0 ili 1. Ukoliko je  $v(p) = 0$ , tj. ukoliko Ana nije zubar, na

<sup>1</sup>Za istinitosne vrednosti namerno se ne uzimaju  $\perp$  i  $\top$ , već 0 i 1, da bi se jasno razlikovali svet sintakse i svet semantike iskaznih formula.

osnovu definicije semantike, za valuaciju  $v$  važi  $I_v(p) = 0$  i, dalje,  $I_v(p \wedge q) = 0$ , tj. tvrđenje „Ana je zubar i Ana živi u Beogradu“ nije tačno (bez obzira na istinitosnu vrednost iskaza „Ana živi u Beogradu“).

**Primer 8.6.** U primeru 8.2 (koji je nastavak primera 8.1), ako je  $v(p_{a1}) = 1$  i  $v(p_{a2}) = 1$ , onda za uslov  $p_{a1} \Rightarrow \neg p_{a2}$  važi  $I_v(p_{a1} \Rightarrow \neg p_{a2}) = 0$ . Ovo govori da ni u jednom rešenju ne mogu dame da budu i na polju  $a1$  i na polju  $a2$ .

**Definicija 8.5** (Zadovoljivost, valjanost, kontradiktornost, porecivost). Iskazna formula  $A$  je:

- zadovoljiva (eng. satisfiable), ako postoji valuacija  $v$  u kojoj je  $A$  tačna (i tada se kaže da je  $v$  model za  $A$ );
- valjana (eng. valid) ili tautologija (eng. tautology), ako je svaka valuacija  $v$  model za  $A$  i to zapisujemo  $\models A$ ;
- nezadovoljiva (eng. unsatisfiable) ili kontradikcija (eng. contradictory), ako ne postoji valuacija u kojoj je tačna;
- poreciva (eng. falsifiable), ako postoji valuacija u kojoj nije tačna.

**Primer 8.7.** Tvrđenje „Ana je zubar ili Ana nije zubar“ može da se opiše kao disjunkcija  $p \vee \neg p$ , gde iskazna promenljiva  $p$  odgovara elementarnom iskazu „Ana je zubar“. Može biti da je Ana zubar i može biti da Ana nije zubar, tj. istinitosna vrednost promenljive  $p$  može biti 0 ili 1. To su dve moguće relevantne valuacije za dato tvrđenje. Ukoliko je  $v(p) = 1$ , tj. ukoliko Ana jeste zubar, na osnovu definicije semantike važi  $I_v(p \vee \neg p) = 1$ , a ukoliko je  $v(p) = 0$ , tj. ukoliko Ana nije zubar, na osnovu definicije semantike važi  $I_v(\neg p) = 1$  i, dalje,  $I_v(p \vee \neg p) = 1$ . Dakle, formula  $p \vee \neg p$ , tj. tvrđenje „Ana je zubar ili Ana nije zubar“ je tautologija.

**Primer 8.8.** Ispitajte da li je iskazna formula  $p \Rightarrow p$  zadovoljiva i poreciva, a iskazna formula  $p \wedge \neg p$  kontradikcija.

**Primer 8.9.** Ako su iskazne formule  $A$  i  $A \Rightarrow B$  tautologije, onda je i  $B$  tautologija. Zaista, prepostavimo da su  $A$  i  $A \Rightarrow B$  tautologije i da postoji valuacija  $v$  takva da u interpretaciji  $I_v$  formula  $B$  nije tačna. Formula  $A$  je tautologija, pa je tačna i u interpretaciji  $I_v$ . Kako je u interpretaciji  $I_v$  formula  $A$  tačna, a formula  $B$  netačna, formula  $A \Rightarrow B$  u njoj nije tačna, što protivreći prepostavci da je  $A \Rightarrow B$  tautologija. Dakle, formula  $B$  je tačna za svaku valuaciju, pa je ona tautologija.

**Definicija 8.6** (Zadovoljivost i kontradiktornost skupa formula). Skup iskaznih formula  $\Gamma$  je

- zadovoljiv, ako postoji valuacija  $v$  u kojoj je svaka formula iz  $\Gamma$  tačna. Za takvu valuaciju  $v$  kaže se da je model za  $\Gamma$ .
- nezadovoljiv ili kontradiktoran, ako ne postoji valuacija  $v$  u kojoj je svaka formula iz  $\Gamma$  tačna.

**Primer 8.10.** Skup iskaznih formula  $\{p \Rightarrow q, p, \neg q\}$  je kontradiktoran (ali nijedan njegov pravi podskup nije kontradiktoran).

Naredna teorema govori da uslov koji rešenje nekog problema mora da zadovolji može da se razmatra ne samo kao skup svih pojedinačnih poduslova, već i kao konjunkcija formula koje odgovaraju tim poduslovima.

**Teorema 8.1.** *Valuacija  $v$  je model skupa formula  $\{A_1, \dots, A_n\}$  ako i samo ako je  $v$  model formule  $A_1 \wedge \dots \wedge A_n$ .*

## 8.2 Logičke posledice i logički ekvivalentne formule

Često je veoma važno pitanje da li je neki iskaz posledica nekih drugih iskaza. Ovo pitanje može se opisati u terminima pojma *logičke posledice*.

**Definicija 8.7** (Logička posledica i logička ekvivalencija). *Ako je svaki model za skup iskaznih formula  $\Gamma$  istovremeno i model za iskaznu formulu  $A$ , onda se kaže da je  $A$  logička posledica (eng. logical consequence) skupa  $\Gamma$  i piše se  $\Gamma \models A$ .*

*Ako je svaki model iskazne formule  $A$  model i iskazne formule  $B$  i obratno (tj. ako važi  $\{A\} \models B$  i  $\{B\} \models A$ ), onda se kaže se da su formule  $A$  i  $B$  logički ekvivalentne (eng. logically equivalent) i piše se  $A \equiv B$ .*

**Primer 8.11.** *Tvrđenje „Boban živi u Kruševcu“ je logička posledica skupa tvrđenja { „Boban živi u Beogradu ili Boban živi u Kruševcu“, „Boban ne živi u Beogradu“ }. Zaista, ako je valuacija  $v$  proizvoljan model formula „Boban živi u Beogradu ili Boban živi u Kruševcu“ i „Boban ne živi u Beogradu“, na osnovu definicije semantike, u interpretaciji  $I_v$  mora da je tačan iskaz „Boban živi u Kruševcu“.*

**Primer 8.12.** *Važi  $\{A, A \Rightarrow B\} \models B$ . Zaista, ako je valuacija  $v$  proizvoljan model formula  $A$  i  $A \Rightarrow B$ , onda kako je  $I_v(A) = 1$ , iz  $I_v(A \Rightarrow B) = 1$  sledi da ne može da važi  $I_v(B) = 0$ , pa mora da važi  $I_v(B) = 1$ , tj.  $v$  je model i za  $B$ , što je i trebalo dokazati.*

Ako ne važi  $\Gamma \models A$ , onda to zapisujemo  $\Gamma \not\models A$ .

Ako važi  $\{\} \models A$ , onda je svaka valuacija model za formulu  $A$ , tj.  $A$  je valjana formula, a važi i obratno — ako je  $A$  valjana formula, onda važi  $\{\} \models A$ . Zato umesto  $\{\} \models A$ , pišemo kraće  $\models A$ , baš kao što već zapisujemo valjane formule.

Ako je skup  $\Gamma$  kontradiktoran, onda je proizvoljna formula njegova logička posledica (naime, ako skup  $\Gamma$  nema modela, onda za proizvoljnu formulu  $A$  važi da je svaki model za  $\Gamma$  — a takvih nema — istovremeno i model za  $A$ ). Specijalno, svaka formula je logička posledica skupa  $\{\perp\}$ .

Ako je skup  $\Gamma$  konačan, onda umesto  $\{A_1, \dots, A_n\} \models B$  pišemo kraće  $A_1, \dots, A_n \models B$ . Koristeći teoremu 8.1, lako se može dokazati naredna teorema.

**Teorema 8.2.** *Važi  $A_1, \dots, A_n \models B$  ako i samo ako važi  $A_1 \wedge \dots \wedge A_n \models B$ .*

Ako važi  $A \equiv B$ , onda u bilo kojoj valuaciji formule  $A$  i  $B$  imaju jednake vrednosti. Tvrđenja oblika  $A \equiv B$  zovemo *logičkim ekvivalentcijama*. Relacija  $\equiv$  je, očigledno, relacija ekvivalencije nad skupom iskaznih formula. U razmatranju skupova uslova, bilo koji uslov očigledno može biti zamenjen nekim njemu logički ekvivalentnim uslovom jer time neće biti promenjen skup modela celokupnog uslova.

**Primer 8.13.** *Za formule  $p_{a1} \Rightarrow \neg p_{a3}$  i  $p_{a3} \Rightarrow \neg p_{a1}$  iz primera 8.1, može se pokazati da važi:  $p_{a1} \Rightarrow \neg p_{a3} \equiv p_{a3} \Rightarrow \neg p_{a1}$ . To govori da nije potrebno da u skupu uslova postoje obe formule, dovoljno je zadržati jednu od njih. Isto važi i za druge analogne parove formule, te je dovoljno razmatrati sledeći skup formula:*

$$p_{a1} \vee p_{a2} \vee p_{a3}, p_{b1} \vee p_{b2} \vee p_{b3}, p_{c1} \vee p_{c2} \vee p_{c3},$$

$$p_{a1} \Rightarrow \neg p_{a2}, p_{a1} \Rightarrow \neg p_{a3}, p_{a2} \Rightarrow \neg p_{a3},$$

$$p_{b1} \Rightarrow \neg p_{b2}, p_{b1} \Rightarrow \neg p_{b3}, p_{b2} \Rightarrow \neg p_{b3},$$

$$p_{c1} \Rightarrow \neg p_{c2}, p_{c1} \Rightarrow \neg p_{c3}, p_{c2} \Rightarrow \neg p_{c3},$$

$$p_{a1} \Rightarrow \neg p_{b1}, p_{a1} \Rightarrow \neg p_{c1}, p_{b1} \Rightarrow \neg p_{c1},$$

$$p_{a2} \Rightarrow \neg p_{b2}, p_{a2} \Rightarrow \neg p_{c2}, p_{b2} \Rightarrow \neg p_{c2},$$

$p_{a3} \Rightarrow \neg p_{b3}$ ,  $p_{a3} \Rightarrow \neg p_{c3}$ ,  $p_{b3} \Rightarrow \neg p_{c3}$ ,  
 $p_{a3} \Rightarrow \neg p_{b2}$ ,  $p_{a3} \Rightarrow \neg p_{c1}$ ,  $p_{b2} \Rightarrow \neg p_{c1}$ ,  
 $p_{a2} \Rightarrow \neg p_{b1}$ ,  $p_{b3} \Rightarrow \neg p_{c2}$ ,  
 $p_{a1} \Rightarrow \neg p_{b2}$ ,  $p_{a1} \Rightarrow \neg p_{c3}$ ,  $p_{b2} \Rightarrow \neg p_{c3}$ ,  
 $p_{a2} \Rightarrow \neg p_{b3}$ ,  $p_{b1} \Rightarrow \neg p_{c2}$ .

**Primer 8.14.** Neke od logičkih ekvivalencija (ili, preciznije, neke od shema logičkih ekvivalencija) su:

$\neg\neg A \equiv A$	zakon dvojne negacije
$A \vee \neg A \equiv \top$	zakon isključenja trećeg
$A \wedge A \equiv A$	zakon idempotencije za $\wedge$
$A \vee A \equiv A$	zakon idempotencije za $\vee$
$A \wedge B \equiv B \wedge A$	zakon komutativnosti za $\wedge$
$A \vee B \equiv B \vee A$	zakon komutativnosti za $\vee$
$A \Leftrightarrow B \equiv B \Leftrightarrow A$	zakon komutativnosti za $\Leftrightarrow$
$A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C$	zakon asocijativnosti za $\wedge$
$A \vee (B \vee C) \equiv (A \vee B) \vee C$	zakon asocijativnosti za $\vee$
$A \Leftrightarrow (B \Leftrightarrow C) \equiv (A \Leftrightarrow B) \Leftrightarrow C$	zakon asocijativnosti za $\Leftrightarrow$
$A \wedge (A \vee B) \equiv A$	zakon apsorpcije
$A \vee (A \wedge B) \equiv A$	zakon apsorpcije
$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$	zakon distributivnosti $\wedge$ u odnosu na $\vee$
$(B \vee C) \wedge A \equiv (B \wedge A) \vee (C \wedge A)$	zakon distributivnosti $\wedge$ u odnosu na $\vee$
$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$	zakon distributivnosti $\vee$ u odnosu na $\wedge$
$(B \wedge C) \vee A \equiv (B \vee A) \wedge (C \vee A)$	zakon distributivnosti $\vee$ u odnosu na $\wedge$
$\neg(A \wedge B) \equiv \neg A \vee \neg B$	De Morganov zakon
$\neg(A \vee B) \equiv \neg A \wedge \neg B$	De Morganov zakon
$A \wedge \top \equiv A$	zakon konjunkcije sa tautologijom
$A \vee \top \equiv \top$	zakon disjunkcije sa tautologijom
$A \wedge \perp \equiv \perp$	zakon konjunkcije sa kontradikcijom
$A \vee \perp \equiv A$	zakon disjunkcije sa kontradikcijom

Logičke ekvivalencije navedene u primeru 8.14, pokazuju, između ostalog, da su konjunkcija i disjunkcija komutativni i asocijativni veznici. Zato možemo smatrati da konjunkcija (i disjunkcija) mogu da povezuju više od dve formule, pri čemu ne moramo da vodimo računa o njihovom poretku. Svaki član uopštene konjunkcije zovemo *konjunkt*, a svaki član uopštene disjunkcije zovemo *disjunkt*. Disjunkciju više literalata (pri čemu njihov poredak nije bitan) zovemo *klausa* (eng. *clause*). Klausa je *jedinična* ako sadrži samo jedan literal.

Naglasimo da, na primer,  $B \models A$  i  $B \equiv A$  nisu formule iskazne logike, nego su to *formule koje govore o iskaznim formulama*, pa ih zato zovemo *meta formule* (a iskazne formule čine *objektne formule, objektni nivo*). Naredna teorema povezuje meta i objektni nivo i govori o tome kako ispitivanje da li su neke dve formule logički ekvivalentne i da li je jedna logička posledica druge može da se svede na objektni nivo i na problem ispitivanja da li je neka formula tautologija.

### Teorema 8.3.

Važi  $A \models B$  ako i samo ako je iskazna formula  $A \Rightarrow B$  tautologija.

Važi  $A \equiv B$  ako i samo ako je iskazna formula  $A \Leftrightarrow B$  tautologija.

Na osnovu prethodne dve teoreme sledi da važi  $A_1, \dots, A_n \models B$  ako i samo ako je formula  $A_1 \wedge \dots \wedge A_n \Rightarrow B$  tautologija.

**Primer 8.15.** Ako važi „kiša pada“ i „ako kiša pada, onda će meč biti otkazan“, onda važi i „meč će biti otkazan“. Naime, iskaz „meč će biti otkazan“ je logička posledica skupa iskaza { „kiša pada“, „ako kiša pada, onda će meč biti otkazan“ } ako i samo ako je formula

„kiša pada“  $\wedge$  „ako kiša pada, onda će meč biti otkazan“  $\Rightarrow$  „meč će biti otkazan“ tautologija (što jeste ispunjeno).

Već je rečeno da u razmatranju skupova uslova, svaki uslov može biti zamenjen nekim njemu logički ekvivalentnim uslovom (i time neće biti promenjen skup modela celokupnog uslova). Naredna teorema tvrdi i sledeće: ako se u formuli  $A$  neka njena potformula zameni njoj logički ekvivalentnom formulom, biće dobijena formula koja je logički ekvivalentna formuli  $A$ . Ova teorema opravdava *transformisanje* iskazne formule u neki oblik pogodan za, na primer, ispitivanje zadovoljivosti.

**Teorema 8.4** (Teorema o zameni). *Ako je  $C \equiv D$ , onda je  $A[C \mapsto D] \equiv A$ .*

**Primer 8.16.** *Kako je  $q \Rightarrow r \equiv \neg q \vee r$ , važi  $(p \wedge (q \Rightarrow r))[q \Rightarrow r \mapsto \neg q \vee r] = p \wedge (\neg q \vee r) \equiv p \wedge (q \Rightarrow r)$ .*

**Primer 8.17.** *Kako je  $\neg(\neg p \wedge \neg q) \equiv p \vee q$ , važi da su formule  $\neg(\neg p \wedge \neg q) \vee r$  i  $p \vee q \vee r$  logički ekvivalentne.*

*Odatle dalje sledi da se u C programu red*

*if (!(!a && !b) || c)*

*može zameniti sledećim redom:*

*if (a || b || c)*

*Primetimo da zagrade oko izraza a || b nisu neophodne jer je disjunkcija asocijativni veznik.*

### 8.3 Ispitivanje zadovoljivosti i valjanosti u iskaznoj logici

U praktičnim primenama, onda kada je neki cilj formulisan na jeziku iskazne logike, centralni problem postaje ispitivanje da li je neka iskazna formula tautologija, da li je zadovoljiva i slično. Početno pitanje je da li su ti problemi uopšte odlučivi, tj. da li postoje algoritmi koji *uvek* mogu da daju odgovor na njih. Na primer, pitanje je da li postoji algoritam koji za svaku formulu koja je tautologija može da utvrdi da jeste tautologija, a za svaku formulu koja nije tautologija može da utvrdi da nije tautologija. Jednostavno se, kao što je pokazano u narednom poglavljju, pokazuje da takav algoritam postoji, pa je problem ispitivanja tautoličnosti odlučiv. Dodatno, odlučivi su i problemi ispitivanja zadovoljivosti, porecivosti i kontradiktornosti. Štaviše, ovi problemi su blisko povezani i bilo koja tri mogu se lako svesti na četvrti. To znači da je dovoljno da imamo efikasan algoritam za jedan od ova četiri problema. U nastavku ćemo se fokusirati na rešavanje problema zadovoljivosti. Algoritam za ispitivanje zadovoljivosti onda možemo iskoristiti i za preostala tri problema: formula  $A$  je tautologija akko  $\neg A$  nije zadovoljiva,  $A$  je poreciva akko je  $\neg A$  zadovoljiva, a  $A$  je kontradikcija akko  $A$  nije zadovoljiva.

#### 8.3.1 Istinitosne tablice i odlučivost problema valjanosti i zadovoljivosti

Na osnovu definicije interpretacije, može se konstruisati istinitosna tablica za proizvoljnu iskaznu formulu. Istinitosne tablice pogodne su za ispitivanje i valjanosti i zadovoljivosti i nezadovoljivosti i porecivosti. U istinitosnoj tablici za neku formulu, svakoj vrsti odgovara jedna valuacija iskaznih slova koja se pojavljuju u toj formuli. Svakoj koloni odgovara jedna potformula te formule. Ukoliko iskazna formula  $A$  sadrži iskazne promenljive  $p_1, p_2, \dots, p_n$ , onda istinitosna tablica treba da sadrži sve moguće valuatorije za ovaj skup promenljivih (valuatorije za druge promenljive nisu relevantne). Takođe, valuatorije imaju  $2^n$ . U zavisnosti od vrednosti iskaznih promenljivih, izračunavaju se vrednosti potformula razmatrane iskazne formule, sve do sâme te formule. Ako su u koloni koja odgovara sâmoj iskaznoj formuli sve vrednosti jednake 1, onda je formula tautologija. Ako je bar jedna vrednost jednaka 1, formula je zadovoljiva. Ako je bar jedna vrednost jednaka 0, formula je poreciva. Ako su sve vrednosti jednake 0, formula je kontradikcija. Ovo pokazuje da su problemi ispitivanja valjanosti, zadovoljivosti, nezadovoljivosti i porecivosti odlučivi problemi, tj. postoje algoritmi koji ih mogu rešiti. Međutim, opisani algoritam, zasnovan na istinitosnim tablicama, naivan je i potpuno neupotrebljiv za realne probleme. Za rešavanje teških realnih problema potrebno je razviti znatno efikasnije algoritme.

**Primer 8.18.** *Iskaznoj formuli  $(\neg q \Rightarrow \neg p) \Rightarrow (p \Rightarrow q)$  odgovara sledeća istinitosna tablica:*

$p$	$q$	$\neg q$	$\neg p$	$\neg q \Rightarrow \neg p$	$p \Rightarrow q$	$(\neg q \Rightarrow \neg p) \Rightarrow (p \Rightarrow q)$
0	0	1	1	1	1	1
0	1	0	1	1	1	1
1	0	1	0	0	0	1
1	1	0	0	1	1	1

Dakle, data formula je zadovoljiva i valjana. Ona nije poreciva i nije kontradikcija.

**Primer 8.19.** U primeru 8.1, za tablu dimenzije  $3 \times 3$ , razmatra se skup formula nad 9 iskaznih promenljivih, te bi odgovarajuća istinitosna tablica imala  $2^9 = 512$  vrsta.

**Primer 8.20.** Date su tri kutije, zna se da je tačno u jednoj od njih zlato. Na njima piše:

na kutiji 1: „zlato nije ovde“

na kutiji 2: „zlato nije ovde“

na kutiji 3: „zlato je u kutiji 2“

Ako se zna da je tačno jedna tvrdnja tačna, treba pogoditi gde je zlato.

Neka iskazne promenljive  $b_1, b_2$  i  $b_3$  odgovaraju tvrdnjama „zlato je u kutiji 1“, „zlato je u kutiji 2“ i „zlato je u kutiji 3“.

Uslov da je tačno u jednoj od njih zlato može se opisati sledećom formulom:

$$A = (b_1 \wedge \neg b_2 \wedge \neg b_3) \vee (\neg b_1 \wedge b_2 \wedge \neg b_3) \vee (\neg b_1 \wedge \neg b_2 \wedge b_3)$$

Uslov da je tačno jedna ispisana tvrdnja tačna, može se opisati sledećom formulom:

$$B = (\neg b_1 \wedge \neg \neg b_2 \wedge \neg b_3) \vee (\neg \neg b_1 \wedge \neg b_2 \wedge \neg b_3) \vee (\neg \neg b_1 \wedge \neg \neg b_2 \wedge b_3)$$

Potrebno je pronaći valuaciju u kojoj su obe formule  $A$  i  $B$  tačne:

$b_1$	$b_2$	$b_3$	$A$	$B$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

Postoji samo jedna valuacija v u kojoj su obe formule tačne. U njoj važi  $I_v(b_1) = 1$ , te je zlato u prvoj kutiji.

### 8.3.2 Normalne forme i potpuni skupovi veznika

Napredne procedure za ispitivanje valjanosti ili zadovoljivosti se, zbog jednostavnosti i veće efikasnosti, obično definišu samo za neke specifične vrste iskaznih formula, za formule koje su u nekoj specifičnoj formi.

**Definicija 8.8** (Konjunktivna normalna forma). Iskazna formula je u konjunktivnoj normalnoj formi (KNF) ako je oblika

$$A_1 \wedge A_2 \wedge \dots \wedge A_n$$

pri čemu je svaka od formula  $A_i$  ( $1 \leq i \leq n$ ) klauza (tj. disjunkcija literala).

**Definicija 8.9** (Disjunktivna normalna forma). Iskazna formula je u disjunktivnoj normalnoj formi (DNF) ako je oblika

$$A_1 \vee A_2 \vee \dots \vee A_n$$

pri čemu je svaka od formula  $A_i$  ( $1 \leq i \leq n$ ) konjunkcija literala.

Ako je iskazna formula  $A$  logički ekvivalentna iskaznoj formuli  $B$  i iskazna formula  $B$  je u konjunktivnoj (disjunktivnoj) normalnoj formi, onda se kaže da je formula  $B$  konjunktivna (disjunktivna) normalna forma formule  $A$ . Jedna iskazna formula može da ima više različitih konjunktivnih (disjunktivnih) normalnih formi (na primer, i formula  $(p \vee r) \wedge (q \vee r) \wedge (p \vee s) \wedge (q \vee s)$  i formula  $(s \vee q) \wedge (p \vee r) \wedge (q \vee r) \wedge (p \vee s) \wedge (p \vee \neg p)$  su konjunktivne normalne forme formule  $(p \wedge q) \vee (r \wedge s)$ ). Slično, jedna formula koja je u konjunktivnoj normalnoj formi može biti konjunktivna normalna forma za više iskaznih formula.

Korišćenjem pogodnih ekvivalencija, svaka iskazna formula može biti transformisana u svoju konjunktivnu (disjunktivnu) normalnu formu. Transformisanje iskazne formule u konjunktivnu normalnu formu može biti opisano algoritmom prikazanim na slici 8.2. Kada se govori o „primeni neke logičke ekvivalencije“ misli se na korišćenje logičke ekvivalencije na osnovu teoreme o zameni (teorema 8.4).

### Algoritam: KNF

**Ulaz:** Iskazna formula  $F$

**Izlaz:** Konjunktivna normalna forma formule  $F$

- 1: **dok god** je to moguće **radi**
- 2: primeni logičku ekvivalenciju (eliminiši veznik  $\Leftrightarrow$ ):  

$$A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A).$$
- 3: **dok god** je to moguće **radi**
- 4: primeni logičku ekvivalenciju (eliminiši veznik  $\Rightarrow$ ):  

$$A \Rightarrow B \equiv \neg A \vee B.$$
- 5: **dok god** je to moguće **radi**
- 6: primeni neku od logičkih ekvivalencija:  

$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$
  

$$\neg(A \vee B) \equiv \neg A \wedge \neg B.$$
- 7: **dok god** je to moguće **radi**
- 8: primeni logičku ekvivalenciju (eliminiši višestruke veznike  $\neg$ ):  

$$\neg\neg A \equiv A.$$
- 9: **dok god** je to moguće **radi**
- 10: primeni neku od logičkih ekvivalencija:  

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$
  

$$(B \wedge C) \vee A \equiv (B \vee A) \wedge (C \vee A).$$

Slika 8.2: Algoritam KNF.

Zaustavljanje algoritma KNF može se dokazati korišćenjem pogodno odabrane mere zaustavljanja.<sup>2</sup> Za neke pojedinačne korake, može se dokazati da se zaustavljaju korišćenjem jednostavnih mera — na primer, za prvi korak algoritma, kao mera se može koristiti broj veznika  $\Leftrightarrow$  u formuli. Izlazna formula logički je ekvivalentna ulaznoj formuli na osnovu teoreme o zameni (teorema 8.4) i činjenice da se u algoritmu koriste samo logičke ekvivalencije.

**Teorema 8.5** (Korektnost algoritma KNF). *Algoritam KNF se zaustavlja i ima sledeće svojstvo: ako je  $F$  ulazna formula, onda je izlazna formula  $F'$  u konjunktivnoj normalnoj formi i logički je ekvivalentna sa  $F$ .*

Transformisanje formule u disjunktivnu normalnu formu opisuje se algoritmom analognim algoritmu KNF.

Algoritmom KNF proizvoljna iskazna formula može se transformisati u formulu koja ne sadrži veznike  $\Leftrightarrow$  i  $\Rightarrow$ . Dobijena formula sadržaće, dakle, samo veznike  $\neg$ ,  $\wedge$  i  $\vee$ . Kaže se da je skup veznika  $\{\neg, \wedge, \vee\}$  potpun, jer je svaka iskazna formula logički ekvivalentna nekoj iskaznoj formuli nad samo ova tri veznika i bez logičkih konstanti  $\top$  i  $\perp$ . Štaviše, zahvaljujući logičkoj ekvivalenciji  $A \vee B \equiv \neg(\neg A \wedge \neg B)$ , može se dokazati i da je skup  $\{\neg, \wedge\}$  potpun. Analogno se može dokazati da je potpun i skup  $\{\neg, \vee\}$ . Postoje i (tačno) dva jednočlana

<sup>2</sup> U cilju dokazivanja zaustavljanja postupka transformisanja formule u konjunktivnu normalnu formu definiše se preslikavanje  $\tau$  (mera zaustavljanja) iz skupa iskaznih formula u skup prirodnih brojeva:

$$\begin{aligned}\tau(A) &= 2 \quad (\text{gde je } A \text{ atomička formula}) \\ \tau(\neg A) &= 2^{\tau(A)} \\ \tau(A \vee B) &= \tau(A) \cdot \tau(B) \\ \tau(A \wedge B) &= \tau(A) + \tau(B) + 1\end{aligned}$$

Može se jednostavno dokazati da je vrednost  $\tau(F_2)$  uvek manja od  $\tau(F_1)$  ako je formula  $F_2$  dobijena primenom nekog koraka algoritma na formulu  $F_1$ . Na primer, važi da je  $\tau(\neg(A \vee B)) = 2^{\tau(A \vee B)} = 2^{\tau(A) \cdot \tau(B)}$  veće od  $\tau(\neg A \wedge \neg B) = \tau(\neg A) + \tau(\neg B) + 1 = 2^{\tau(A)} + 2^{\tau(B)} + 1$ . Odатле sledi da se postupak transformisanja proizvoljne formule u konjunktivnu normalnu formu zaustavlja za proizvoljnu ulaznu formulu  $F$  (jer ne postoji beskonačan strogo opadajući niz prirodnih brojeva čiji je prvi element  $\tau(F)$ ).

vrsta formule	rezultujuće klauze
$A \Leftrightarrow (\neg B)$	$(p_A \vee p_B) \wedge (\neg p_A \vee \neg p_B)$
$A \Leftrightarrow (B \wedge C)$	$(p_A \vee \neg p_B \vee \neg p_C) \wedge (\neg p_A \vee p_B) \wedge (\neg p_A \vee p_C)$
$A \Leftrightarrow (B \vee C)$	$(\neg p_A \vee p_B \vee p_C) \wedge (p_A \vee \neg p_B) \wedge (p_A \vee \neg p_C)$
$A \Leftrightarrow (B \Rightarrow C)$	$(\neg p_A \vee \neg p_B \vee p_C) \wedge (p_A \vee p_B) \wedge (p_A \vee \neg p_C)$
$A \Leftrightarrow (B \Leftrightarrow C)$	$(\neg p_A \vee \neg p_B \vee p_C) \wedge (\neg p_A \vee p_B \vee \neg p_C) \wedge (p_A \vee p_B \vee p_C) \wedge (p_A \vee \neg p_B \vee \neg p_C)$

Tabela 8.1: Pravila za Cejtinovu transformaciju.

potpuna skupa veznika:  $\{\downarrow\}$  i  $\{\uparrow\}$ , pri čemu su veznici  $\downarrow$  (nil ili Lukašijevičeva funkcija) i  $\uparrow$  (ni ili Šeferova funkcija) definisani na sledeći način:  $A \downarrow B$  je jednako  $\neg(A \vee B)$ , a  $A \uparrow B$  je jednako  $\neg(A \wedge B)$ . Lako se pokazuje da je  $\neg A \equiv (A \downarrow A)$  i  $A \wedge B \equiv ((A \downarrow A) \downarrow (B \downarrow B))$ , pa kako je skup veznika  $\{\neg, \wedge\}$  potpun, sledi da je potpun i skup  $\{\downarrow\}$ . Analogno važi i za skup  $\{\uparrow\}$ .

Transformisanje formule u njenu konjunktivnu normalnu formu može da dâ formulu čija je složenost eksponencijalna u odnosu na složenost polazne formule. Na primer, transformisanjem formule

$$(A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \dots \vee (A_n \wedge B_n)$$

(koja ima  $n$  disjunkata) u njenu konjunktivnu normalnu formu, dobija se formula koja ima  $2^n$  konjunkta.

Zbog potencijalno ogromne izlazne formule, umesto algoritma KNF, u praksi se najčešće koristi Cejtinovo kodiranje – koje je linearno i u smislu vremena i u smislu prostora, ali uvodi dodatne promenljive, te zato rezultujuća formula nije logički ekvivalentna polaznoj već samo *slabo ekvivalentna*: početna formula je zadovoljiva ako i samo ako je zadovoljiva rezultujuća formula. To je za primene obično dovoljno dobro i, štaviše, iz modela za rezultujuću formulu (ukoliko oni postoje) mogu se rekonstruisati modeli za polaznu formulu. Smatraćemo da su iz formule, korišćenjem pogodnih logičkih ekvivalencija, eliminisane sve konstante  $\top$  i  $\perp$ . Cejtinova transformacija može se opisati na sledeći način: Neka  $Sub(F)$  označava skup svih potformula formule  $F$ . Za svaku formulu  $A$  iz  $Sub(F)$  koja nije iskazna promenljiva, uvodi se nova iskazna promenljiva (*definicione promenljive*)  $p_A$ . Ako je  $A$  iskazna promenljiva, onda  $p_A$  označava samu formulu  $A$  (i tada se  $p_A$  naziva *osnovna promenljiva*). Formula  $F$  se najpre transformiše u sledeću formulu (gde  $\star$  označava binarni iskazni veznik iz skupa binarnih veznika koji se pojavljuju u  $F$ ):

$$p_F \wedge \bigwedge_{\substack{A \in Sub(F) \\ A=B \star C}} (p_A \Leftrightarrow (p_B \star p_C)) \wedge \bigwedge_{\substack{A \in Sub(F) \\ A=\neg B}} (p_A \Leftrightarrow \neg p_B)$$

Lako se može dokazati da je navedena formula slabo ekvivalentna sa formulom  $F$ . Na kraju, navedena formula se trivijalno transformiše u KNF oblik primenom pravila iz tabele 8.1. Svaki konjunkt se transformiše u KNF formulu sa najviše četiri klauze, od kojih svaka ima najviše tri literala.

Cejtinova transformacija daje formulu čija veličina je linearna u odnosu na veličinu polazne formule. Preciznije, ako polazna formula sadrži  $n$  logičkih veznika, onda će izlazna formula sadržati najviše  $4n$  klauza, od kojih svaka ima najviše tri literala. To se može pokazati jednostavnim induktivnim argumentom.

**Primer 8.21.** Data je iskazna formula  $(p \wedge (q \wedge r)) \vee ((q \wedge r) \wedge \neg p)$ . Definicione promenljive  $p_4, p_5, p_6, p_7, p_8$  uvode se na sledeći način:

$$\underbrace{(p \wedge \underbrace{(q \wedge r)}_{p_4}) \vee \underbrace{((q \wedge r) \wedge \neg p)}_{\substack{p_7 \\ p_5}}}_{p_8}$$

Međuoblik za Cejtinovu formu je onda:

$$p_8 \wedge (p_8 \Leftrightarrow (p_6 \vee p_7)) \wedge (p_6 \Leftrightarrow (p \wedge p_4)) \wedge (p_7 \Leftrightarrow (p_4 \wedge p_5)) \wedge (p_4 \Leftrightarrow (q \wedge r)) \wedge (p_5 \Leftrightarrow \neg p)$$

Konačno, izlazna KNF formula je:

$$\begin{aligned} & p_8 \wedge \\ & (\neg p_8 \vee p_6 \vee p_7) \wedge (p_8 \vee \neg p_6) \wedge (p_8 \vee \neg p_7) \wedge \\ & (p_6 \vee \neg p \vee \neg p_4) \wedge (\neg p_6 \vee p) \wedge (\neg p_6 \vee p_4) \wedge \\ & (p_7 \vee \neg p_4 \vee \neg p_5) \wedge (\neg p_7 \vee p_4) \wedge (\neg p_7 \vee p_5) \wedge \end{aligned}$$

$$(p_4 \vee \neg q \vee \neg r) \wedge (\neg p_4 \vee q) \wedge (\neg p_4 \vee r) \wedge \\ (p_5 \vee p) \wedge (\neg p_5 \vee \neg p)$$

Problem sa Cejtinovom transformacijom je u tome što ona uvodi mnogo novih promenljivih. Postoje raznovrsne tehnike za smanjivanje broja promenljivih i broja kluaza.

### 8.3.3 Problem SAT i DPLL procedura

Za svaku iskaznu formulu postoji njena konjunktivna normalna forma i većina primena iskazne logike svodi se na ispitivanje zadovoljivosti neke formule koja je u tom, KNF obliku.

Problem ispitivanja zadovoljivosti date iskazne formule u KNF obliku označava se sa SAT (od engleskog *satisfiability problem* – problem zadovoljivosti). Programi koji rešavaju instance SAT problema zovu se SAT rešavači. SAT problem je NP-kompletan i on ima ogroman i teorijski i praktični značaj. Problem ispitivanja nezadovoljivosti date iskazne formule u KNF obliku je co-NP-kompletan.<sup>3</sup>

S obzirom na to da se još uvek ne zna da li su klase P i NP problema jednake, to znači da se još uvek ne zna da li postoji algoritam za ispitivanje zadovoljivosti iskazne formule koji je polinomske složenosti.<sup>4</sup> Kako je opšte uverenje da su klase problema P i NP različite, veruje se i da ne postoji algoritam polinomske složenosti za rešavanje SAT problema.

Problem ispitivanja zadovoljivosti formula u DNF obliku suštinski je drugačiji od ispitivanja zadovoljivosti formula u KNF obliku. Drugi je NP-kompletan, a prvi je trivijalan (dovoljno je razmatrati zadovoljivost disjunkata pojedinačno, a to se svodi na provere da li u disjunktu postoji logička konstanta  $\perp$  ili literal i njegova negacija) i pripada klasi P (podrazumeva se složenost u terminima broja bitova potrebnih za zapis ulazne formule). Ipak, svodenje problema SAT na problem ispitivanja zadovoljivosti DNF formule nije, u opštem slučaju, razuman put za rešavanje problema SAT, zbog kompleksnosti same transformacije, na primer, formula koje su već u KNF obliku u DNF oblik. Dodajmo da problem ispitivanja tautoličnosti formule u KNF obliku pripada klasi P (dovoljno je razmatrati tautoličnost konjunkata pojedinačno, a to se svodi na provere da li u konjunktu postoji logička konstanta  $\top$  ili literal i njegova negacija), a da je problem ispitivanja tautoličnosti formule u DNF obliku co-NP-kompletan problem.

Dejvis–Patnam–Logman–Lavlendova ili DPLL procedura<sup>5</sup> je procedura za ispitivanje zadovoljivosti iskaznih formula u KNF obliku, to jest, procedura za rešavanje instanci SAT problema. Ulazna formula je konjunkcija kluaza. Pri tome (kako su konjunkcija i disjunkcija komutativne i asocijativne) nije bitan poredak tih kluaza niti je u bilo kojoj od tih kluaza bitan poredak literala, te se ulazna formula može smatrati skupom (ili, preciznije, multiskupom)<sup>6</sup> kluaza, od kojih se svaka može smatrati skupom (ili, preciznije, multiskupom) literala. Ipak, radi određenosti rada algoritma, smatraćemo da je skup (odnosno multiskup) kluaza uređen.

U proceduri se podrazumevaju sledeće konvencije:

- prazan skup kluza je zadovoljiv;
- kluza koja ne sadrži nijedan literal (zvaćemo je *prazna kluza*) je nezadovoljiva i formula koja sadrži praznu kluazu je nezadovoljiva.

DPLL procedura prikazana je na slici 8.3, a njena svojstva daje teorema 8.6.

**Teorema 8.6** (Korektnost DPLL procedure). *Za svaku iskaznu formulu DPLL procedura se zaustavlja i vraća odgovor DA ako i samo ako je polazna formula zadovoljiva.*

**Primer 8.22.** *Formula iz primera 8.13 trivijalno se može transformisati u KNF oblik i na nju se može primeniti DPLL procedura. Prvo pravilo koje je primenljivo je split i može da se primeni, na primer, na promenljivu  $p_{a1}$ . U prvoj grani koja se razmatra  $p_{a1}$  se zamenjuje sa  $\top$  (što odgovara pridruživanju*

<sup>3</sup> Problem odlučivanja  $\mathcal{X}$  je komplementan problemu odlučivanja  $\mathcal{Y}$  ako za svaku instancu za koju problem  $\mathcal{Y}$  daje odgovor „da“, problem  $\mathcal{X}$  daje odgovor „ne“. Problem odlučivanja  $\mathcal{X}$  pripada klasi co-NP ako njemu komplementan problem pripada klasi NP. Na primer, problem zadovoljivosti za iskazne formule u KNF obliku pripada klasi NP, pa problem nezadovoljivosti za iskazne formule u KNF obliku pripada klasi co-NP. Nije poznato da li su klase NP i co-NP jednake. Problem koji pripada klasi co-NP je co-NP-kompletan ako se svaki problem iz klase co-NP može svesti na njega u polinomskom vremenu.

<sup>4</sup>Kada se govori o klasama složenosti, obično se podrazumeva da se složenost algoritma izražava u terminima broja bitova potrebnih za zapisivanje ulaza.

<sup>5</sup>Prva verzija procedure čiji su autori Dejvis (Martin Davis) i Patnam (Hilary Putnam), unapredena je dve godine kasnije u radu Dejvisa, Logmana (Georg Logemann) i Lavlenda (Donald Loveland), pa otuda naziv DPLL.

<sup>6</sup> Neformalno, multiskup je skup u kojem se elementi mogu pojavljivati više puta.

**Algoritam:** DPLL

**Ulaz:** Multiskup klauza  $D$  ( $D = \{C_1, C_2, \dots, C_n\}$ )

**Izlaz:**  $DA$ , ako je multiskup  $D$  zadovoljiv,  $NE$ , inače;

- 1: **ako**  $D$  je prazan **onda**  
2:     vrati  $DA$ ;
- 3: zameni sve literale  $\neg\perp$  sa  $\top$  i zameni sve literale  $\neg\top$  sa  $\perp$ ;
- 4: obriši sve literale jednake  $\perp$ ;
- 5: **ako**  $D$  sadrži praznu klauzu **onda**  
6:     vrati  $NE$ ;
- 7: {Korak tautology:}
- 8: **ako** neka klauza  $C_i$  sadrži  $\top$  ili sadrži neki literal i njegovu negaciju **onda**  
9:     vrati vrednost koju vraća  $DPLL(D \setminus C_i)$ ;
- 10: {Korak unit propagation:}
- 11: **ako** neka klauza je jedinična i jednaka nekom iskaznom slovu  $p$  **onda**  
12:     vrati vrednost koju vraća  $DPLL(D[p \mapsto \top])$ ;
- 13: **ako** neka klauza je jednična i jednaka  $\neg p$ , za neko iskazno slovo  $p$  **onda**  
14:     vrati vrednost koju vraća  $DPLL(D[p \mapsto \perp])$ ;
- 15: {Korak pure literal:}
- 16: **ako**  $D$  sadrži literal  $p$  (gde je  $p$  neko iskazno slovo), ali ne i  $\neg p$  **onda**  
17:     vrati vrednost koju vraća  $DPLL(D[p \mapsto \top])$  ;
- 18: **ako**  $D$  sadrži literal  $\neg p$  (gde je  $p$  neko iskazno slovo), ali ne i  $p$  **onda**  
19:     vrati vrednost koju vraća  $DPLL(D[p \mapsto \perp])$ ;
- 20: {Korak split:}
- 21: **ako**  $DPLL(D[p \mapsto \top])$  (gde je  $p$  jedno od iskaznih slova koja se javljaju u  $D$ ) vraća  $DA$  **onda**  
22:     vrati  $DA$ ;
- 23: **inače**  
24:     vrati vrednost koju vraća  $DPLL(D[p \mapsto \perp])$ .

Slika 8.3: DPLL procedura.

vrednosti tačno) i u narednim koracima se, primenom pravila unit propagation promenljive  $p_{a2}, p_{a3}, p_{b1}, p_{c1}, p_{b2}, p_{c3}$  zamenjuju sa  $\perp$ . Zatim se promenljive  $p_{b3}$  i  $p_{c2}$  zamenjuju sa  $\top$ , nakon čega klauza  $\neg p_{b3} \vee \neg p_{c2}$  postaje prazna. Slično se dešava i u grani u kojoj se  $p_{a1}$  zamenjuje sa  $\perp$ , te procedura vraća odgovor  $NE$ , što znači da ne postoji rešenje problema n dama za  $n = 3$ .

DPLL procedura je u najgorem slučaju eksponencijalne vremenske složenosti po broju iskaznih promenljivih u formuli, usled rekurzivne primene *split* pravila. Eksponencijalne složenosti su i svi drugi do sada poznati algoritmi za ispitivanje zadovoljivosti. Ipak, svi ti algoritmi su znatno efikasniji od metode istinitosnih tablica.

DPLL procedura može se razmatrati kao algoritam pretrage potpunog stabla valuacija promenljivih koje učestvuju u formuli. Koraci algoritma omogućavaju da se ne pretražuje nužno čitavo stablo. Heuristike koje određuju način na koji se primenjuje pravilo *split* usmeravaju pretragu i mogu bitno da utiču na efikasnost pretrage. Na primer, izbor iskaznog slova u pravilu *split* veoma je važan. Neke varijante ovog pravila su da se bira iskazno slovo sa najviše pojavljivanja u tekućoj formuli, da se bira neko od iskaznih slova iz najkraće klauze itd. Pošto se ispituje da li postoji valuacija u kojoj su sve klauze formule tačne, pohlepni algoritam bi mogao da za *split* promenljivu bira, na primer, onu koja čini najveći broj klauza tačnim u tekućoj parcijalnoj valuaciji. Ovakva heuristika, kao ni druge slične, ne garantuje optimalnost procesa pretrage.

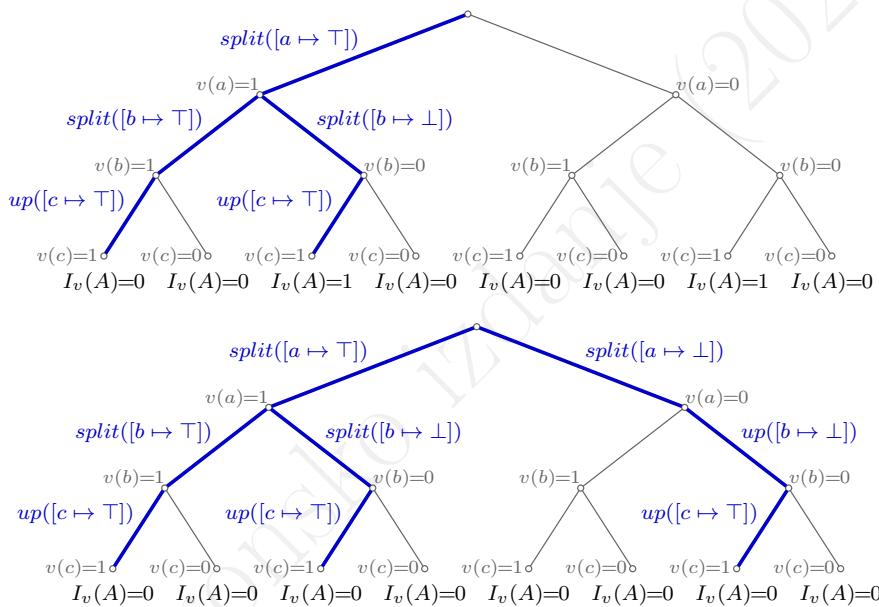
**Primer 8.23.** Neka je potrebno ispitati zadovoljivost formule date klužama:

- $C_1 : \neg a, \neg b, c$
- $C_2 : a, \neg b$
- $C_3 : b, c$
- $C_4 : \neg b, \neg c$

Formula ima dve zadovoljavajuće valuacije. Proverom zadovoljivosti procedurom DPLL, pronalazi se jedna od te dve valuacije. Prvo stablo na slici 8.4 prikazuje proces pretrage u slučaju datog skupa kluža. Kako obe zadovoljavajuće valuacije pridružuju promenljivoj  $b$  vrednost 0, a promenljivoj  $c$  vrednost 1, nakon dodavanja kluze

- $C_5 : b, \neg c$

prethodni skup kluža postaje nezadovoljiv. Proces pretrage procedurom DPLL u ovom slučaju, prikazan je na drugom stablu na istoj slici. U ovom primeru upečatljivo je da DPLL procedura ispituje svega tri od osam listova zahvaljujući tome što osim koraka pretrage oštećenih u pravilu split, postoje i koraci zaključivanja koje se vrši primenom pravila unit propagation, pri čemu se ne ispituju alternative tim pravilom učinjenih zameni.



Slika 8.4: Proces provere zadovoljivosti procedurom DPLL prikazan u vidu pretrage u potpunom stablu valuacija za dva skupa kluža. Pretraga se vrši obilaskom stabla u dubinu sleva nadesno. U prvom slučaju, formula  $A$  data je skupom kluža  $\{\neg a \vee \neg b \vee c, a \vee \neg b, b \vee c, \neg b \vee \neg c\}$  koji je zadovoljiv. U drugom slučaju, skupu kluža dodata je i kluza  $b \vee \neg c$  i dobijeni skup je nezadovoljiv.

DPLL procedura proverava da li je formula zadovoljiva, ali ona se, kao što je već rečeno, može koristiti i za ispitivanje da li je neka formula valjana, poreciva ili kontradikcija. Na primer, formula  $A$  je valjana ako i samo ako je formula  $\neg A$  nezadovoljiva, što se može proveriti DPLL procedurom (pri čemu je, naravno, formulu  $\neg A$  potrebno najpre transformisati u konjunktivnu normalnu formu).

## 8.4 Rešavanje problema svedenjem na SAT

Mnogi praktični problemi mogu se rešiti korišćenjem iskazne logike. Obično je postupak rešavanja ovakav:

- elementarni iskazi (tvrđanje) koji figurišu u opisu problema, predstavljaju se iskaznim promenljivim (u duhu nekog kodiranja);
- uslovi problema se predstavljaju iskaznim formulama nad tim iskaznim promenljivim;
- konjunkcija tih iskaznih formula transformiše se u konjunktivnu normalnu formu;
- zadovoljivost formule u konjunktivnoj normalnoj formi ispituje se SAT rešavačem;

- ukoliko je formula zadovoljiva, svaki njen model daje jedno rešenje polaznog problema.

Svođenjem na SAT mogu se pogodno opisati mnogi problemi nad konačnim domenima. Naredni primer pokazuje kako sabiranje prirodnih brojeva, a i rešavanje jednačina koje uključuju takvo sabiranje mogu biti svedeni na SAT.

**Primer 8.24.** Neka su  $u$  i  $v$  prirodni brojevi manji od 4. Onda, ako je broj  $u$  predstavljen parom iskaznih promenljivih  $(p, q)$  (koje odgovaraju njegovim ciframa u binarnom zapisu) a broj  $v$  predstavljen parom  $(r, s)$ , onda je broj  $u + v$  (po modulu  $2^2$ ) predstavljen parom  $((p \vee r) \underline{\vee} (q \wedge s), q \underline{\vee} s)$  (gde  $\underline{\vee}$  označava ekskluzivnu disjunkciju).

**Primer 8.25.** Neka je zadat problem određivanja vrednosti  $u$ , ako je poznato da je  $v = 2$  i  $v = u + 1$  (po modulu 4). Broj 1 može se predstaviti parom  $(\perp, \top)$  i, kako je poznato da važi  $v = 2$  i  $v = u + 1$ , onda se  $v$  može predstaviti i kao  $(\top, \perp)$  i kao  $((p \vee \perp) \underline{\vee} (q \wedge \top), q \underline{\vee} \top)$ , i, nakon pojednostavljanja, sa  $(p \vee q, \neg q)$ . Da bi se dobila formula koja odgovara zadatim uslovima i iz koje se može dobiti vrednost broja  $u$ , formule na obe pozicije moraju da budu ekvivalentne i sledeća formula mora biti zadovoljiva:  $((p \vee q) \Leftrightarrow \top) \wedge (\neg q \Leftrightarrow \perp)$ . Ona je zadovoljiva i ima samo jedan model. U tom modelu promenljiva  $p$  ima vrednost 0 a promenljiva  $q$  ima vrednost 1. Dakle, nepoznata vrednost  $u$  ima binarni zapis 01, pa je ona jednaka 1.

Navedeni primer je trivijalan, ali ilustruje kako se rešavanje jednačina nad prirodnim brojevima može svesti na SAT. Analogno se mogu rešavati i mnogo kompleksniji problemi.

Rešavanje problema svođenjem na SAT biće ilustrovano kroz nekoliko različitih vrsta svođenja na SAT i nekoliko konkretnih primera.

#### 8.4.1 Primeri kodiranja

**Retka kodiranja.** U praktičnim problemima koji se rešavaju svođenjem na SAT, ne figurišu samo iskazne promenljive, već često i celobrojne promenljive  $v_i$  koje mogu imati vrednosti iz nekog ograničenog skupa. U takvim situacijama često se koristi *retko kodiranje* (eng. *sparse encoding*) u kojem se uvode iskazne promenljive  $p_{v,i}$  koje su tačne ako i samo ako promenljiva  $v$  ima vrednost  $i$ . Time se uslov da promenljiva  $v$  ima jednu vrednost iz zadatog domena  $I$  zadata uslovom („uslov barem-jedna“):

$$\bigvee_{i \in I} p_{v,i}$$

Promenljiva  $v$  ne može imati dve vrednosti istovremeno, što se opisuje formulom („uslov najviše-jedna“):

$$\bigwedge_{i,j \in I, i \neq j} \neg p_{v,i} \vee \neg p_{v,j}$$

Pored uslova koji su potrebni kako bi se iskazalo da promenljiva ima (tačno jednu) vrednost iz nekog konačnog skupa, potrebno je kodirati i razna druga ograničenja. Za neka ograničenja koja se često koriste, postoje ustaljeni načini kodiranja koji daju specifične varijante retkog kodiranja.

U *direktnom kodiranju* (eng. *direct encoding*), za svaku kombinaciju vrednosti  $f(v)$  promenljivih  $v$  iz nekog skupa  $S$  koja nije dozvoljena uvodi se tzv. „klauza konflikta“:

$$\neg \bigwedge_{v \in A} p_{v,f(v)}$$

to jest

$$\bigvee_{v \in A} \neg p_{v,f(v)}$$

U *potpornom kodiranju* (eng. *support encoding*) ograničenja oblika „ako  $v$  ima vrednost  $i$ , onda  $w$  mora imati neku od vrednosti iz skupa  $A$ “ opisuju se formulama (tzv. „potporama“) sledećeg oblika:

$$\neg p_{v,i} \vee \bigvee_{j \in A} p_{w,j}.$$

**Log kodiranje.** U *log kodiranju* (eng. *log encoding*) svakom bitu vrednosti numeričkih promenljivih (zapisanih u binarnoj reprezentaciji) pridružuje se jedna iskazna promenljiva. U ovoj reprezentaciji ne postoji potreba za uslovima „barem-jedna“ i „najviše-jedna“, jer svaka kombinacija vrednosti uvedenih iskaznih promenljivih daje tačno jednu vrednost odgovarajuće promenljive. Naravno, kada je broj mogućih vrednosti numeričke promenljive manji od broja mogućih vrednosti iskaznih promenljivih koji se koriste za njeno kodiranje, neke kombinacije vrednosti iskaznih promenljivih potrebno je zabraniti dodatnim klauzama (na primer, ako promenljiva  $n$  može da ima vrednosti od 0 do 6, za njeno kodiranje se koriste tri iskazne promenljive, ali se zabranjuje njihova kombinacija koja daje vrednost 7).

I u log kodiranju moguće je izraziti uslove koje u slučaju retkih kodiranja izražavaju direktno i potporno kodiranje, ali zbog prirode log kodiranja, te uslove potrebno je zadati nad binarnim kombinacijama koje predstavljaju vrednosti numeričkih promenljivih. Na primer, neka promenljive  $v$  i  $w$  uzimaju celobrojne vrednosti od 0 do 7 i neka su kodirane iskaznim promenljivim  $p_{v,1}, p_{v,2}, p_{v,3}, p_{w,1}, p_{w,2}$  i  $p_{w,3}$ , pri čemu viši indeksi označavaju bitove veće težine. Ukoliko se vrednost 3 promenljive  $v$  uzajamno isključuje sa vrednošću 6 promenljive  $w$ , taj uslov može se kodirati u terminima bitova, klauzom

$$\neg p_{v,1} \vee \neg p_{v,2} \vee p_{v,3} \vee p_{w,1} \vee \neg p_{w,2} \vee \neg p_{w,3}.$$

**Primer 8.26.** Zadatak je obojiti dve kuće (neka su označene sa  $v$  i  $w$ ) po jednom od tri raspoložive boje (neka su označene brojevima 1, 2, 3), ali tako da budu obojene različito.

U retkim kodiranjima problema biće potrebni „barem jedna“ uslovi:

$$p_{v,1} \vee p_{v,2} \vee p_{v,3}$$

$$p_{w,1} \vee p_{w,2} \vee p_{w,3}$$

i „najviše jedna“ uslovi:

$$\neg p_{v,1} \vee \neg p_{v,2}$$

$$\neg p_{v,1} \vee \neg p_{v,3}$$

$$\neg p_{v,2} \vee \neg p_{v,3}$$

$$\neg p_{w,1} \vee \neg p_{w,2}$$

$$\neg p_{w,1} \vee \neg p_{w,3}$$

$$\neg p_{w,2} \vee \neg p_{w,3}$$

Dodatno, u direktnom kodiranju biće opisan i uslov da nisu obe kuće obojene istom bojom (klauze konflikta):

$$\neg p_{v,1} \vee \neg p_{w,1}$$

$$\neg p_{v,2} \vee \neg p_{w,2}$$

$$\neg p_{v,3} \vee \neg p_{w,3}$$

Alternativno, navedene klauze konflikta bi mogle, korišćenjem potpornog kodiranja, da se zadaju na sledeći način:

$$\neg p_{v,1} \vee (p_{w,2} \vee p_{w,3})$$

$$\neg p_{v,2} \vee (p_{w,1} \vee p_{w,3})$$

$$\neg p_{v,3} \vee (p_{w,1} \vee p_{w,2})$$

$$\neg p_{w,1} \vee (p_{v,2} \vee p_{v,3})$$

$$\neg p_{w,2} \vee (p_{v,1} \vee p_{v,3})$$

$$\neg p_{w,3} \vee (p_{v,1} \vee p_{v,2})$$

Pažljivom analizom može se pokazati da poslednje tri navedene klauze nisu potrebne.

Problem može biti opisan i korišćenjem log kodiranja: neka iskazne promenljive  $p_{v,0}, p_{v,1}$  odgovaraju ciframa binarnog zapisa boje koja odgovara kući  $v$ , a  $p_{w,0}, p_{w,1}$  odgovaraju ciframa binarnog zapisa boje koja odgovara kući  $w$ . Ako su bojama pridružene vrednosti 0, 1 i 2, treba zabraniti vrednost 3, pa zato postoje uslovi:

$$\neg p_{v,0} \vee \neg p_{v,1}$$

$$\neg p_{w,0} \vee \neg p_{w,1}$$

Klauze konflikta su sledeće klauze:

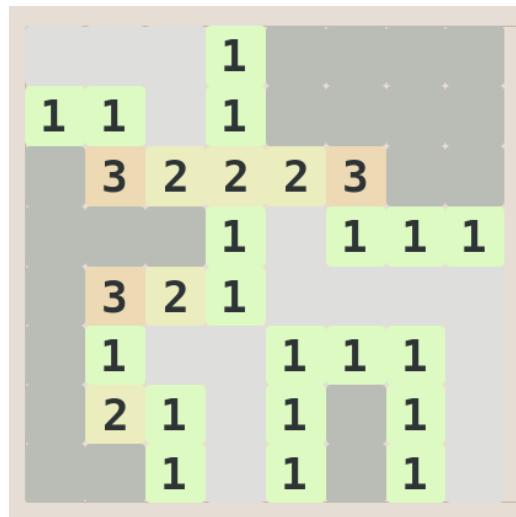
$$p_{v,0} \vee p_{v,1} \vee p_{w,0} \vee p_{w,1} \text{ (nisu obe boje 0)}$$

$$\neg p_{v,0} \vee p_{v,1} \vee \neg p_{w,0} \vee p_{w,1} \text{ (nisu obe boje 1)}$$

$$p_{v,0} \vee \neg p_{v,1} \vee p_{w,0} \vee \neg p_{w,1} \text{ (nisu obe boje 2)}$$

#### 8.4.2 Igra „čistač mina“

Razmotrimo kako iskazna logika može pomoći u igranju računarske igre „čistač mina“ (eng. *minesweeper*, slika 8.5).



Slika 8.5: Igra „čistač mina“.

Data je tabla dimenzija  $n \times m$ . Kada igrač izabere jedno zatvoreno polje na tabli, igra je završena ako se pokaže da je na njoj mina. Inače, igrač saznaće ukupan broj mina na susednim poljima. Ukoliko je taj broj jednak 0, onda se on ne ispisuje a otvaraju se sva susedna polja (jer je jasno da na njima nema mina). Cilj igre je otvoriti sva polja na kojima nema mina. Pokazaćemo da korišćenjem iskazne logike možemo da dobijemo potrebno rasudivanje. Pridružimo iskazne promenljive  $p, q, r, s, t, u$  poljima u levom gornjem uglu table (u nastavku ćemo imenima ovih promenljivih nazivati i polja kojima su pridružena):

$p$	$q$	$r$
$s$	$t$	$u$

Prepostavimo da smo otvorili polje  $p$  i da na njemu nema mine. Prepostavimo i da smo za polje  $p$  dobili broj 1. Opišimo ovu situaciju sredstvima iskazne logike. Smatraćemo da stanje tabele određuje valuaciju uvedenih promenljivih: na nekom polju postoji mina ako i samo ako odgovarajuća promenljiva ima vrednost 1 u toj valuaciji. Kako znamo da na polju  $p$  nema mina, u valuaciji  $v$  koja odgovara stanju tabele, mora da je tačna formula  $\neg p$ . Kako je na polju  $p$  vrednost 1, to znači da je neka mina ili na polju  $q$  ili na polju  $s$  ili na polju  $t$ . Ove uslove opisuju formule:

$$\begin{aligned} q \vee s \vee t, \\ q \Rightarrow \neg s \wedge \neg t, \\ s \Rightarrow \neg q \wedge \neg t, \\ t \Rightarrow \neg q \wedge \neg s \end{aligned}$$

Iz raspoloživog znanja nije moguće odrediti na kojem od tri polja  $q, s, t$  je mina. Moramo da rizikujemo, prepostavimo da smo otvorili polje  $q$ , da na njemu nema mine i da smo dobili opet broj 1.

1	1	$r$
$s$	$t$	$u$

Pokažimo da sada bezbedno možemo da otvorimo polja  $r$  i  $u$ : Kako je na polju  $q$  vrednost 1, to znači da je neka mina ili na polju  $p$  ili na polju  $s$  ili na polju  $t$  ili na polju  $r$  ili na polju  $u$ . Ove uslove opisuju formule:

$$\begin{aligned} p \vee s \vee t \vee r \vee u, \\ p \Rightarrow \neg s \wedge \neg t \wedge \neg r \wedge \neg u, \\ s \Rightarrow \neg p \wedge \neg t \wedge \neg r \wedge \neg u, \\ t \Rightarrow \neg p \wedge \neg s \wedge \neg r \wedge \neg u, \\ r \Rightarrow \neg p \wedge \neg s \wedge \neg t \wedge \neg u, \\ u \Rightarrow \neg p \wedge \neg s \wedge \neg t \wedge \neg r. \end{aligned}$$

Kako nema mina na poljima  $p$  i  $q$ , u valuaciji  $v$  mora da su tačne formule  $\neg p$  i  $\neg q$ , na osnovu čega može biti pojednostavljen navedeni skup iskaznih formula.

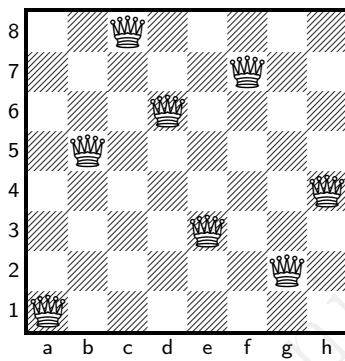
Pitanje da li možemo bezbedno da otvorimo polje  $r$  svodi se na pitanje da li skup navedenih formula ima za logičku posledicu formulu  $\neg r$ . To je isto kao i pitanje da li je implikacija konjunkcije navedenih formula i

formule  $\neg r$  tautologija i isto kao i pitanje da li je konjunkcija navedenih formula i formule  $r$  nezadovoljiva. SAT rešavač može lako da utvrdi da ta formula nije zadovoljiva, te mina ne može biti na polju  $r$ . Analogno se može zaključiti da mina ne može biti na polju  $u$ .

U konkretnom slučaju, zaključak se lako izvodi, ali se opisani mehanizam može uspešno koristiti i u mnogo komplikovanim situacijama.

#### 8.4.3 Problem $n$ dama

Za svako konkretno  $n$ , analogno kao u slučaju  $n = 3$ , problem  $n$  dama (slika ??) može se rešiti korišćenjem iskazne logike. Primenimo kodiranje u stilu retkog kodiranja: neka svakom  $(i, j)$  polju šahovske table odgovara jedna iskazna promenljiva  $p_{ij}$  ( $1 \leq i, j \leq 8$ ). Neka promenljiva  $p_{ij}$  ima vrednost 1 ako je na polju  $(i, j)$  neka dama, a 0 inače.



Slika 8.6: Jedno rešenje za problem 8 dama.

Zadata ograničenja o nenapadanju dama mogu se, korišćenjem direktnog kodiranja, opisati na sledeći način:

1. u svakoj koloni mora da bude barem jedna dama:

$$\bigvee_{i=1, \dots, n} p_{ji}, \text{ za } 1 \leq j \leq n;$$

2. u svakoj koloni mora da bude najviše jedna dama:

$$\bigwedge_{i=1, \dots, n-1; j=i+1, \dots, n} \neg p_{ki} \vee \neg p_{kj}, \text{ za } 1 \leq k \leq n;$$

3. u svakoj vrsti mora da bude najviše jedna dama:

$$\bigwedge_{i=1, \dots, n-1; j=i+1, \dots, n} \neg p_{ik} \vee \neg p_{jk}, \text{ za } 1 \leq k \leq n;$$

4. nema dama koje se napadaju dijagonalno:

$$\bigwedge_{i=1, \dots, n; j=1, \dots, n; k=1, \dots, n; l=1, \dots, n} \neg p_{ij} \vee \neg p_{kl}, \text{ za } |k-i| = |l-j|, k > i.$$

Naglasimo sledeće: kako prva dva skupa uslova obezbeđuju da ima ukupno  $n$  dama, a treći da u svakoj vrsti ima najviše jedna dama, nije potrebno zadavati i uslov da u svakoj vrsti mora da bude barem jedna dama.

Konjunkcija navedenih uslova daje formulu koja opisuje zadati problem. Ona je već u konjunktivnoj normalnoj formi i njena zadovoljivost može biti ispitana nekim SAT rešavačem. Na primer, za  $n = 8$ , formula ima 92 modela i svaki od njih daje po jedno raspoređivanje dama koje ispunjava date uslove.

#### 8.4.4 Raspoređivanje sportskih utakmica

Iskazna logika često se koristi u problemima raspoređivanja. Jedan od takvih problema je raspoređivanje sportskih utakmica. Pretpostavićemo da se koristi kružni sistem takmičenja po principu „igra svako sa svakim“ koji se karakteriše sledećim uslovima:

1. Postoji  $n$  timova ( $n$  je paran broj) i svaka dva tima jednom igraju jedan protiv drugog.
2. Sezona traje  $n - 1$  nedelja.
3. Svaki tim svake nedelje igra jednu utakmicu.

4. Postoji  $n/2$  terena i svake nedelje se na svakom terenu igra jedna utakmica.
5. Nijedan tim ne igra više od dva puta na istom terenu.

Neka su timovi označeni brojevima od 1 do 10. Primer ispravnog rasporeda dat je u tabeli 8.2.

Teren/nedelja	1	2	3	4	5	6	7	8	9
1	6-9	4-6	1-8	4-10	2-8	7-9	5-7	1-2	3-5
2	2-3	1-5	2-4	1-7	9-10	8-10	3-6	4-9	6-8
3	5-10	2-7	3-9	5-9	1-3	1-6	4-8	6-10	4-7
4	1-4	8-9	5-6	3-8	6-7	2-5	1-10	3-7	2-10
5	7-8	3-10	7-10	2-6	4-5	3-4	2-9	5-8	1-9

Tabela 8.2: Primer ispravnog rasporeda za 10 timova.

Osnovni iskaz relevantan za sastavljanje rasporeda je da „tim  $k_1$  igra protiv tima  $k_2$  na terenu  $i$  u nedelji  $j$ “. Kako je u nekim ograničenjima potrebno govoriti o pojedinačnim timovima, a ne samo o parovima, ovo tvrdjenje neće biti predstavljeno jednom promenljivom, nego dvema. Promenljiva  $p_{ij}^{1l}$  označava da tim  $l$  igra (protiv nekog tima) na terenu  $i$  u nedelji  $j$ , kao prvi tim u paru. Promenljiva  $p_{ij}^{2l}$  označava da tim  $l$  igra (protiv nekog tima) na terenu  $i$  u nedelji  $j$ , kao drugi tim u paru. Stoga, skup promenljivih je:

$$\{p_{ij}^{1l} \mid 1 \leq i \leq n/2, 1 \leq j \leq n-1, 1 \leq l \leq n\} \cup \{p_{ij}^{2l} \mid 1 \leq i \leq n/2, 1 \leq j \leq n-1, 1 \leq l \leq n\}$$

Dakle, ukupno se koristi  $2 \cdot (n/2) \cdot (n-1) \cdot n = n^2(n-1)$  promenljivih. Raspored čini skup parova  $(p_{ij}^{1k_1}, p_{ij}^{2k_2})$  koji izražavaju iskaze „tim  $k_1$  igra protiv tima  $k_2$  na terenu  $i$  u nedelji  $j$ “. Ograničenja se izražavaju formulom koja predstavlja konjunkciju sledećih klauza:

1. na svakoj utakmici je redni broj prvog tima manji od rednog broja drugog tima:

$$\neg p_{ij}^{1k_1} \vee \neg p_{ij}^{2k_2}$$

za sve  $1 \leq i \leq n/2, 1 \leq j \leq n-1, 1 \leq k_1, k_2 \leq n$ , takve da važi  $k_1 \geq k_2$ ;

2. na svakom terenu svake nedelje održava se neka utakmica:

$$p_{ij}^{11} \vee \dots \vee p_{ij}^{1n}$$

za sve  $1 \leq i \leq n/2, 1 \leq j \leq n-1$  (dovoljno je obezbediti da svake nedelje na svakom terenu igra barem jedan tim);

3. nijedan tim nijedne nedelje ne igra više od jedne utakmice:

$$\neg p_{i_1j}^{r_1l} \vee \neg p_{i_2j}^{r_2l}$$

za sve  $1 \leq i_1, i_2 \leq n/2, 1 \leq j \leq n-1, 1 \leq r_1, r_2 \leq 2, 1 \leq l \leq n$ , za koje važi  $i_1 \neq i_2$  ili  $r_1 \neq r_2$ ;

4. dva različita tima sastaju se najviše jednom:

$$\neg p_{i_1j_1}^{1k_1} \vee \neg p_{i_1j_1}^{2k_2} \vee \neg p_{i_2j_2}^{1k_1} \vee \neg p_{i_2j_2}^{2k_2}$$

za sve  $1 \leq i_1, i_2 \leq n/2, 1 \leq j_1, j_2 \leq n-1, 1 \leq k_1, k_2 \leq n$ , takve da važi  $j_1 \neq j_2$  i  $k_1 < k_2$ ;

5. nijedan tim ne igra više od dva puta na istom terenu:

$$\neg p_{ij_1}^{r_1k} \vee \neg p_{ij_2}^{r_2k} \vee \neg p_{ij_3}^{r_3k}$$

za sve  $1 \leq i \leq n/2, 1 \leq j_1, j_2, j_3 \leq n-1, 1 \leq k \leq n, 1 \leq r_1, r_2, r_3 \leq 2$ , takve da važi  $j_1 \neq j_2, j_1 \neq j_3$  i  $j_2 \neq j_3$ .

Ukupan broj klauza u formuli je reda  $O(n^6)$  (tog reda je broj klauza koje opisuje 4. korak). Iako je, i za relativno male vrednosti  $n$ , taj broj klauza veoma veliki, savremeni SAT rešavači uspešno mogu ispitati zadovoljivost odgovarajućih formula.

Pažljivom analizom može se pokazati da su neke od navedenih klauza redundantne. Ipak, ukoliko sâm proces rešavanja nije previše zahtevan, racionalno je modelovanje održavati jednostavnim i lakim za razumevanje.

### 8.4.5 Verifikacija softvera

Zamislimo da se u nekoj softverskoj kompaniji radi na reviziji postojećeg koda i, kako je on veoma važan, potrebno ga je unaprediti – učiniti ga efikasnijim i čitljivijim. Nekome se čini da je narednu funkciju `f` napisanu na programskom jeziku C:

```
int f(int y)
{
    int x;
    x = x ^ y;
    y = x ^ y;
    x = x ^ y;
    return x * x;
}
```

moguće zameniti narednom funkcijom `g`:

```
int g(int n)
{
    return n * n;
}
```

Pitanje je, dakle, da li ove dve funkcije za iste argumente uvek vraćaju iste rezultate. Pogodno je u razmatranje uvesti i promenljive za međurezultate (jer programska promenljiva u toku izvršavanja programa može da menja vrednost, a iskazna promenljiva u okviru jedne formule uvek ima istu vrednost):

```
x1 = x ^ y;
y1 = x1 ^ y;
x2 = x1 ^ y1;
```

Kako funkcija `g` vraća vrednost `n*n`, gde je `n` nepromenjena vrednost parametra, dovoljno je dokazati da za funkciju `f` važi da pre naredbe `return x * x;` promenljiva `x` ima istu vrednost kao parametar `y`, tj. da je vrednost `x2` jednaka vrednosti `y`, tj. da važi `x2 == y`. Ako se iskoristi način na koji su zadate vrednosti međurezultata, ova jednakost postaje:

$$(x \wedge y) \wedge ((x \wedge y) \wedge y) == y$$

Ako prepostavimo da tip `int` ima širinu četiri bajta, programske promenljive možemo da predstavimo nizovima od po  $4 \cdot 8 = 32$  iskazne promenljive. Onda izrazu  $(x \wedge y) \wedge ((x \wedge y) \wedge y)$  odgovara niz od 32 iskazne formule i za dokazivanje navedene jednakosti potrebno je dokazati da je svaka od 32 formule koja odgovara izrazu  $(x \wedge y) \wedge ((x \wedge y) \wedge y)$  logički ekvivalentna odgovarajućoj iskaznoj promenljivoj iz reprezentacije programske promenljive `y`. U ovom konkretnom slučaju (a nije tako uvek), može se primetiti da bitovski operator `\wedge` (operator ekskluzivne disjunkcije) применjen na dve vrednosti dejstvuje samo bit-po-bit, tj. rezultat operacije применjen na neka dva bita operanada ne utiče na druge bitove rezultata. Zato se razmatranje navedene jednakosti može svesti na razmatranje pojedinačnih bitova, te je dovoljno dokazati logičku ekvivalentenciju ( $\vee$  označava veznik ekskluzivne disjunkcije):

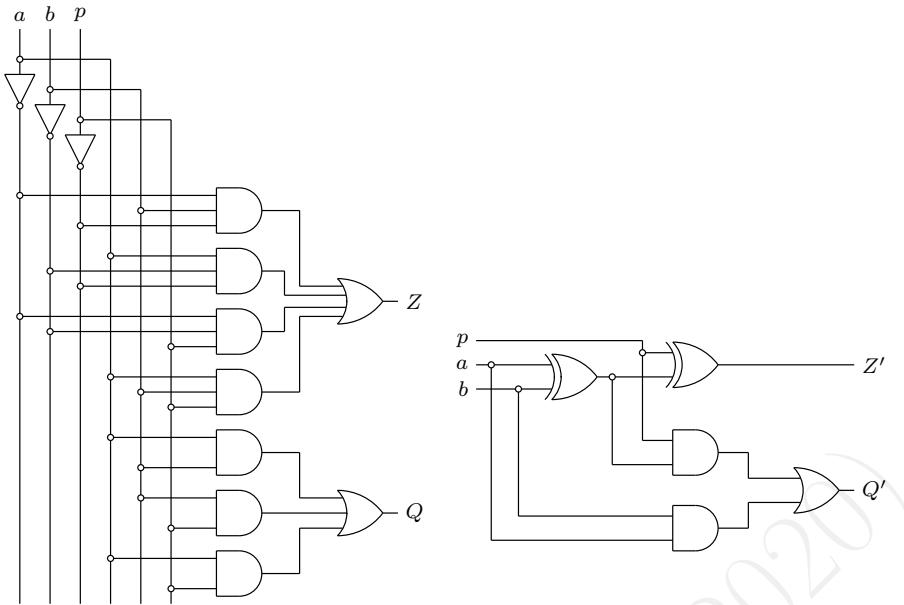
$$(p \vee q) \vee ((p \vee q) \vee q) \equiv q$$

Kako je formula  $(p \vee q) \vee ((p \vee q) \vee q) \Leftrightarrow q$  zaista tautologija, navedena jednakost uvek je tačna, pa su funkcije `f` i `g` ekvivalentne i jedna može biti zamenjena drugom. (Primetimo da programska promenljiva `x` nije inicijalizovana u okviru funkcije `f`, ali to ipak ne utiče na rezultat rada funkcije.)

### 8.4.6 Provera ekvivalentnosti kombinatornih kola

Iskazna logika ima primene u automatizaciji dizajna elektronskih kola, koje uključuju simulaciju, minimizaciju i verifikaciju dizajna kola. Vrsta elektronskih kola koja je najpogodnija za primenu metoda iskazne logike su kombinatorna kola — mreže povezanih logičkih elemenata kod kojih vrednosti izlaza zavise isključivo od vrednosti ulaza. Logički elementi predstavljaju elektronske implementacije logičkih veznika. Takođe, navedeno svojstvo važi i za istinitosne vrednosti iskaznih formula — za datu formulu, one zavise isključivo od vrednosti koje valuacija dodeljuje iskaznim promenljivim u toj formuli.

Jedan problem verifikacije hardvera koji je u vezi sa prethodno uočenom analogijom je provera ekvivalentnosti kombinatornih kola. Dva logička kola su ekvivalentna ukoliko za sve kombinacije vrednosti na svojim ulazima, daju iste izlaze.



Slika 8.7: Osnovni i optimizovani dizajn sabirača.

Ova vrsta verifikacije je korisna u sledećem kontekstu. Kombinatorna kola mogu biti vrlo složena i dizajniraju se u alatima koji podržavaju neki od jezika za opis hardvera kao što su Verilog ili VHDL. Pre nego što se na osnovu kreiranog dizajna pristupi fizičkoj implementaciji logičkog kola, taj dizajn prolazi kroz niz transformacija kojima se vrše optimizacije kola kako bi se uštedelo na njegovoj površini, brzini i slično. Svaki od koraka ovog postupka može biti vrlo složen i iako alogoritmi na kojima pomenute transformacije počivaju garantuju održanje korektnosti, usled složenosti softvera u kojem su ti algoritmi implementirani, uvek postoji mogućnost da je u nekom koraku napravljen greška i da finalni, optimizovani, dizajn kola više nije ekvivalentan polaznom. Zbog toga je pre fizičke izrade logičkog kola potrebno proveriti ekvivalentnost polaznog i finalnog dizajna kola. Treba primetiti da ustanovljena ekvivalentnost ne garantuje funkcionalnu korektnost kola — to da ono zaista radi ono što bi trebalo. Međutim, i to je moguće ustanoviti proverom ekvivalentnosti sa kolom za koje je poznato da je funkcionalno korektno, ukoliko takvo kolo postoji.

Provera ekvivalentnosti kombinatornih kola vrši se tako što se za svaki izlaz kreiraju iskazne formule koje odgovaraju njegovom dizajnu u dva kola. Neka su to formule  $F$  i  $F'$ . Ukoliko su kola ekvivalentna, za sve kombinacije vrednosti ulaza vrednosti ovih formula su jednakе. U terminima iskaznih formula, za svaku valuaciju  $v$  mora da važi  $I_v(F) = I_v(F')$ . Dakle, formula  $F \Leftrightarrow F'$  mora biti tautologija, a formula  $\neg(F \Leftrightarrow F')$  nezadovoljiva. Zadovoljivost iskazne formule može se proveriti pomoću SAT-rešavača. Ukoliko za svaki izlaz rešavač ustanovi da je ovakva formula nezadovoljiva, kola su ekvivalentna, a ukoliko ustanovi da postoji zadovoljavajuća valuacija, ta valuacija predstavlja vrednosti ulaza za koje se izlazi kola razlikuju, što može poslužiti kao polazna tačka u otklanjanju greške.

Postupak provere ekvivalentnosti ilustrovaćemo na primeru sabirača (slika 8.7, levo): ulazne vrednosti  $a$  i  $b$  su cifre brojeva koji se sabiraju,  $p$  je prethodni prenos, a na izlazu je formula koja opisuje cifru zbiru  $Z$  i formula koja opisuje novi prenos  $Q$ . Pretpostavimo da je optimizovanjem kola na slici dobijeno drugo kolo (slika 8.7, desno): ulazne vrednosti  $a$  i  $b$  su cifre brojeva koji se sabiraju,  $p$  je prethodni prenos, a na izlazu su formule koje opisuju cifru zbiru  $Z'$  i novi prenos  $Q'$ . Na osnovu dizajna, za svaki izlaz može se kreirati iskazna formula koja mu odgovara:

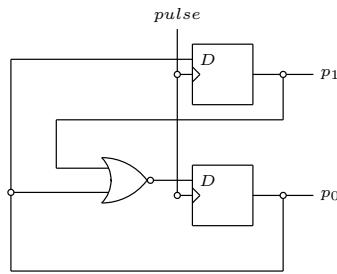
$$Z = (\neg a \wedge b \wedge \neg p) \vee (a \wedge \neg b \wedge \neg p) \vee (\neg a \wedge \neg b \wedge p) \vee (a \wedge b \wedge p)$$

$$Q = (a \wedge b) \vee (b \wedge p) \vee (a \wedge p)$$

$$Z' = (a \vee b) \vee p$$

$$Q' = (a \wedge b) \vee (\neg a \wedge b) \wedge p$$

Kola su ekvivalentna ukoliko su formule  $\neg(Z \Leftrightarrow Z')$  i  $\neg(Q \Leftrightarrow Q')$  nezadovoljive. Treba imati u vidu da formule koje se dobijaju iz ovakvih primena mogu imati i desetine hiljada pa i stotine hiljada promenljivih, ali da SAT-rešavači ipak uspevaju da provere njihovu zadovoljivost (pre svega zahvaljujući nekim pravilnostima u strukturi tih formula, a koje se u toku procesa rešavanja mogu naučiti i iskoristiti).



Slika 8.8: Dizajn brojača koji broji ciklično od 0 do 2.

$p_1^i$	$p_0^i$	$p_1^{i+1}$	$p_0^{i+1}$
0	0	0	1
0	1	1	0
1	0	0	0
1	1	1	0

Tabela 8.3: Tablica prelaska brojača koji broji od 0 do 2.

#### 8.4.7 Ograničena provera modela

Jedna od najkorišćenijih tehniku u verifikaciji hardvera i softvera je *ograničena provera modela* (eng. *bounded model checking*). Funkcionisanje hardvera ili softvera se može apstraktno opisati konačnim automatima čija stanja opisuju stanja sistema koji se izučava, a grane moguće prelaska sistema iz stanja u stanje. Takav konačni automat predstavlja model sistema koji se analizira. Jedan od ciljeva verifikacije je dokazivanje da sistem zadovoljava određena svojstva, poput svojstva da nikad neće doći u stanje koje predstavlja grešku ili bezbednosni rizik. Za sistem koji pruža odgovore na zadate zahteve, primer takvog svojstva je da, nakon stanja u kojem je primljen zahtev, sistem sigurno dolazi u neko stanje u kojem će traženi odgovor biti dat. Ispitivanje ovakvih svojstava, koje se naziva *proverom modela*, može predstavljati neodlučiv problem. Stoga se u praksi obično koristi *ograničena provera modela* koja se svodi na dokazivanje da neko svojstvo važi u svim stanjima u koja se iz polaznog stanja može dospeti u najviše  $k$  prelaza.

Da bi se sprovedla ograničena provera modela tehnikom svodenja na SAT, potrebno je uočiti kako se stanja mogu modelovati iskaznim promenljivim, a potom iskaznim formulama nad tim promenljivim zapisati željeno svojstvo, polazne pretpostavke i način na koji se promenljive menjaju prilikom prelaska iz stanja u stanje.

Ograničenu proveru modela ćemo ilustrovati na primeru dvobitnog brojača prikazanog na slici 8.8. Ulaz *pulse* daje signal brojaču (tj. njegovim flip-flopovima) kad treba da izračuna nove vrednosti izlaza  $p_0$  i  $p_1$ . Te izlazne vrednosti biće kasnije upotrebljene kao nove ulazne vrednosti flip-flopova. Neka su na izlazima  $p_0$  i  $p_1$ , na primer, vrednosti 0 i 0. Kada stigne signal *pulse*, flip-flopovi se aktiviraju,  $p_1$  dobija vrednost 0, a  $p_0$  dobija vrednost izlaza NOR kola za ulaze 0 i 0, tj. vrednost 1. Dakle, ako su ulazne vrednosti  $(p_1, p_0)$  jednake  $(0, 0)$ , izlazne su  $(0, 1)$ . Slično, ako su ulazne vrednosti  $(0, 1)$ , izlazne su  $(1, 0)$ . Generalno, ako je stanje u trenutku  $i$  opisano bitovima  $p_0^i$  i  $p_1^i$ , iz dizajna brojača sledi da za svaka dva uzastopna stanja važi:  $p_0^{i+1} \equiv \neg p_0^i \wedge \neg p_1^i$  i  $p_1^{i+1} \equiv p_0^i$ . Ponašanje brojača sažeto je dato tabelom prelaska prikazanoj na slici 8.3. Kao što se vidi iz tabele, ako se kreće od stanja  $(0, 0)$ , ide se u stanje  $(0, 1)$ , pa u stanje  $(1, 0)$ , pa u stanje  $(0, 0)$ , i to se ciklično ponavlja. Ako se stanja razmatraju kao brojevi u binarnom zapisu, od stanja 0 ide se u stanje 1, pa u stanje 2, pa u stanje 0 i to se ciklično ponavlja, te se zato opisano kolo naziva *brojač*.

Cilj je dokazati da ako brojač započne brojanje od bilo kog stanja (tj. broja) koje nije 3, onda nikada neće doći do broja 3. Željeno svojstvo opisuje se formulom  $\neg p_0^i \vee \neg p_1^i$ . Uslov koji važi u polaznom stanju je  $\neg p_0^0 \vee \neg p_1^0$ . Da bi se pokazalo da stanje  $k$  ne odgovara broju 3, treba ustanoviti da je sledeća formula nezadovoljiva:

$$(\neg p_0^0 \vee \neg p_1^0) \wedge (p_0^1 \Leftrightarrow \neg p_0^0 \wedge \neg p_1^0) \wedge (p_1^1 \Leftrightarrow p_0^0) \wedge \dots \wedge (p_0^k \Leftrightarrow \neg p_0^{k-1} \wedge \neg p_1^{k-1}) \wedge (p_1^k \Leftrightarrow p_0^{k-1}) \wedge (p_0^k \wedge p_1^k)$$

Prvi konjunkt u gornjoj konjunkciji predstavlja polazni uslov, potom slede po dva konjunkta koji opisuju prelase između susednih stanja, a poslednji konjunkt izražava negaciju željenog svojstva u poslednjem stanju. Ukoliko je formula zadovoljiva za neko  $k$ , to znači da postoji put od polaznog stanja kojim se može doći do stanja koje ne zadovoljava traženo svojstvo. U tom slučaju, dobijena valuacija određivala bi stanja preko kojih se može doći do problematičnog stanja, što može pomoći u pronalaženju greške u analiziranom sistemu. U našem primeru, navedena formula nezadovoljiva je za svako  $k$  (ali to opšte svojstvo ne može se utvrditi SAT rešavačem).

### 8.4.8 Link gramatike

*Link gramatike* (eng. *link grammars*) su sintaksički model prirodnog jezika u kojem se parovi reči povezuju *vezama*. U rečniku konkretnog jezika, svakoj reči jezika pridružena je lista mogućih *konektora* različitih tipova, od kojih neki povezuju ulevo a neki udesno. Konektori koji povezuju udesno označavaju se znakom „+“, a konektori koji povezuju ulevo označavaju se znakom „–“. Dve reči u rečenici mogu biti povezane vezom ako postoji konektor koji levu reč povezuje udesno i konektor istog tipa koji desnu reč povezuje ulevo. Na primer, reč „big“ može da ima konektor A+, a reč „cat“ konektor A–, što omogućava kreiranje fraze „big cat“. Povezivanje se odnosi na konkretna pojavljivanja neke reči, pa ukoliko neka reč ima dva pojavljivanja, ta dva pojavljivanja razmatraju se potpuno nezavisno.

Reči mogu da budu povezane sa više reči i ulevo i udesno. Pojedinačnim rečima pridružena su i dodatna pravila koja konjunkcijama i disjunkcijama povezuju moguće konektore. Ovakvi opisi mogu da sadrže proizvoljno ugnjezdene disjunkcije i konjunkcije. Na primer, reč „cat“ može da bude opisana na sledeći način:

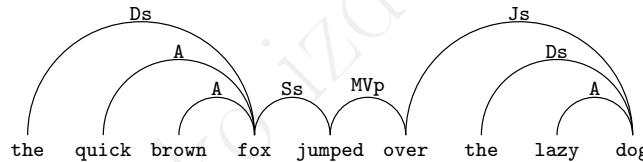
cat (Ds- and (Ss+ or SIs-)) or (A- and Ds- and (Ss+ or SIs-))

Ovaj opis znači da reč „cat“ mora imati konektor Ds ulevo i, istovremeno, konektor Ss udesno ili SIs ulevo ili, alternativno, konektore A i Ds ulevo i, istovremeno, konektor Ss udesno ili SIs ulevo.

Niz reči čini ispravnu rečenicu jezika definisanog gramatikom ako postoji način da se povežu njegove reči tako da važe sledeći uslovi:

- povezanost: vezama su povezane sve reči u rečenici;
- zadovoljenje: veze zadovoljavaju sve postojeće uslove za sve reči u rečenici;
- planarnost: veze se ne sekut (kada su iscrtane iznad reči).

U nastavku je dato ispravno povezivanje za rečenicu na engleskom jeziku „the quick brown fox jumped over the lazy dog“:



Svaki konektor ima specifično značenje (a za konkretan jezik može postojati oko stotinu vrsta konektora). Za engleski jezik, na primer, konektor Ds povezuje član (*the* ili *a*) sa imenicom u jednini, konektor A povezuje atribut sa imenicom, konektor Ss povezuje imenicu-subjekat u jednini sa glagolom, konektor 0 povezuje glagole sa svojim imenicom-objektom. Imena konektora su mnemonička i asociiraju na svoje značenje. Na primer, ime Ds je skovano od engleskih reči „determiner“ (što znači određivač) i „singular“ (što znači jednina).

Parser zasnovan na link gramatikama je algoritam koji zadatoj rečenici pridružuje sintaksičku strukturu sačinjenu od skupa veza između parova reči. Poznavanje sintaksičke strukture rečenice omogućava njen „razumevanje“, dalju automatsku obradu, automatsko prevođenje na drugi jezik i slično.

Prilikom parsiranja rečenice potrebno je ispitati veliki broj mogućih povezivanja reči. Naime, za svaku reč u rečenici potrebno je razmatrati sve moguće konektore zadate opisom te reči u rečniku. Onda je potrebno ispitati da li je tako izabrane konektore moguće povezati na ispravan način. Pošto se svi uslovi mogu svesti na uslove da ima ili nema nekog konektora i ima ili nema neke veze, očigledno je da se ovakvo parsiranje prirodno može svesti na problem SAT. I, zaista, parseri za link gramatike često koriste taj pristup.

Ilustrujmo, pojednostavljenno, na primeru rečenice na engleskom jeziku „the big cat chased John“ parsiranje korišćenjem link gramatika. Prepostavimo da je raspoloživ sledeći sadržaj rečnika:

```
the      Ds+ or Dp+
big      A+
cat      (Ds- and (Ss+ or SIs-)) or (A- and (Ds- and (Ss+ or SIs-)))
chased   ((Ss- or T-) and 0+) or V-
John     0- or (A- and 0-)
```

Modelujmo iskaznim promenljivim i iskaznim formulama finalno ispravno povezivanje reči date rečenice. Neka iskazna promenljiva  $K[\text{the}, \text{ Ds+}]$  označava da će u finalnom povezivanju reč „the“ (prva reč u rečenici, uvek se misli na jednu konkretnu) biti povezana udesno konektorom Ds, neka  $K[\text{the}, \text{ Dp+}]$  označava da će u finalnom povezivanju reč „the“ biti povezana udesno konektorom Dp, neka  $K[\text{cat}, \text{ (Ds- and (Ss+ or SIs-))}]$

označava da će u finalnom povezivanju reč „cat“ biti povezana tako da zadovoljava uslov ( $Ds-$  and ( $Ss+$  or  $SIs-$ )) i tako dalje. Zbog uslova povezanosti i uslova zadovoljenja moraju biti zadovoljeni uslovi:

$$\begin{aligned} K[\text{the}, Ds+] \vee K[\text{the}, Dp+] \\ K[\text{big}, A+] \\ K[\text{cat}, Ds- \text{ and } (Ss+ \text{ or } SIs-)] \vee K[\text{cat}, (A- \text{ and } (Ds- \text{ and } (Ss+ \text{ or } SIs-)))] \\ K[\text{chased}, (Ss- \text{ or } T-) \text{ and } O+] \vee K[\text{chased}, V-] \\ K[\text{John}, O- \text{ or } (A- \text{ and } O-)] \end{aligned}$$

Zbog uslova zadovoljenja, za složene uslove potrebno je dodati i formule koje opisuju logičku strukturu pravila, na primer:

$$K[\text{cat}, Ds- \text{ and } (Ss+ \text{ or } SIs-)] \Leftrightarrow K[\text{cat}, Ds-] \wedge (K[\text{cat}, Ss+] \vee K[\text{cat}, SIs-])$$

Neka iskazna promenljiva  $V[\text{the}-Ds-\text{cat}]$  označava da će u finalnom povezivanju reči „the“ i „cat“ biti povezane konektorom  $Ds$ . Za svaki par reči koji ima isti konektor usmeren u dva smera, zbog uslova ispravnog povezivanja potrebno je dodati formule poput:

$$V[\text{the}-Ds-\text{cat}] \Rightarrow K[\text{the}, Ds+] \wedge K[\text{cat}, Ds-]$$

Ukoliko bi u rečenici postojala još neka reč sa konektorom  $Ds-$  (na primer, „dog“), moralo bi da se obezbedi da se (jedna konkretna) reč „the“ povezuje samo sa jednom:

$$\neg (V[\text{the}-Ds-\text{dog}] \wedge V[\text{the}-Ds-\text{cat}])$$

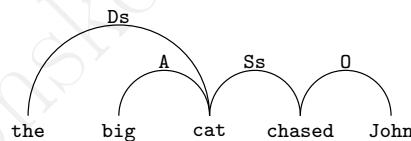
Potrebno je obezbediti i uslov planarnosti: treba zabraniti parove veza koje se sekut. Na primer, ne mogu da budu povezane reči „the“ i „cat“ i istovremeno reči „big“ i „John“:

$$\neg (V[\text{the}-Ds-\text{cat}] \wedge V[\text{big}-A-\text{John}]).$$

Konjunkcija svih ovih uslova daje formula čiju zadovoljivost treba ispitati. Ako je formula zadovoljiva, onda (svaki) njen model daje jedno ispravno povezivanje, tj. parsiranje zadate rečenice. U navedenom primeru, model je valuacija  $v$  u kojoj važi:

$$\begin{aligned} v(V[\text{the}-Ds-\text{cat}]) &= 1 \\ v(V[\text{big}-A-\text{cat}]) &= 1 \\ v(V[\text{cat}-Ss-\text{chased}]) &= 1 \\ v(V[\text{chased}-O-\text{John}]) &= 1 \end{aligned}$$

Što daje ispravno povezivanje ilustrovano sledećom slikom:



Izražajna snaga link gramatika jednaka je izražajnoj snazi kontekstno slobodnih gramatika, ali je opisivanje gramatike prirodnog jezika i parsiranje često znatno lakše za link gramatike.

Rečnici potrebni za parsiranje razlikuju se, naravno, od jezika do jezika. Njihovo kreiranje, posebno za prirodne jezike, zahtevan je zadatak.

#### 8.4.9 SAT-rešavači i dimacs-cnf format

Programe koji rešavaju instance SAT problema zovemo SAT-rešavači (eng. SAT-solvers). Većina savremenih SAT-rešavača zasnovana je na DPLL proceduri, ali je obogaćena mnogim tehnikama i heuristikama. Neki od popularnih SAT-rešavača su MiniSAT, PicoSAT i zChaff.

SAT-rešavači obično očekuju ulaz u DIMACS-CNF formatu. U ovom formatu, prvi red sadrži informaciju o broju iskaznih promenljivih i broju kluaza, a naredni redovi sadrže zapis po jedne kluaze. Promenljive su označene rednim brojevima, negirane promenljive odgovarajućim negativnim brojevima i svaki red završava se brojem 0. Na primer, sadržaj

```
p cnf 3 2
1 -3 0
-1 2 3 0
```

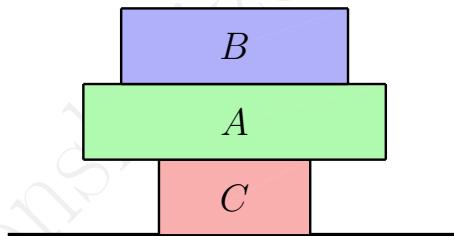
odgovara formuli (sa tri promenljive i dve kluze):  $(p_1 \vee \neg p_3) \wedge (\neg p_1 \vee p_2 \vee p_3)$ .



## Automatsko rasuđivanje u logici prvog reda

Logika prvog reda, predikatska logika (eng. *first order logic, predicate logic*), znatno je izražajnija od iskazne logike. Ključna novina u odnosu na iskaznu logiku je uvođenje kvantifikovanja, univerzalnog i egzistencijalnog. Zahvaljujući kvantifikatorima, u logici prvog reda mogu se formulisati tvrđenja koja nije moguće formulisati na jeziku iskazne logike. U logici prvog reda dozvoljeno je samo kvantifikovanje promenljivih.<sup>1</sup> U okviru logike prvog reda mogu se opisati mnoge matematičke teorije i mnogi praktični problemi. Za neke probleme (nad konačnim domenima) pogodnije je rešavanje korišćenjem iskazne logike, ali za neke je opisivanje i rešavanje znatno lakše korišćenjem predikatske logike.

Kao i u iskaznoj logici, centralni problemi u predikatskoj logici su ispitivanje da li je data formula valjana i da li je data formula zadovoljiva. Za razliku od iskazne logike, ovi problemi nisu odlučivi, te ne postoje efektivni algoritmi za njihovo rešavanje. No, problem ispitivanja nezadovoljivosti i problem ispitivanja valjanosti za predikatsku logiku su poluodlučivi, pa postoje metode koje za svaku valjanu formulu mogu da dokažu da je ona valjana (ali ne mogu za bilo koju formulu koja nije valjana da utvrde da nije valjana).



Slika 9.1: Ilustracija za problem uređenja tri kutije.

**Primer 9.1.** Razmotrimo jednu jednostavnu varijantu problema slaganja kutija: kutije (označene slovima) poređane su jedna na drugu. Za neke se zna da li su ispod ili iznad neke druge kutije, ali nije zadata potpuna informacija o poretku svih kutija. U ovom jednostavnom primeru dovoljno je baratati informacijama o tome koja je kutija iznad neke kutije. Ipak, koriste se i informacije o odnosu ispod, čime se ilustruju pogodnosti upotrebe jezika koji nije minimalni. U mnogim realnim primenama, od minimalnog jezika znatno je pogodniji neki izražajniji jezik.

Prepostavimo da su nekako naslagane tri kutije  $A$ ,  $B$  i  $C$  i da je poznato da je  $B$  iznad  $A$ , a da je  $C$  ispod  $A$ . Pitanje je da li je  $B$  iznad ili ispod  $C$ . Opisani problem možemo opisati u terminima iskazne logike: iskazna promenljiva  $a_{AB}$  (od engleskog above) može da označava da je  $A$  iznad  $B$ ,  $a_{AC}$  da je  $A$  iznad  $C$ ,  $a_{BA}$  da je  $B$  iznad  $A$ ,  $b_{BC}$  (od engleskog below) da je  $B$  ispod  $C$ , itd. Potrebno je za svake dve kutije obezbediti da važi da ako je prva iznad druge, onda druga nije iznad prve, na primer, za kutije  $A$  i  $B$  važi:  $a_{AB} \Rightarrow \neg a_{BA}$ . Potrebno je za svake dve kutije obezbediti da važi da ako je prva iznad druge, onda je druga ispod prve i obratno, na primer, za kutije  $A$  i  $B$  važi:  $a_{AB} \Leftrightarrow b_{BA}$ . Potrebno je za svake tri kutije obezbediti da važi: ako je prva iznad druge i druga iznad treće, onda je prva iznad treće, na primer, za

<sup>1</sup>U logici višeg reda predikati i funkcije kao argumente mogu imati druge predikate i funkcije i dozvoljeno je njihovo kvantifikovanje. Na primer, u logici drugog reda predikati i funkcije mogu za argumente imati predikate i funkcije prvog reda i mogu biti kvantifikovani. Predikati i funkcije reda  $n$  mogu za argumente imati predikate i funkcije reda  $n - 1$  i mogu biti kvantifikovani.

kutije  $A, B$  i  $C$  važi:  $a_{AB} \wedge a_{BC} \Rightarrow a_{AC}$ . Ako postoje tri kutije, onda ovakvih uslova ima  $3! = 6$ , a ako ih ima  $n$ , onda tih uslova ima  $6\binom{n}{3}$ . Dakle, iako jeste moguće, kodiranje u terminima iskazne logike može da bude rogočatno i prostorno veoma zahtevno. Bilo bi dobro ako bismo umesto  $6\binom{n}{3}$  uslova mogli da koristimo samo jedan: „za svake tri kutije  $X, Y, Z$  važi: ako je  $a_{XY}$  i  $a_{YZ}$  onda je  $a_{XZ}$ .“ Logika prvog reda daje takvu mogućnost i zadati problem mogao bi da se elegantno opiše sledećim uslovima, pri čemu se ne koriste iskazne promenljive poput  $a_{AB}$  nego atomičke formule sa argumentima poput  $\text{above}(A, B)$  i  $\text{below}(A, B)$ , pri čemu  $\text{above}(A, B)$  označava da je  $A$  iznad  $B$ , a  $\text{below}(A, B)$  označava da je  $A$  ispod  $B$ :

- „za svake dve kutije  $x, y$  važi: ako je  $\text{above}(x, y)$  onda nije  $\text{above}(y, x)$ “;
- „za svake dve kutije  $x, y$  važi:  $\text{above}(x, y)$  ako i samo ako  $\text{below}(y, x)$ “;
- „za svake tri kutije  $x, y, z$  važi: ako je  $\text{above}(x, y)$  i  $\text{above}(y, z)$  onda je  $\text{above}(x, z)$ “.

Za sve navedene, a i druge slične formule, potrebno je definisati način na koji im se pridružuje vrednost tačno ili netačno.

Za rešavanje početnog problema, potrebno je iz navedenih uslova i iz prepostavki da važi  $\text{above}(B, A) \wedge \text{below}(C, A)$  izvesti zaključak  $\text{above}(B, C)$  ili  $\text{below}(B, C)$ .

## 9.1 Sintaksa i semantika logike prvog reda

Sintaksa logike prvog reda govori o njenom *jeziku* – o skupu njenih (ispravno formiranih) formula i ne razmatra njihovo značenje i njihove (moguće) istinitosne vrednosti. Značenje formula logike prvog reda uvodi semantika logike prvog reda.

### 9.1.1 Sintaksa logike prvog reda

Skup formula logike prvog reda obično se definiše za fiksiran, prebrojiv skup promenljivih  $V$ , dve logičke konstante —  $\top$  (te) i  $\perp$  (nete), konačan skup osnovnih logičkih veznika: unarnog — *negacija* i binarnih — *konjunkcija*, *disjunkcija*, *implikacija*, *ekvivalencija*, kao i dva kvantifikatora (kvantor) — *univerzalni* i *egzistencijalni*. Nabrojane komponente čine *logički* (ili *opšti*) deo jezika prvog reda. Pored njih, u izgradnji formula logike prvog reda koriste se elementi *signature*.

**Definicija 9.1** (Signatura). *Signaturu  $\mathcal{L}$  čine:*

- (najviše prebrojiv) skup funkcionalnih simbola (sa svojim arnostima);
- (najviše prebrojiv) skup predikatskih (relacijskih) simbola (sa svojim arnostima);

Funkcijske simbole arnosti 0 zovemo simbolima konstanti.

Na primer, sa  $f_2$  označavamo da funkcionalni simbol  $f$  ima arnost 2.

Pojedinačni jezici prvog reda razlikuju se po signaturi. Za svaki zaseban problem (ili matematičku teoriju) bira se specifična, pogodna signatura. Na primer, za rešavanje problema kutija (iz primera 9.1), koristiće se predikatski simboli koji odgovaraju položajima ispod i iznad, u izgradnji geometrije koristiće se predikatski simbol za paralelnost, a u izgradnji aritmetike – funkcionalni simbol +.

Naredna definicija uvodi pojmove *termova* i *formula logike prvog reda* (ili kraće – skup *formula*) nad nekom (fiksiranom) signaturom  $\mathcal{L}$ . Termovi, atomičke formule i formule nad signaturom  $\mathcal{L}$  zvaćemo, kraće, samo *termovi* (eng. *terms*), *atomičke formule* (eng. *atomic formulae*) i *formule* (eng. *formulae*), ako je signatura jasno određena kontekstom ili ako nije relevantna. Intuitivno, termovi odgovaraju elementima nekog domena (na primer, zbiru može da odgovara broj), a formulama odgovaraju tvrđenja koja mogu biti tačna ili netačna.

**Definicija 9.2** (Skup termova i formula logike prvog reda nad signaturom  $\mathcal{L}$ ). *Za signaturu  $\mathcal{L}$ :*

- *promenljive i funkcionalni simboli arnosti 0* su termovi; term je i objekat dobijen primenom funkcionalnog simbola  $f$  arnosti  $n$  na termove  $t_1, \dots, t_n$ ; termovi mogu biti dobijeni samo na opisane načine.
- atomička formula je logička konstanta ili objekat dobijen primenom predikatskog simbola  $p$  arnosti  $n$  na termove  $t_1, \dots, t_n$ ;

- atomičke formule su formule;
- ako su  $\mathcal{A}$  i  $\mathcal{B}$  formule, onda su formule i objekti dobijeni kombinovanjem ovih formula logičkim veznicima i kvantifikatorima (sa promenljivim);
- formule mogu biti dobijene samo na načine opisane u prethodne dve stavke.

*Literal* (eng. *literal*) je atomička formula ili negacija atomičke formule. *Klauza* (eng. *clause*) je disjunkcija više literala.

U navedenoj definiciji (u duhu *apstraktne sintakse*) ne govori se o tome kako se zapisuju ili čitaju formule logike prvog reda, već samo o tome kako se grade na apstraktan način (implicitno — u vidu stabla). Način na koji se logički veznici i iskazne formule zapisuju može se zadati konkretnom sintaksom. Analogno iskaznom slučaju, uobičajeno je da se negacija zapisuje kao  $\neg$ , konjunkcija kao  $\wedge$ , disjunkcija kao  $\vee$ , implikacija kao  $\Rightarrow$ , ekvivalencija kao  $\Leftrightarrow$ , univerzalni kvantifikator kao  $\forall$ , egzistencijalni kao  $\exists$ . Primenu funkcijskog simbola  $f$  na termove  $t_1, \dots, t_n$  zapisujemo prefiksno kao  $f(t_1, \dots, t_n)$  (i analogno za predikatske simbole). Binarne funkcijске i predikatske simbole ponekad ćemo zapisivati i u infiksnom obliku. U ovakovom konkretnom zapisu (koji će se koristiti u nastavku) ako su  $\mathcal{A}$  i  $\mathcal{B}$  formule i  $x$  element skupa  $V$ , onda su formule i  $(\neg\mathcal{A})$ ,  $(\mathcal{A} \wedge \mathcal{B})$ ,  $(\mathcal{A} \vee \mathcal{B})$ ,  $(\mathcal{A} \Rightarrow \mathcal{B})$ ,  $(\mathcal{A} \Leftrightarrow \mathcal{B})$ ,  $(\forall x)\mathcal{A}$ ,  $(\exists x)\mathcal{A}$  i slično. Na primer, zapis  $(\forall x)\mathcal{A}$  čitamo „za svako  $x$   $\mathcal{A}$ “, zapis  $(\exists x)\mathcal{A}$  čitamo „postoji  $x$  takvo da je  $\mathcal{A}$ “. U ovakovom, konkretnom zapisu, neophodno je koristiti zagrade kako bi se izbegla višesmislenost. Da bi se izbeglo korišćenje velikog broja zagrada obično se izostavljaju spoljne zagrade i podrazumeva prioritet veznika kao u iskaznoj logici, uz dodatak da kvantifikatori imaju viši prioritet od svih veznika.

Uz indeks ili bez indeksa, simbole konstanti obično (mada ne isključivo) označavamo simbolima  $a, b, c, \dots$ , funkcijске simbole arnosti veće od 0 simbolima  $f, g, h, \dots$ , predikatske simbole simbolima  $p, q, r, \dots$ , promenljive simbolima  $x, y, z, \dots$ , formule simbolima  $\mathcal{A}, \mathcal{B}, \mathcal{C}, \dots$ , skupove formula simbolima  $\Gamma, \Delta, \dots$

Ako su dve formule  $\mathcal{A}$  i  $\mathcal{B}$  identične (tj. ako su, zapisane u konkretnoj sintaksi, jednake kao nizovi simbola), onda to označavamo  $\mathcal{A} = \mathcal{B}$  a inače pišemo  $\mathcal{A} \neq \mathcal{B}$ .

**Primer 9.2.** Signatura za problem iz primera 9.1 je  $\mathcal{L} = (\{\}, \{above_{/2}, below_{/2}\})$ . Za ovu signaturu i skup promenljivih  $V = \{x, y, z, \dots\}$ , neke od formula su  $above(x, z)$ ,  $below(x, y)$ ,  $\forall x \neg above(x, x)$ .

**Primer 9.3.** Za signaturu  $\mathcal{L} = (\{e_{/0}, \star_{/2}\}, \{\prec_{/2}\})$ , i skup promenljivih  $V = \{x, y, z, \dots\}$ , neki od termova su:  $e$ ,  $\star(x, y)$ , a neke od formula su  $\prec(x, e)$ ,  $\prec(\star(x, y), z)$ ,  $\forall x \neg(\prec(x, x))$ . Ukoliko se binarni funkcijski i predikatski simboli zapišu infiksno (umesto prefiksno), onda se navedeni termovi i formule zapisuju na sledeći način:  $e$ ,  $x \star y$ ,  $x \prec e$ ,  $(x \star y) \prec z$ ,  $\forall x \neg(x \prec x)$ .

**Primer 9.4.** Razmotrimo signaturu  $\mathcal{L} = (\{sokrat_{/0}\}, \{man_{/1}, mortal_{/1}\})$ . Neki od termova nad ovom signaturom i skupom promenljivih  $V = \{x, y, z, \dots\}$  su:  $x$ ,  $y$ ,  $sokrat$ , neke od atomičkih formula su  $man(x)$ ,  $mortal(y)$ ,  $mortal(sokrat)$ , a neke od formula su  $\forall x man(x)$  i  $\forall x (man(x) \Rightarrow mortal(x))$ .

**Primer 9.5.** U neformalnom i poluformalnom matematičkom izražavanju, mogu se sresti konstrukcije poput  $\forall x < \delta \dots$ . Iako, po simbolima koje uključuje, podseća na formulu logike prvog reda, ovo očigledno nije dobro zasnovana formula. Na primer, neprekidnost i ravnomerna neprekidnost (totalne) realne funkcije  $f$  često se definišu na sledeći način:

$$(\forall a)(\forall \varepsilon > 0)(\exists \delta > 0)(\forall x)(|x - a| < \delta \Rightarrow |f(x) - f(a)| < \varepsilon)$$

$$(\forall \varepsilon > 0)(\exists \delta > 0)(\forall x_1, x_2)(|x_1 - x_2| < \delta \Rightarrow |f(x_1) - f(x_2)| < \varepsilon)$$

Odgovarajuće (precizno zapisane) dobro zasnovane formule su:

$$(\forall a)(\forall \varepsilon > 0 \Rightarrow (\exists \delta)(\delta > 0 \wedge (\forall x)(|x - a| < \delta \Rightarrow |f(x) - f(a)| < \varepsilon)))$$

$$(\forall \varepsilon)(\varepsilon > 0 \Rightarrow (\exists \delta)(\delta > 0 \wedge (\forall x_1)(\forall x_2)(|x_1 - x_2| < \delta \Rightarrow |f(x_1) - f(x_2)| < \varepsilon)))$$

**Definicija 9.3** (Slobodno i vezano pojavljivanje promenljive). U formulama  $\forall x\mathcal{A}$  i  $\exists x\mathcal{A}$ , formula  $\mathcal{A}$  je doseg kvantifikatora.

Pojavljivanje promenljive  $x$  je vezano u  $\forall x$  i  $\exists x$ , kao i ako je u dosegu kvantifikatora  $\forall x$  ili  $\exists x$ , a inače je slobodno. Promenljiva je vezana (slobodna) u formuli ako i samo ako ima vezano (slobodno) pojavljivanje u toj formuli.

Primetimo da promenljiva može biti i slobodna i vezana u jednoj formuli.

**Primer 9.6.** U formuli  $p(x, y)$ , pojavljivanje promenljive  $x$  je slobodno i ona je slobodna u ovoj formuli.

U formuli  $p(x, y) \Rightarrow (\forall x)q(x)$  prvo pojavljivanje promenljive  $x$  je slobodno, a drugo i treće pojavljivanje je vezano. U ovoj formuli, promenljiva  $x$  je i slobodna i vezana.

U formuli  $(\forall x)p(x, y) \Rightarrow (\forall x)q(x)$ , sva pojavljivanja promenljive  $x$  su vezana i promenljiva je vezana u ovoj formuli.

U sva tri primera, pojavljivanja promenljive  $y$  su slobodna.

Često se zapisom  $\mathcal{A}(x_1, x_2, \dots, x_n)$  naglašava da formula  $\mathcal{A}$  ima slobodne promenljive  $x_1, x_2, \dots, x_n$ .

Priroda vezanih promenljivih u formuli bitno je drugačija od prirode slobodnih promenljivih. Naime, značenje formule  $(\forall x)\mathcal{A}(x)$  (koje će tek u nastavku biti definisano) ne zavisi od  $x$  i isto je kao i značenje formule  $(\forall y)\mathcal{A}(y)$  (slično kao što vrednost  $\int_0^\pi \sin(x)dx$  ne zavisi od  $x$ , iako  $x$  figuriše u ovom izrazu). Dakle, sva pojavljivanja vezanih promenljivih mogu se preimenovati, bez uticaja na značenje formule. Na primer, formula  $p(x, y) \Rightarrow (\forall x)q(x)$  iz primera 9.6 ima isto značenje (koje će biti precizno definisano u nastavku), kao i formula  $p(x, y) \Rightarrow (\forall z)q(z)$ . S druge strane, značenje formula zavisi od slobodnih promenljivih u njima.

Formule bez slobodnih promenljivih zovu se *zatvorene formule* ili *rečenice*. Ako formula  $\mathcal{A}$  ima kao slobodne samo promenljive  $x_1, x_2, \dots, x_k$  onda formulu  $(\forall x_1)(\forall x_2) \dots (\forall x_k)\mathcal{A}$  nazivamo *univerzalnim zatvorenjem* formule  $\mathcal{A}$ , a formulu  $(\exists x_1)(\exists x_2) \dots (\exists x_k)\mathcal{A}$  *egzistencijalnim zatvorenjem* formule  $\mathcal{A}$ .

### 9.1.2 Zamena

U logici prvog reda, razmatramo *zamene promenljive termom* i *zamene formule formulom*. Zamene promenljive termom biće korišćene u kontekstu unifikacije (poglavlje 9.3.3), a zamene formule formulom u kontekstu transformisanja formula zarad jednostavnijeg ispitivanja valjanosti ili zadovoljivosti. U daljem tekstu ćemo pod terminom *izraz* podrazumevati i termove i formule. Izraz koji je rezultat primene zamene  $\sigma$  nad izrazom  $E$ , označavamo sa  $E\sigma$ .

Zamena promenljive termom definiše se u logici prvog reda u istom duhu kao u iskaznoj logici: ako termove i formule reprezentujemo u računarskom programu u vidu stabla, onda zamenu promenljive termom možemo opisati rekurzivnom funkcijom koja sve listove formule koji odgovaraju zadatoj promenljivoj zamenjuje nekim drugim zadatim stablom. U tome postoji jedno dodatno pravilo. Ukoliko se primenjuje zamena promenljive  $x$  termom  $t$ , koju označavamo  $[x \mapsto t]$ , nijedna promenljiva iz  $t$  ne sme da postane vezana (jer bi to menjalo značenje formule). Zato, ako se, na primer, primenjuje zamena  $[x \mapsto t]$  na formulu  $(\forall y)\mathcal{A}$  (ili  $(\exists y)\mathcal{A}$ ) a pri tome term  $t$  sadrži promenljivu  $y$ , onda promenljivu  $y$  najpre treba preimenovati u formuli  $(\forall y)\mathcal{A}$ , dajući  $(\forall z)\mathcal{A}[y \mapsto z]$ . Time se značenje formule  $(\forall y)\mathcal{A}$  (koje će tek u nastavku biti definisano) ne menja i isto je kao i značenje formule  $(\forall z)\mathcal{A}[y \mapsto z]$ . Zahvaljujući tom pravilu, na primer,  $((\exists y)otac(x, y))[x \mapsto y]$  jednak je  $(\exists z)otac(y, z)$  a ne  $(\exists y)otac(y, y)$  (što ima drugačije značenje).

**Definicija 9.4** (Supstitucija i kompozicija supstitucija). Supstitucija (ili uopštena zamena)  $\sigma$  je niz uzaštopnih zameni  $[x_1 \mapsto t_1], [x_2 \mapsto t_2], \dots, [x_n \mapsto t_n]$  gde su  $x_i$  promenljive,  $t_i$  su proizvoljni termovi, važi  $x_i \neq x_j$  za  $i \neq j$  i nijedan od termova  $t_i$  ne sadrži nijednu od promenljivih  $x_j$ . Takvu zamenu zapisujemo kraće  $[x_1 \mapsto t_1, x_2 \mapsto t_2, \dots, x_n \mapsto t_n]$ .

Za supstitucije  $\phi = [x_1 \mapsto t_1, x_2 \mapsto t_2, \dots, x_n \mapsto t_n]$  i  $\lambda = [y_1 \mapsto s_1, y_2 \mapsto s_2, \dots, y_m \mapsto s_m]$ , pri čemu nije dan od termova  $y_i$ ,  $s_i$  ne sadrži nijednu od promenljivih  $x_j$ , kompozicija supstitucija  $\phi\lambda$  je supstitucija  $[x_1 \mapsto t_1\lambda, x_2 \mapsto t_2\lambda, \dots, x_n \mapsto t_n\lambda, y_1 \mapsto s_1, y_2 \mapsto s_2, \dots, y_m \mapsto s_m]$ .

**Primer 9.7.** Za  $\sigma = [x \mapsto f(y)]$  i  $s = g(a, x)$  važi  $s\sigma = g(a, f(y))$ .

Za  $\sigma = [x \mapsto f(x)]$  i  $s = g(a, x)$  važi  $s\sigma = g(a, f(x))$ .

Za  $\sigma = [x \mapsto f(y), y \mapsto a]$ ,  $s = g(a, x)$  i  $t = g(y, g(x, y))$  važi  $s\sigma = g(a, f(y))$  i  $t\sigma = g(a, g(f(y), a))$ .

Za  $\sigma = [x \mapsto \text{sokrat}, y \mapsto \text{platon}]$  i  $s = \text{učitelj}(x, y)$  važi  $s\sigma = \text{učitelj}(\text{sokrat}, \text{platon})$ .  
 Za  $\sigma = [x \mapsto a]$  i  $s = u(x, x)$  važi  $s\sigma = u(a, a)$ .  
 Za  $\phi = [x \mapsto f(y)]$  i  $\lambda = [y \mapsto g(z)]$ , važi  $\phi\lambda = [x \mapsto f(g(z)), y \mapsto g(z)]$ .  
 Za  $\phi = [x \mapsto f(y)]$  i  $\lambda = [y \mapsto g(x)]$ , važi  $\phi\lambda = [x \mapsto f(g(x)), y \mapsto g(x)]$ .

Pored zamene promenljive termom, potreban nam je i pojam zamene formule formulom, uveden narednom definicijom.

**Definicija 9.5** (Zamena formule formulom). *Formulu dobijenu zamenom (supstitucijom) formule  $\mathcal{B}_1$  formulom  $\mathcal{B}_2$  u formuli  $\mathcal{A}$ , označavamo sa  $\mathcal{A}[\mathcal{B}_1 \mapsto \mathcal{B}_2]$  i definišemo na sledeći način:*

- ako je  $\mathcal{A} = \mathcal{B}_1$ , onda je  $\mathcal{A}[\mathcal{B}_1 \mapsto \mathcal{B}_2] = \mathcal{B}_2$ ;
- ako je formula  $\mathcal{A}$  atomička i nije jednaka  $\mathcal{B}_1$ , onda je  $\mathcal{A}[\mathcal{B}_1 \mapsto \mathcal{B}_2] = \mathcal{A}$ ;
- $(\neg \mathcal{A})[\mathcal{B}_1 \mapsto \mathcal{B}_2] = \neg(\mathcal{A}[\mathcal{B}_1 \mapsto \mathcal{B}_2])$ ;
- $(\mathcal{A} \wedge \mathcal{B})[\mathcal{B}_1 \mapsto \mathcal{B}_2] = \mathcal{A}[\mathcal{B}_1 \mapsto \mathcal{B}_2] \wedge \mathcal{B}[\mathcal{B}_1 \mapsto \mathcal{B}_2]$ ;
- $(\mathcal{A} \vee \mathcal{B})[\mathcal{B}_1 \mapsto \mathcal{B}_2] = \mathcal{A}[\mathcal{B}_1 \mapsto \mathcal{B}_2] \vee \mathcal{B}[\mathcal{B}_1 \mapsto \mathcal{B}_2]$ ;
- $(\mathcal{A} \Rightarrow \mathcal{B})[\mathcal{B}_1 \mapsto \mathcal{B}_2] = \mathcal{A}[\mathcal{B}_1 \mapsto \mathcal{B}_2] \Rightarrow \mathcal{B}[\mathcal{B}_1 \mapsto \mathcal{B}_2]$ ;
- $(\mathcal{A} \Leftrightarrow \mathcal{B})[\mathcal{B}_1 \mapsto \mathcal{B}_2] = \mathcal{A}[\mathcal{B}_1 \mapsto \mathcal{B}_2] \Leftrightarrow \mathcal{B}[\mathcal{B}_1 \mapsto \mathcal{B}_2]$ ;
- $(\forall x \mathcal{A})[\mathcal{B}_1 \mapsto \mathcal{B}_2] = (\forall x)\mathcal{A}[\mathcal{B}_1 \mapsto \mathcal{B}_2]$ ;
- $(\exists x \mathcal{A})[\mathcal{B}_1 \mapsto \mathcal{B}_2] = (\exists x)\mathcal{A}[\mathcal{B}_1 \mapsto \mathcal{B}_2]$ .

Ako formule reprezentujemo u računarskom programu u vidu stabla, onda zamenu formule formulom možemo, slično kao u iskaznom slučaju, opisati rekurzivnom funkcijom (čiji su bazni slučajevi opisani prvim dvema stavkama naredne definicije). Ta funkcija zamenjuje jedno podstablo formule (zapravo — sva podstabla koja odgovaraju istoj formuli) nekim drugim zadatim stablom.

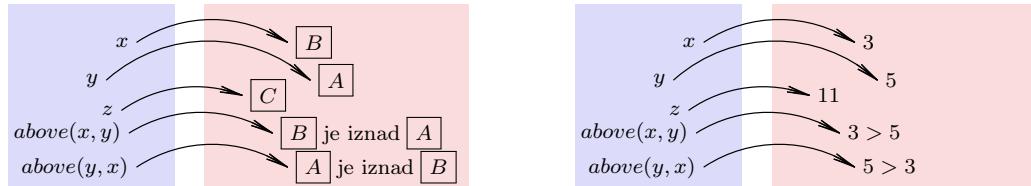
### 9.1.3 Semantika logike prvog reda

Semantika logike prvog reda govori o *značenju* formula. Kao i u jednostavnijem slučaju iskazne logike, osnovna ideja semantike je da istinitosne vrednosti formula definiše u skladu sa uobičajenim, svakodnevnim rasuđivanjem. U odnosu na iskazni slučaj, stvari komplikuju kvantifikatori, kao i potpuno drugačija priroda promenljivih. Naime, promenljive se ne preslikavaju, kao u iskaznoj logici, u skup  $\{0, 1\}$ , nego se mogu preslikavati u elemente proizvoljnog skupa koji zovemo *univerzum* (na primer, u skup celih brojeva ili u skup stanovnika neke države).<sup>2</sup> Značenje formula, dodatno, nije određeno samo načinom na koji su promenljivim pridružene vrednosti, nego i time što odgovara funkcijskim i predikatskim simbolima — to će biti neke konkretne funkcije i relacije nad izabranim domenom. Zbog toga, istinitosna vrednost formule zavisi od više izbora i za različite izbore može da bude drugačija. Reći ćemo da je formula *valjana* ako je tačna za svaki od tih izbora.

Razmotrimo, na primer, formulu  $\text{above}(x, y)$ . Možemo je interpretirati na mnogo različitih načina. Jedan je da smatramo da  $x$  i  $y$  označavaju kutije naslagane na nekoj konkretnoj gomili, a da predikatskom simbolu  $\text{above}$  odgovara relacija „biti iznad“ za kutije na toj konkretnoj gomili. Drugi način je da smatramo da  $x$  i  $y$  označavaju cele brojeve, a da predikatskom simbolu  $\text{above}$  odgovara relacija  $>$  nad celim brojevima. U svakom slučaju, formulama pridružujemo značenje vezano za neki konkretan skup i neke konkretne funkcije i relacije nad njim i u stanju smo da za bilo koje elemente tog skupa efektivno izračunamo vrednosti tih funkcija i efektivno odredimo da li je zadovoljena neka relacija (na primer, u stanju smo da efektivno proverimo da li važi  $3 > 5$ , to je jednostavan uslov nad celim brojevima). Slika 9.2 ilustruje dve pomene interpretacije, na svakoj slici levo je svet sintakse, a desno — svet semantike, levo su termovi i formule, a desno — njihovo izabrano značenje. U interpretaciji ilustrovanoj na slici 9.2 desno, promenljivim  $x$  i  $y$  pridružene su vrednosti 3 i 5, a predikatskom simbolu  $\text{above}$  relacija  $>$ , te se formula  $\text{above}(x, y)$  preslikava u vrednost 0, jer  $3 > 5$  je netačno nad celim brojevima.

Uvedimo sada pojam interpretacije precizno. Interpretacija je određena  $\mathcal{L}$ -strukturom (univerzumom i izabranim značenjem funkcijskih i predikatskih simbola) i *valuacijom* (vrednošću promenljivih).

<sup>2</sup>Po prirodi pridruženog značenja, promenljivim iskazne logike u logici prvog reda ne odgovaraju promenljive, već predikatski simboli arnosti nula (koji se preslikavaju u skup  $\{0, 1\}$ ).



Slika 9.2: Ilustracija za semantiku logike prvog reda.

**Definicija 9.6 ( $\mathcal{L}$ -struktura).** Za datu signaturu  $\mathcal{L}$ ,  $\mathcal{L}$ -struktura  $\mathfrak{D}$  je par  $(D, I)$ , gde je  $D$  skup, a  $I$  funkcija i važi:

- $D$  je neprazan skup i zovemo ga univerzum (ponekad i domen ili nosač);
- svakom simbolu konstante  $c$  iz  $\mathcal{L}$  (tj. svakom funkcijском simbolu arnosti 0), funkcija  $I$  pridružuje jedan element  $c_I$  iz  $D$ ;
- svakom funkcijском simbolu  $f$  iz  $\mathcal{L}$  arnosti  $n$ ,  $n > 0$ , funkcija  $I$  pridružuje jednu totalnu funkciju  $f_I$  iz  $D^n$  u  $D$ ;
- svakom predikatsком simbolu  $p$  iz  $\mathcal{L}$  arnosti  $n$ ,  $n > 0$ , funkcija  $I$  pridružuje jednu relaciju, tj. totalnu funkciju  $p_I$  iz  $D^n$  u  $\{0, 1\}$ .

**Primer 9.8.** Za signaturu  $\mathcal{L} = (\{\}, \{\text{above}_{/2}, \text{below}_{/2}\})$  (iz primera 9.2), jedna moguća  $\mathcal{L}$ -struktura je  $(\mathbf{K}, I)$ , gde je  $\mathbf{K}$  konkretan skup kutija prikazanih na slici 9.1. Predikatski simboli  $\text{above}$  i  $\text{below}$  preslikavaju se u relacije „jeste iznad“ i „jeste ispod“ nad tim konkretnim kutijama, u konkretnom odnosu koji imaju.

**Primer 9.9.** Za signaturu  $\mathcal{L} = (\{e_{/0}, \star_{/2}\}, \{\prec_{/2}\})$ , iz primera 9.3, jedna moguća  $\mathcal{L}$ -struktura je par  $(\mathbf{N}, I)$ , gde je  $\mathbf{N}$  skup prirodnih brojeva, a  $I$  funkcija koja simbol  $e$  preslikava u prirodni broj 1, funkcijski simbol  $\star$  u operaciju množenja prirodnih brojeva, a predikatski simbole  $\prec$  u relaciju  $<$  nad prirodnim brojevima.

**Definicija 9.7 (Valuacija).** Valuacija  $v$  za skup promenljivih  $V$  u odnosu na domen  $D$  je preslikavanje koje svakom elementu iz  $V$  dodeljuje jedan element iz  $D$ . Ako je  $v(x_i) = d_j$ , onda kažemo da je  $d_j$  vrednost promenljive  $x_i$  u valuaciji  $v$ .

Ako su  $v$  i  $w$  valuacije za isti skup promenljivih i u odnosu na isti domen, onda sa  $v \sim_x w$  označavamo da je  $v(y) = w(y)$  za svaku promenljivu  $y$  različitu od  $x$ , pri čemu vrednosti  $v(x)$  i  $w(x)$  mogu a ne moraju biti iste.

**Primer 9.10.** Za domen iz primera 9.8, valuacija  $v$  može da preslikava promenljive  $x$  i  $y$  u (konkretnе) kutije  $\boxed{B}$  i  $\boxed{A}$ , tj.  $v(x) = \boxed{B}$  i  $v(y) = \boxed{A}$ .

Interpretacija proširuje valuaciju i pridružuje značenje i drugim termovima i formulama.

**Definicija 9.8 (Interpretacija).** Vrednost (ili značenje) terma  $t$  u interpretaciji  $I_v$ , određenoj  $\mathcal{L}$ -strukturom  $\mathfrak{D}$  i valuacijom  $v$ , označavamo sa  $I_v(t)$  i definišemo na sledeći način:

ako je term  $t$  jednak nekoj promenljivoj  $x$ , onda je  $I_v(t) = v(x)$ ;

- ako je term  $t$  jednak nekom simbolu konstante  $c$ , onda je  $I_v(t) = c_I$ ;
- ako je term  $t$  jednak  $f(t_1, t_2, \dots, t_n)$  onda je  $I_v(t) = f_I(I_v(t_1), I_v(t_2), \dots, I_v(t_n))$ .

Istinitosnu vrednost (ili značenje) formule u interpretaciji  $I_v$  određenoj  $\mathcal{L}$ -strukturom  $\mathfrak{D}$  i valuacijom  $v$ , definišemo na sledeći način:

$$I_v(\top) = 1 \text{ i } I_v(\perp) = 0;$$

- $I_v(p(t_1, t_2, \dots, t_n)) = p_I(I_v(t_1), I_v(t_2), \dots, I_v(t_n));$
- $I_v(\neg \mathcal{A}) = \begin{cases} 1, & \text{ako je } I_v(\mathcal{A}) = 0 \\ 0, & \text{inače} \end{cases}$
- $I_v(\mathcal{A} \wedge \mathcal{B}) = \begin{cases} 1, & \text{ako je } I_v(\mathcal{A}) = 1 \text{ i } I_v(\mathcal{B}) = 1 \\ 0, & \text{inače} \end{cases}$
- $I_v(\mathcal{A} \vee \mathcal{B}) = \begin{cases} 0, & \text{ako je } I_v(\mathcal{A}) = 0 \text{ i } I_v(\mathcal{B}) = 0 \\ 1, & \text{inače} \end{cases}$
- $I_v(\mathcal{A} \Rightarrow \mathcal{B}) = \begin{cases} 0, & \text{ako je } I_v(\mathcal{A}) = 1 \text{ i } I_v(\mathcal{B}) = 0 \\ 1, & \text{inače} \end{cases}$
- $I_v(\mathcal{A} \Leftrightarrow \mathcal{B}) = \begin{cases} 1, & \text{ako je } I_v(\mathcal{A}) = I_v(\mathcal{B}) \\ 0, & \text{inače} \end{cases}$
- $I_v((\forall x)\mathcal{A}) = \begin{cases} 1, & \text{ako za svaku valuaciju } w \text{ takvu da je } w \sim_x v \text{ važi } I_w(\mathcal{A}) = 1 \\ 0, & \text{inače} \end{cases}$
- $I_v((\exists x)\mathcal{A}) = \begin{cases} 1, & \text{ako postoji valuacija } w \text{ takva da je } w \sim_x v \text{ i } I_w(\mathcal{A}) = 1 \\ 0, & \text{inače} \end{cases}$

Objasnimo dodatno definiciju značenja za  $I_v((\exists x)\mathcal{A})$ . Cilj je određivanje vrednosti formule  $(\exists x)\mathcal{A}$  svesti na određivanje vrednosti jednostavnije formule, formule  $\mathcal{A}$ . Formula  $\mathcal{A}$  zavisi od promenljive  $x$  ali i od nekih drugih promenljivih (od svih promenljivih koje u njoj imaju slobodna pojavljivanja). Valuacija  $v$  u svim tim promenljivim dodeljuje neke vrednosti. I intuitivno je očekivano da istinitosna vrednost formule  $(\exists x)\mathcal{A}$  u interpretaciji  $I_v$  ne zavisi od  $v(x)$ , ali da zavisi od vrednosti drugih slobodnih promenljivih u  $v$ . Razmotrimo primer  $(\exists x)p(x, y)$  i pitanje da li je ova formula tačna u interpretaciji  $v$ . Intuitivno, tačna je ako možemo pogodno da izaberemo vrednost za  $x$ , da je vrednost  $y$  nepromenjena, određena valuacijom  $v$  i da je za takve vrednosti promenljivih formula  $p(x, y)$  tačna. Ako smo nekako pogodno izabrali vrednost za  $x$ , a zadržali vrednost za  $y$ , time smo, zapravo, definisali novu valuaciju –  $w$ , koja ima istu vrednost za  $y$  kao i valuacija  $v$ . Dakle, ako smo mogli da izaberemo takvu valuaciju, onda je  $I_v((\exists x)p(x, y)) = 1$ . Opšti slučaj – slučaj  $(\exists x)\mathcal{A}$  – je analogan: ako možemo da izaberemo vrednost za  $x$ , zadržavši vrednosti promenljivih kao u  $v$ , tako da je formula  $\mathcal{A}$  tačna, onda je  $I_v((\exists x)\mathcal{A}) = 1$ . Drugim rečima, ako postoji valuacija  $w$  takva da je  $w \sim_x v$  i  $I_w(\mathcal{A}) = 1$ , onda je  $I_v((\exists x)\mathcal{A}) = 1$ . Analogno objašnjenje važi i za definiciju značenja za  $I_v((\forall x)\mathcal{A})$ .

Može se dokazati da je datom definicijom svakoj formuli  $\mathcal{A}$  nad signaturom  $\mathcal{L}$  i skupom  $V$  pridružena (jedinstvena) vrednost  $I_v(\mathcal{A})$ . Primetimo da  $I_v(\mathcal{A})$  zavisi od  $v(x)$  samo ako promenljiva  $x$  ima slobodna pojavljivanja u formuli  $\mathcal{A}$ . Vrednost  $I_v(\mathcal{A})$ , dakle, zavisi samo od slobodnih promenljivih u formuli  $\mathcal{A}$ . Specijalno, ako je  $\mathcal{A}$  rečenica, vrednost  $I_v(\mathcal{A})$  uopšte ne zavisi od  $v$ .

**Primer 9.11.** Za  $L$ -strukturu iz primera 9.8, ako je, na primer,  $v(x) = \boxed{B}$  i  $v(y) = \boxed{A}$ , onda je  $I_v(\text{above}(x, y)) = \text{„jeste iznad“}(I_v(x), I_v(y)) = \text{„jeste iznad“}(\boxed{B}, \boxed{A}) = 1$ , jer za konkretnu gomilu kuhinja sa slike 9.1 jeste tačno da je  $\boxed{B}$  iznad  $\boxed{A}$ . Slično,  $I_v(\text{above}(y, x)) = \text{„jeste iznad“}(I_v(y), I_v(x)) = \text{„jeste iznad“}(\boxed{A}, \boxed{B}) = 0$ .

Ako je, na primer,  $v(x) = \boxed{C}$  i  $v(y) = \boxed{A}$ , onda je  $I_v(\text{above}(x, y)) = \text{„jeste iznad“}(I_v(x), I_v(y)) = \text{„jeste iznad“}(\boxed{C}, \boxed{A}) = 0$ .

**Primer 9.12.** Moguća  $\mathcal{L}$ -struktura za signaturu  $\mathcal{L} = (\{\}, \{\text{above}_{/2}, \text{below}_{/2}\})$  (iz primera 9.2) je i  $(\mathbf{Z}, I)$ , gde je  $\mathbf{Z}$  skup celih brojeva, a  $I$  je funkcija koja predikske simbole  $\text{above}$  i  $\text{below}$  preslikava u relacije  $>$  i  $<$  nad celim brojevima.

Ako je, na primer,  $v(x) = 3$  i  $v(y) = 5$ , onda je  $I_v(\text{above}(x, y))$  jednako  $>(I_v(x), I_v(y))$  tj.  $>(3, 5) = 0$ , jer za cele brojeve 3 i 5 je  $3 > 5$  netačno.

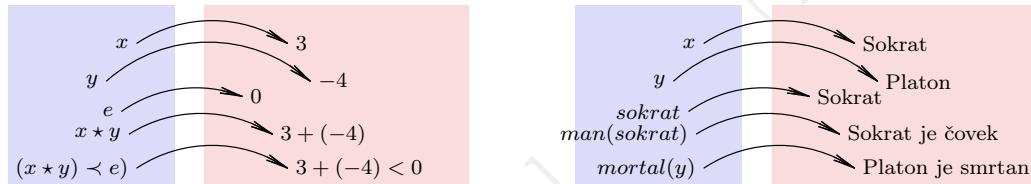
Vrednost  $I_v(\exists x \text{ above}(x, x))$  jednaka je 0, jer ne postoji valuacija  $w$  za koju važi  $w \sim_x v$  (kako nema relevantnih promenljivih sem  $x$ , ovo ograničenje nije relevantno) takva da je  $I_w(\text{above}(x, x)) = 1$ , jer ne postoji ceo broj  $c$  (jednak  $w(x)$ ) takav da je  $c > c$ .

Vrednost  $I_v(\forall x \neg \text{above}(x, x))$  jednaka je 1, jer u svakoj valuaciji  $w$  važi da je  $I_w(\neg \text{above}(x, x)) = 1$ , tj.  $I_w(\text{above}(x, x)) = 0$ , jer je  $c > c$  netačno za svaki ceo broj  $c$ .

**Primer 9.13.** Za signaturu  $\mathcal{L} = (\{e_{/0}, \star_{/2}\}, \{\prec_{/2}\})$ , iz primera 9.3, jedna moguća  $\mathcal{L}$ -struktura je par  $(\mathbf{Z}, I)$ , gde je  $\mathbf{Z}$  skup celih brojeva, a  $I$  funkcija koja simbol  $e$  preslikava u ceo broj 0, funkcijski simbol  $\star$  u operaciju sabiranja nad celim brojevima, a predikatski simboli  $\prec$  u relaciju  $<$  nad celim brojevima. Ako je, na primer,  $v(x) = 3$ ,  $v(y) = -4$ , onda je (koristeći infiksni zapis)  $I_v((x \star y) \prec e)$  jednako  $I_v(x) + I_v(y) < I_v(0)$ , tj.  $3 + (-4) < 0$ , tj. jednako 1 (slika 9.3, levo).

Za istu signaturu moguća je slična interpretacija, sa univerzumom koji je skup prirodnih, a ne celih brojeva.

Za istu signaturu jedna  $\mathcal{L}$ -struktura je i par  $(\mathbf{D}, I)$ , gde je  $\mathbf{D}$  skup dana u nedelji – {ponedeljak, utorak, sreda, četvrtak, petak, subota, nedelja}, a  $I$  funkcija koja simbol  $e$  preslikava (na primer) u element nedelja, simbol  $\prec$  u relaciju prethodni dan, itd.



Slika 9.3: Ilustracija za semantiku logike prvog reda sa primerima skupa celih brojeva i skupa svih živih bića kao univerzumima.

**Primer 9.14.** Za signaturu  $\mathcal{L} = (\{\text{sokrat}_{/0}\}, \{\text{man}_{/1}, \text{mortal}_{/1}\})$  iz primera 9.4 jedna moguća  $\mathcal{L}$ -struktura je par  $(\mathbf{D}, I)$ , gde je  $\mathbf{D}$  skup svih živih bića, a  $I$  funkcija koja simbol  $\text{sokrat}$  preslikava (na primer) u osobu Sokrat, predikatski simbol  $\text{man}$  u relaciju „biti čovek”, predikatski simbol  $\text{mortal}$  u relaciju „biti smrtan” (slika 9.3, desno).

Za istu signaturu jedna  $\mathcal{L}$ -struktura je i par  $(\mathbf{N}, I)$ , gde je  $\mathbf{N}$  skup prirodnih brojeva, a  $I$  funkcija koja simbol  $\text{sokrat}$  preslikava (na primer) u broj 0, predikatski simbol  $\text{man}$  u unarnu relaciju „biti složen broj”, predikatski simbol  $\text{mortal}$  u unarnu relaciju „biti paran broj”.

Sve navedeno još uvek deluje čudno, pa i beskorisno: ukoliko formulama možemo pridružiti skoro proizvoljno značenje, kakva korist može biti od njih? Postoji nekoliko odgovora na ovo pitanje. Jedan odgovor je da se u razmatranju formula u okviru neke primene često razmatra samo jedna konkretna vrsta interpretacija (na primer, nad celim brojevima), prilagođena konkretnoj primeni. Drugi odgovor je da se u mnogim primenama razmatra pitanje da li je neka formula valjana, tj. tačna u svakoj mogućoj interpretaciji. U takvim primenama, formule su konstruisane na način koji eliminiše sve nerelevantne modele. O oba scenarija biće reči u daljem tekstu.

**Definicija 9.9** (Zadovoljivost i valjanost). Ako je interpretacija  $I_v$  određena  $\mathcal{L}$ -strukturom  $\mathfrak{D}$  i valuacijom  $v$  i ako za  $\mathcal{L}$ -formulu  $\mathcal{A}$  važi  $I_v(\mathcal{A}) = 1$ , onda kažemo da je formula  $\mathcal{A}$  tačna u interpretaciji  $I_v$ , da je par  $(\mathfrak{D}, v)$  model (eng. model) formule  $\mathcal{A}$  i pišemo  $(\mathfrak{D}, v) \models \mathcal{A}$ .

$\mathcal{L}$ -formula  $\mathcal{A}$  je zadovoljiva (eng. satisfiable) ako postoje  $\mathcal{L}$ -struktura  $\mathfrak{D}$  i valuacija  $v$  takve da važi  $(\mathfrak{D}, v) \models \mathcal{A}$ , a inače kažemo da je kontradiktorna (eng. contradictory) ili nezadovoljiva (eng. unsatisfiable).

Ako je za neku  $\mathcal{L}$ -strukturu  $\mathfrak{D}$  važi  $(\mathfrak{D}, v) \models \mathcal{A}$  za svaku valuaciju  $v$ , onda kažemo da je formula  $\mathcal{A}$  valjana (eng. valid) u  $\mathfrak{D}$ , da je  $\mathfrak{D}$  model formule  $\mathcal{A}$  i pišemo  $\mathfrak{D} \models \mathcal{A}$ .

Ako je  $\mathcal{L}$ -formula  $\mathcal{A}$  valjana u svakoj  $\mathcal{L}$ -strukturi, onda za tu formulu kažemo da je valjana i pišemo  $\models \mathcal{A}$ .

Analogne definicije uvodimo za skupove formula (na primer, skup formula  $\Gamma$  je *konzistentan* (ili *zadovoljiv*) ako ima bar jedan model, (a inače kažemo da je skup  $\Gamma$  *nekonzistentan*, *nezadovoljiv*, ili *kontradiktoran*).

**Primer 9.15.** Formula  $\exists x(x \prec e)$  tačna je u prvoj interpretaciji iz primera 9.13 (za univerzum celih brojeva), a nije tačna u drugoj interpretaciji (za univerzum prirodnih brojeva). Dakle, ona nije valjana.

**Primer 9.16.** Formula  $\forall x(\text{man}(x) \Rightarrow \text{mortal}(x))$  je tačna u prvoj interpretaciji iz primera 9.14 (za univerzum koji čine sva živa bića), a nije tačna u drugoj interpretaciji (za univerzum prirodnih brojeva). Dakle, ona nije valjana.

**Primer 9.17.** Formule  $\forall x(p(x) \Rightarrow q(y))$  i  $(\forall x p(x)) \Rightarrow q(y)$  koje se razlikuju po dosegu kvantifikatora  $\forall x$  mogu imati različita značenja.

Jedna moguća interpretacija ovih formula određena je domenom koji čine svi studenti koji pohađaju neki kurs,  $p(x)$  se interpretira kao „student  $X$  će položiti ispit, a  $q(y)$  se interpretira kao „student  $Y$  će se obradovati“ (gde su  $X$  i  $Y$  vrednosti promenljivih  $x$  i  $y$  u interpretaciji). Prva formula se, onda, interpretira kao „za bilo kojeg studenta važi: ako taj student položi ispit,  $Y$  će se obradovati“, tj. „ako neki student položi ispit,  $Y$  će se obradovati“, što odgovara i značenju formule  $(\exists x p(x)) \Rightarrow q(y)$ , a druga kao „ako svaki student položi ispit,  $Y$  će se obradovati“.

**Primer 9.18.** Tvrđnja „postoji država koja se graniči sa Italijom i Austrijom“ može se formulisati u terminima logike prvog reda na sledeći način:

$\exists x (\text{država}(x) \wedge \text{graničeSe}(\text{italija}, x) \wedge \text{graničeSe}(\text{austrija}, x))$   
gde su italija i austrija simboli konstanti, a država/1 i graničeSe/2 su predikatski simboli.

Univerzum za prirodnu interpretaciju čini skup svih država sveta, predikatski simboli su interpretirani na prirodan način i u toj interpretaciji nавена formula je valjana.

Već i na osnovu relativno jednostavnih primera, vidi se da ispitivanje valjanosti ili valjanosti u nekoj  $\mathcal{L}$ -strukturi neposredno, na osnovu definicije, može biti veoma mukotrpno. I zaista, takvo ispitivanje se i ne koristi u primenama. Umesto toga, obično se koriste namenske procedure opisane u narednim poglavljima. One implicitno koriste definicije valjanosti i u stanju su da utvrde da li je neka formula valjana.

## 9.2 Logičke posledice i logički ekvivalentne formule

Često je veoma važno pitanje da li je neka formula posledica nekih drugih formula. Ovo pitanje može se opisati u terminima pojma *logičke posledice*, analogno kao u slučaju iskazne logike.

**Definicija 9.10** (Logička posledica i logička ekvivalencija). Kažemo da je formula  $\mathcal{A}$  logička posledica (eng. logical consequence) skupa formula  $\Gamma$  nad istom signaturom  $\mathcal{L}$  i pišemo  $\Gamma \models \mathcal{A}$ , ako je svaki model za  $\Gamma$  istovremeno i model za  $\mathcal{A}$ .

Kažemo da su formule  $\mathcal{A}$  i  $\mathcal{B}$  nad istom signaturom  $\mathcal{L}$  logički ekvivalentne (eng. logically equivalent) i pišemo  $\mathcal{A} \equiv \mathcal{B}$  ako važi  $\{\mathcal{A}\} \models \mathcal{B}$  i  $\{\mathcal{B}\} \models \mathcal{A}$ .

Ako ne važi  $\Gamma \models \mathcal{A}$ , onda to zapisujemo  $\Gamma \not\models \mathcal{A}$ .

Kao i u iskaznoj logici, formula je logička posledica praznog skupa formula ako i samo ako je valjana. Ako je formula  $\mathcal{A}$  logička posledica praznog skupa formula, onda umesto  $\{\} \models \mathcal{A}$ , to zapisujemo kraće  $\models \mathcal{A}$ , baš kao što već zapisujemo valjane formule.

Kao i u iskaznoj logici, ako je skup  $\Gamma$  kontradiktoran, onda je svaka formula njegova logička posledica. Specijalno, svaka formula je logička posledica skupa  $\{\perp\}$ .

Ako je skup  $\Gamma$  konačan, onda umesto  $\{\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_k\} \models \mathcal{A}$  pišemo kraće  $\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_k \models \mathcal{A}$ .

**Primer 9.19.** Ako sa  $\Gamma$  označimo skup formula (iz primera 9.2):

$\{ \forall x \forall y (\text{above}(x, y) \Rightarrow \neg \text{above}(y, x)), \forall x \forall y (\text{above}(x, y) \Leftrightarrow \text{below}(y, x)), \forall x \forall y \forall z (\text{above}(x, y) \wedge \text{above}(y, z) \Rightarrow$

$\text{above}(x, z)) \},$

onda se može pokazati da važi:  $\Gamma \models \forall x \forall y \forall z (\text{above}(y, x) \wedge \text{below}(z, x) \Rightarrow \text{above}(y, z)).$

Ako važi  $\mathcal{A} \equiv \mathcal{B}$ , tj. ako je svaki model za  $\mathcal{A}$  istovremeno i model za  $\mathcal{B}$  i obratno, onda u bilo kojoj valuatoriji formule  $\mathcal{A}$  i  $\mathcal{B}$  imaju jednake vrednosti. Tvrđenja oblika  $\mathcal{A} \equiv \mathcal{B}$  zovemo *logičkim ekvivalentcijama*. Relacija  $\equiv$  je, očigledno, relacija ekvivalentcije nad skupom formula.

Naredna teorema daje vezu između meta i objektnog nivoa tj. govori da se uslov logičke posledice može svesti na uslov valjanosti i obratno. Ako postoji efektivan način za rešavanje jednog problema onda postoji i efektivan način za rešavanje drugog.

**Teorema 9.1.** Za formule  $\mathcal{A}, \mathcal{B}, \mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_k$ , nad istom signaturom  $\mathcal{L}$  važi:

- $\mathcal{A} \models \mathcal{B}$  akko je formula  $\mathcal{A} \Rightarrow \mathcal{B}$  valjana;
- $\mathcal{A} \equiv \mathcal{B}$  akko je formula  $\mathcal{A} \Leftrightarrow \mathcal{B}$  valjana;
- $\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_k \models \mathcal{A}$  akko  $\mathcal{B}_1 \wedge \mathcal{B}_2 \wedge \dots \wedge \mathcal{B}_k \models \mathcal{A}$  tj.  
akko je formula  $\mathcal{B}_1 \wedge \mathcal{B}_2 \wedge \dots \wedge \mathcal{B}_k \Rightarrow \mathcal{A}$  valjana tj.  
akko je formula  $\mathcal{B}_1 \wedge \mathcal{B}_2 \wedge \dots \wedge \mathcal{B}_k \wedge \neg \mathcal{A}$  nezadovoljiva.

Ako se u proizvoljnoj logičkoj ekvivalentciji iskazne logike, svaka iskazna formula zameni formulom logike prvog reda, biće očigledno dobijena logička ekvivalentcija logike prvog reda, na primer,  $\neg \neg \mathcal{A} \equiv \mathcal{A}$ . Postoje i logičke ekvivalentcije logike prvog reda koje ne mogu biti dobijene na taj način.

**Primer 9.20.** Neke od shema logičkih ekvivalentcija logike prvog reda su:

$\neg(\exists x)\mathcal{A}$	$\equiv$	$(\forall x)\neg\mathcal{A}$	De Morganov zakon
$\neg(\forall x)\mathcal{A}$	$\equiv$	$(\exists x)\neg\mathcal{A}$	De Morganov zakon
$\forall x \forall y \mathcal{A}$	$\equiv$	$\forall y \forall x \mathcal{A}$	zamena poretku kvantifikatora
$\exists x \exists y \mathcal{A}$	$\equiv$	$\exists y \exists x \mathcal{A}$	zamena poretku kvantifikatora
$(\exists x)(\mathcal{A} \vee \mathcal{B})$	$\equiv$	$(\exists x)\mathcal{A} \vee (\exists x)\mathcal{B}$	zakon distributivnosti $\exists$ prema $\vee$
$(\forall x)(\mathcal{A} \wedge \mathcal{B})$	$\equiv$	$(\forall x)\mathcal{A} \wedge (\forall x)\mathcal{B}$	zakon distributivnosti $\forall$ prema $\wedge$
$(\exists x)(\mathcal{A} \wedge \mathcal{B})$	$\equiv$	$(\exists x)\mathcal{A} \wedge \mathcal{B}$	zakon distributivnosti $\exists$ prema $\wedge$ (pri čemu $\mathcal{B}$ ne sadrži slobodna pojavljivanja promenljive $x$ )
$(\forall x)(\mathcal{A} \vee \mathcal{B})$	$\equiv$	$(\forall x)\mathcal{A} \vee \mathcal{B}$	zakon distributivnosti $\forall$ prema $\vee$ (pri čemu $\mathcal{B}$ ne sadrži slobodna pojavljivanja promenljive $x$ )
$(\forall x)\mathcal{A}$	$\equiv$	$(\forall y)(\mathcal{A}[x \mapsto y])$	zakon o preimenovanju vezane promenljive (pri čemu $\mathcal{A}$ ne sadrži slobodna pojavljivanja promenljive $y$ )
$(\exists x)\mathcal{A}$	$\equiv$	$(\exists y)(\mathcal{A}[x \mapsto y])$	zakon o preimenovanju vezane promenljive (pri čemu $\mathcal{A}$ ne sadrži slobodna pojavljivanja promenljive $y$ )

Logička ekvivalentcija  $\forall x \forall y \mathcal{A} \equiv \forall y \forall x \mathcal{A}$  govori da dva univerzalna kvantifikatora mogu da zamene mesta, a isto važi i za egzistencijalne kvantifikatore. To suštinski znači da u bloku kvantifikatora iste vrste poredak tih kvantifikatora nije bitan. S druge strane, univerzalni i egzistencijalni kvantifikator ne mogu, u opštem slučaju, da razmenjuju mesta. Na primer, formule  $(\forall x)(\exists y)\mathcal{A}$  i  $(\exists y)(\forall x)\mathcal{A}$  nisu u opštem slučaju logički ekvivalentne.

Logičke ekvivalentcije  $\forall x \mathcal{A} \equiv \forall y \mathcal{A}[x \mapsto y]$  i  $\exists x \mathcal{A} \equiv \exists y \mathcal{A}[x \mapsto y]$  govore o tome da se vezane promenljive mogu preimenovati bez uticaja na istinitosnu vrednost formule, kao što je ranije već nagovešteno.

Naredna teorema kaže da ako se u formuli  $\mathcal{A}$  zameni neka njena potformula logički ekvivalentnom formulom, biće dobijena formula koja je logički ekvivalentna formuli  $\mathcal{A}$ .

**Teorema 9.2** (Teorema o zameni). Ako važi  $\mathcal{B}_1 \equiv \mathcal{B}_2$ , onda je  $\mathcal{A} \equiv \mathcal{A}[\mathcal{B}_1 \mapsto \mathcal{B}_2]$ .

**Primer 9.21.** Na osnovu logičkih ekvivalencija iz primera 8.14 i 9.20 važi  $\neg(\exists x)(\mathcal{A} \wedge \neg\mathcal{B}) \equiv (\forall x)\neg(\mathcal{A} \wedge \neg\mathcal{B}) \equiv (\forall x)(\neg\mathcal{A} \vee \neg\neg\mathcal{B}) \equiv (\forall x)(\neg\mathcal{A} \vee \mathcal{B}) \equiv (\forall x)(\mathcal{A} \Rightarrow \mathcal{B})$ .

Iz  $\neg(\exists x)(\mathcal{A} \wedge \neg\mathcal{B}) \equiv (\forall x)(\mathcal{A} \Rightarrow \mathcal{B})$ , na osnovu teoreme 9.1 sledi da je formula  $\neg(\exists x)(\mathcal{A} \wedge \neg\mathcal{B}) \Leftrightarrow (\forall x)(\mathcal{A} \Rightarrow \mathcal{B})$  valjana.

### 9.3 Ispitivanje zadovoljivosti i valjanosti u logici prvog reda

U praktičnim primenama, onda kada je neki cilj formulisan na jeziku logike prvog reda, centralni problem postaje ispitivanje da li je neka iskazna formula valjana, da li je zadovoljiva i slično. Početno pitanje je da li su ti problemi uopšte odlučivi, tj. da li postoje algoritmi koji uvek mogu da daju odgovor na njih. Ukoliko takvi algoritmi ne postoje, pitanje je da li postoje za neke klase formule logike prvog reda. Slično kao u iskaznoj logici, problemi ispitivanja valjanosti i zadovoljivosti su povezani: formula je valjana ako i samo ako je njena negacija nezadovoljiva.

#### 9.3.1 Logika prvog reda i odlučivost

Za razliku od iskazne logike, u logici prvog reda problem ispitivanja valjanosti nije odlučiv, tj. ne postoji algoritam koji za proizvoljnu formulu može da utvrdi da li je valjana ili nije. Naravno, isto važi i za problem ispitivanja (ne)zadovoljivosti. Ipak, ovi problem su poluodlučivi: postoji algoritam koji za proizvoljnu valjanu formulu može da utvrdi da je valjana, ali ne može za proizvoljnu formulu koja nije valjana da utvrdi da nije valjana (obično se u takvim situacijama algoritam ne zaustavlja). Kako je formula valjana akko je njena negacija nezadovoljiva, očito je i problem nezadovoljivosti poluodlučiv: postoji algoritam koji za proizvoljnu nezadovoljivu formulu može da utvrdi da je nezadovoljiva, ali ne može za proizvoljnu zadovoljivu formulu da utvrdi da je zadovoljiva. Kako je problem ispitivanja valjanosti neodlučiv, neodlučiv je (mada je poluodlučiv) i problem ispitivanja da li važi  $\Gamma \models \mathcal{A}$  (za proizvoljan skup  $\Gamma$  i formulu  $\mathcal{A}$ ).

U praktičnim primenama, obično je relevantna samo određena klasa modela (na primer, za univerzum koji čine realni brojevi, ili građani neke države, ili skup kutija u nekoj sobi). Ukoliko se razmatra samo jedna vrsta modela (tj. pitanje  $\mathcal{D} \models \mathcal{A}$ ), onda za neke vrste formula ispitivanje valjanosti može da bude odlučiv problem. Na primer, ako se razmatraju formule nad signaturom (opisanom u primeru 9.2)  $\mathcal{L} = (\{\}, \{above_{/2}, below_{/2}\})$  i i samo  $\mathcal{L}$ -struktura (opisana u primeru 9.8) u kojoj je univerzum skup konkretnih kutija prikazanih na slici 9.1, onda je tako specijalizovan problem ispitivanja valjanosti odlučiv (može se dizajnirati jednostavan algoritam koji rešava ovaj problem). Razmotrimo i nešto složeniji primer: razmotrimo signaturu  $\mathcal{L} = (\{0_{/0}, s_{/1}, +_{/2}\}, \{<_{/2}, =_{/2}\})$  i  $\mathcal{L}$ -strukturu  $(\mathbf{Z}, I)$ , gde je  $\mathbf{Z}$  skup celih brojeva, a  $I$  funkcija koja (prirodno) simbol 0 preslikava u ceo broj 0, simbol  $s$  preslikava u operaciju sledbenik, funkcionalni simbol  $+$  u operaciju sabiranja nad celim brojevima, a predikatske simbole  $<$  i  $=$  u relacije  $<$  i  $=$  nad celim brojevima. Ovakva  $\mathcal{L}$ -struktura određuje teoriju koju zovemo *linearna aritmetika nad celim brojevima* i označavamo *LIA*. Ako je formula  $\mathcal{A}$  valjana u ovoj  $\mathcal{L}$ -strukturi, onda to zapisujemo  $\models_{LIA} \mathcal{A}$ . Na primer, valjane su u *LIA* formule  $\forall x \exists y (x < y + s(0))$  i  $x + y = y + x$  (iako nisu valjane u nekim drugim interpretacijama). Ispitivanje da li važi  $\models_{LIA} \mathcal{A}$  je odlučiv problem. Za ispitivanje valjanosti u ovom i sličnim situacijama koriste se specijalizovani algoritmi, koje obično zovemo SMT rešavači (od engleskog *satisfiability modulo theory* — zadovoljivost u odnosu na teoriju, jer oni obično problem valjanosti svode na problem zadovoljivosti). Rešavač za ispitivanje  $\models_{LIA} \mathcal{A}$  zovemo SMT rešavač za teoriju *LIA*. Postoji mnoštvo takvih rešavača i obično su izgrađeni oko nekog SAT rešavača. Neki od često korišćenih SMT rešavača su rešavači za linearnu aritmetiku nad celim brojevima, za linearnu aritmetiku nad racionalnim brojevima, za teoriju neinterpretiranih funkcija, za teoriju nizova, za teoriju lista, itd. SMT rešavači imaju brojne primene, posebno u verifikaciji softvera i hardvera i u rešavanju raznovrsnih problema ograničenja.

Često nije lako napraviti rešavač za neku specifičnu teoriju (i nameravanu interpretaciju). Umesto toga, specifičnosti željenog značenja formula mogu se nametnuti u vidu skupa uslova  $\Gamma$ . Tada se ispitivanje valjanosti formule  $\mathcal{A}$ , svodi na ispitivanje da li važi  $\Gamma \models \mathcal{A}$ , tj. na ispitivanje da li je  $\Gamma \Rightarrow \mathcal{A}$  valjana formula. Time se od specifičnog razmatranja došlo do opštег razmatranja i valjanost navedene formule ispitivaće se nekom opštom procedurom za ispitivanje valjanosti u logici prvog reda. Takve procedure zovu se *dokazivači za logiku prvog reda* (a možemo da ih, neformalno, zovemo i *univerzalnim rešavačima za logiku prvog reda*).

Postoje načini koji omogućavaju ispitivanje valjanosti i zadovoljivosti ne neposredno na osnovu definicije, nego jednostavnijim, namenskim procedurama od kojih je jedna opisana u nastavku. Zahvaljujući takvoj opštoj proceduri, za mnoge valjane formule može se efikasno dokazati da su valjane. Za takve formule, onda sledi da su valjane i u svakoj konkretnoj  $\mathcal{L}$ -strukturi.

Dokazivanje tvrđenja

$\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_n \models \mathcal{A}$

obično se svodi na dokazivanje da je nezadovoljiva naredna formula

$$\mathcal{B}_1 \wedge \mathcal{B}_2 \wedge \dots \wedge \mathcal{B}_n \wedge \neg \mathcal{A}$$

Metod rezolucije koji će biti opisan u nastavku je metod za ispitivanje nezadovoljnosti (ali, na opisani način, koristi se i za ispitivanje valjanosti).

### 9.3.2 Normalne forme

Iako se zadovoljivost i valjanost mogu ispitivati za formule proizvoljnog oblika, daleko je jednostavnije algoritme ispitivanja formulirati za formule nekog posebnog oblika. Zbog toga se definišu *normalne forme* i algoritmi kojima se neka formula transformiše u te normalne forme. Pod *transformacijom* se podrazumeva konstruisanje formule koja je, na primer, logički ekvivalentna polaznoj formuli i zadovoljava neka sintaksička ograničenja. U nastavku će biti opisan niz koraka koji vodi od formule čija se zadovoljivost razmatra do formule koja ima specifičnu sintaksičku formu i zadovoljiva je ako i samo ako je zadovoljiva polazna formula.

**Definicija 9.11** (Preneks normalna forma). *Kažemo da je formula u preneks normalnoj formi ako je ona oblika*

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n \mathcal{A}$$

pri čemu je  $Q_i$  ili  $\forall$  ili  $\exists$  i  $\mathcal{A}$  ne sadrži kvantifikatore, kao ni slobodne promenljive osim (eventualno) promenljivih  $x_1, x_2, \dots, x_n$ .

Ako je rečenica (zatvorena formula)  $\mathcal{A}$  logički ekvivalentna formuli  $\mathcal{B}$  i formula  $\mathcal{B}$  je u preneks normalnoj formi, onda kažemo da je formula  $\mathcal{B}$  preneks normalna forma formule  $\mathcal{A}$ . Jedna formula može da ima više preneks normalnih formi (na primer, i formula  $(\forall x)(\forall y)(\mathcal{A}(x) \wedge \mathcal{B}(y))$  i formula  $(\forall y)(\forall x)(\mathcal{B}(y) \wedge \mathcal{A}(x))$  su preneks normalne forme formule  $(\forall x)\mathcal{A}(x) \wedge (\forall y)\mathcal{B}(y)$ ). Slično, jedna formula koja je u preneks normalnoj formi može biti preneks normalna forma za više formula. Transformisanje formule u preneks normalnu formu može biti realizovano procedurom prikazanom na slici 9.4 (kao i ranije, kada govorimo o „primeni neke logičke ekvalencije“ mislimo na korišćenje logičkih ekvalencija na osnovu teoreme o zameni (9.2, o efektivnoj primeni govor 9.31)). Radi jednostavnosti procedure i rezultujuće formule, iako to nije neophodno, najpre se eliminišu veznici  $\Leftrightarrow$  i  $\Rightarrow$ .

Procedura na ulazu očekuje zatvorenu formulu, formulu bez slobodnih promenljivih. To nije bitno ograničenje, jer se procedura obično koristi u okviru šireg postupka ispitivanja zadovoljivosti, a tada se, u ranijim fazama, za svaku promenljivu može odrediti način na koji je kvantifikovana.

**Primer 9.22.** Razmotrimo formulu

$$\forall x p(x) \wedge \forall x \exists y \forall z (q(y, z) \Rightarrow r(g(x), y)) .$$

Primenom algoritma PRENEX najpre se (u okviru koraka 1) dobija formula

$$\forall x p(x) \wedge \forall x \exists y \forall z (\neg q(y, z) \vee r(g(x), y)) .$$

Nakon koraka

$$\forall x(p(x) \wedge \forall x \exists y \forall z (\neg q(y, z) \vee r(g(x), y))) ,$$

kako je promenljiva  $x$  slobodna u  $p(x)$ , najpre ćemo preimenovati vezanu promenljivu  $x$  u u (u okviru formule  $\forall x \exists y \forall z (\neg q(y, z) \vee r(g(x), y))$ ):

$$\forall x(p(x) \wedge \forall u \exists y \forall z (\neg q(y, z) \vee r(g(u), y))) .$$

Nakon toga kvantifikatori  $\forall u, \exists y, \forall z$  mogu, jedan po jedan, biti pomereni na početak formule:

$$\forall x \forall u \exists y \forall z (p(x) \wedge (\neg q(y, z) \vee r(g(u), y))) .$$

Korektnost navedenog algoritma može se dokazati slično kao korektnost procedure za transformisanje formule u konjunktivnu normalnu formu (teorema 8.5).

**Teorema 9.3** (Korektnost algoritma PRENEX). *Algoritam PRENEX se zaustavlja i zadovoljava sledeće svojstvo: ako je  $\mathcal{A}$  ulazna formula, onda je izlazna formula  $\mathcal{A}'$  u preneks normalnoj formi i važi  $\mathcal{A} \equiv \mathcal{A}'$ .*

**Algoritam:** PRENEX

**Ulaz:** Zatvorena formula logike prvog reda

**Izlaz:** Preneks normalna forma zadate formule

1: **dok god** je to moguće **radi**

2: primeni neku od logičkih ekvivalentacija:

$$\begin{aligned}\mathcal{A} \Leftrightarrow \mathcal{B} &\equiv (\mathcal{A} \Rightarrow \mathcal{B}) \wedge (\mathcal{B} \Rightarrow \mathcal{A}), \\ \mathcal{A} \Rightarrow \mathcal{B} &\equiv \neg \mathcal{A} \vee \mathcal{B};\end{aligned}$$

3: **dok god** je to moguće **radi**

4: primeni neku od logičkih ekvivalentacija:

$$\begin{aligned}\neg(\mathcal{A} \wedge \mathcal{B}) &\equiv \neg \mathcal{A} \vee \neg \mathcal{B}, \\ \neg(\mathcal{A} \vee \mathcal{B}) &\equiv \neg \mathcal{A} \wedge \neg \mathcal{B}, \\ \neg(\forall x)\mathcal{A} &\equiv (\exists x)\neg \mathcal{A}, \\ \neg(\exists x)\mathcal{A} &\equiv (\forall x)\neg \mathcal{A};\end{aligned}$$

5: **dok god** je to moguće **radi**

6: primeni neku od logičkih ekvivalentacija (eliminiši višestruke veznike koristeći zakon dvojne negacije):  
 $\neg\neg \mathcal{A} \equiv \mathcal{A}$ ;

7: **dok god** je to moguće **radi**

8: primeni neku od logičkih ekvivalentacija:

$$\begin{aligned}(\forall x)\mathcal{A} \wedge \mathcal{B} &\equiv (\forall x)(\mathcal{A} \wedge \mathcal{B}), \\ (\forall x)\mathcal{A} \vee \mathcal{B} &\equiv (\forall x)(\mathcal{A} \vee \mathcal{B}), \\ \mathcal{B} \wedge (\forall x)\mathcal{A} &\equiv (\forall x)(\mathcal{B} \wedge \mathcal{A}) \\ \mathcal{B} \vee (\forall x)\mathcal{A} &\equiv (\forall x)(\mathcal{B} \vee \mathcal{A}), \\ (\exists x)\mathcal{A} \wedge \mathcal{B} &\equiv (\exists x)(\mathcal{A} \wedge \mathcal{B}), \\ (\exists x)\mathcal{A} \vee \mathcal{B} &\equiv (\exists x)(\mathcal{A} \vee \mathcal{B}), \\ \mathcal{B} \wedge (\exists x)\mathcal{A} &\equiv (\exists x)(\mathcal{B} \wedge \mathcal{A}), \\ \mathcal{B} \vee (\exists x)\mathcal{A} &\equiv (\exists x)(\mathcal{B} \vee \mathcal{A}),\end{aligned}$$

pri čemu  $x$  nema slobodna pojavljivanja u formuli  $\mathcal{B}$ ; ako  $x$  ima slobodna pojavljivanja u  $\mathcal{B}$ , onda treba najpre preimenovati promenljivu  $x$  u formuli  $(\forall x)\mathcal{A}$  (odnosno u formuli  $(\exists x)\mathcal{A}$ ).

Slika 9.4: Algoritam PRENEX.

U nekim situacijama moguće je primeniti neki korak navedenog algoritma na više od jednog načina. Na primer, formulu  $(\forall x)p(x) \wedge (\exists y)q(y)$  moguće je transformisati i u  $(\forall x)(p(x) \wedge (\exists y)q(y))$  i u  $(\exists y)((\forall x)p(x) \wedge q(y))$ . Obe ove formule su, naravno, logički ekvivalentne sa polaznom formulom. Ipak, u situacijama kada postoji izbor, uvek ćemo radije egzistencijalni kvantifikator učiniti spoljnim u odnosu na univerzalni nego obratno. Takav prioritet uvodimo zarad jednostavnijeg koraka skolemizacije (o kojem će biti reči u nastavku).

**Definicija 9.12** (Konjunktivna normalna forma). *Formula logike prvog reda koja ne sadrži kvantifikatore je u konjunktivnoj normalnoj formi ako je oblika*

$$\mathcal{A}_1 \wedge \mathcal{A}_2 \wedge \dots \wedge \mathcal{A}_n$$

pri čemu je svaka od formula  $\mathcal{A}_i$  ( $1 \leq i \leq n$ ) disjunkcija literalata.

Konjunktivna normalna forma formule predikatske logike (bez kvantifikatora) može se dobiti na isti način kao u slučaju iskazne logike (videti poglavlje 8.3.2). Dobijena formula logički je ekvivalentna polaznoj formuli.

**Primer 9.23.** Konjunktivna normalna forma formule

$$p(x) \wedge (q(y, z) \Rightarrow r(g(u), y))$$

je formula

$$p(x) \wedge (\neg q(y, z) \vee r(g(u), y)).$$

Nakon primene algoritama PRENEX i KNF, formula je u obliku  $Q_1x_1Q_2x_2 \dots Q_nx_n\mathcal{A}$ , pri čemu je  $Q_i$  ili  $\forall$  ili  $\exists$  i formula  $\mathcal{A}$  je u konjunktivnoj normalnoj formi. Za razmatranje zadovoljivosti ovakve formule bilo bi još pogodnije ako bi svi kvantifikatori  $Q_i$  bili univerzalni. Međutim, ne može se proizvoljna formula (na primer, formula  $(\exists x)p(x)$ ) transformisati u formulu takvog oblika a da joj je, pritom, logički ekvivalentna. No, moguće je nešto drugo: proizvoljna formula može se transformisati u formulu oblika  $\forall x_1\forall x_2 \dots \forall x_n\mathcal{A}$  koja je zadovoljiva ako i samo ako je zadovoljiva polazna formula (tada kažemo da su te dve formule *slabo ekvivalentne*). Tu transformaciju, transformaciju koja „eliminiše“ egzistencijalne kvantifikatore, zovemo *skolemizacija* (po matematičaru Skolemu koji ih je prvi koristio). Ona se zasniva na proširivanju polazne signature novim funkcijskim simbolima. Te dodatne funkcijске simbole zovemo *Skolemovim konstantama* (za funkcijске simbole arnosti 0) i *Skolemovim funkcijama*.

Prepostavimo da rečenica počinje egzistencijalnim kvantifikatorom:  $\exists y\mathcal{A}$ . Treba izabrati novi simbol konstante  $d$  koji se ne pojavljuje u signaturi, obrisati kvantifikator i zameniti promenljivu  $y$  simbolom  $d$ . Na taj način formula  $\exists y\mathcal{A}$  transformiše se u formulu  $\mathcal{A}[y \mapsto d]$ . Može se dokazati da je formula  $\exists y\mathcal{A}$  zadovoljiva ako i samo ako je formula  $\mathcal{A}[y \mapsto d]$  zadovoljiva.

Ako rečenica počinje nizom od  $n$  univerzalnih kvantifikatora:  $\forall x_1\forall x_2 \dots \forall x_n\exists y\mathcal{A}$ , onda uvodimo novi funkcijski simbol  $f$  arnosti  $n$  koji do tada nije postojao u signaturi. Polazna formula biće onda transformisana u formulu  $\forall x_1\forall x_2 \dots \forall x_n\mathcal{A}[y \mapsto f(x_1, x_2, \dots, x_n)]$ . Može se dokazati da je formula  $\forall x_1\forall x_2 \dots \forall x_n\exists y\mathcal{A}$  zadovoljiva ako i samo ako je formula  $\forall x_1\forall x_2 \dots \forall x_n\mathcal{A}[y \mapsto f(x_1, x_2, \dots, x_n)]$  zadovoljiva. (Primetimo da je uvođenje nove konstante samo specijalni slučaj uvođenja novog funkcijskog simbola.)

**Teorema 9.4** (Teorema o skolemizaciji). *Ako je formula  $\mathcal{B}$  nad signaturom  $\mathcal{L}'$  dobijena skolemizacijom od rečenice  $\mathcal{A}$  nad signaturom  $\mathcal{L}$  koja je u preneks normalnoj formi, onda je  $\mathcal{A}$  zadovoljiva ako i samo ako je  $\mathcal{B}$  zadovoljiva.*

**Primer 9.24.** *Skolemizacijom se formula*

$$\forall x\forall u\exists y\forall z (p(x) \wedge (\neg q(y, z) \vee r(g(u), y)))$$

*transformiše u formulu*

$$\forall x\forall u\forall z (p(x) \wedge (\neg q(h(x, u), z) \vee r(g(u), h(x, u)))) .$$

Transformisanje formule u preneks normalnu formu, pa transformisanje dela formule bez kvantifikatora u konjunktivnu normalnu formu, pa eliminisanje skolemizacijom egzistencijalnih kvantifikatora, jednog po jednog, dovodi do specifične sintaksičke forme koju zovemo *klauzalna forma*:

**Definicija 9.13** (Klauzalna forma). *Formula je u klauzalnoj formi ako je oblika*

$$\forall x_1\forall x_2 \dots \forall x_n\mathcal{A}$$

*gde je  $\mathcal{A}$  formula bez kvantifikatora koja je u konjunktivnoj normalnoj formi i  $\mathcal{A}$  nema slobodnih promenljivih osim, eventualno, promenljivih  $x_1, x_2, \dots, x_n$ .*

Ako je formula  $\forall x_1\forall x_2 \dots \forall x_n\mathcal{A}$  u klauzalnoj formi, onda se često u zapisu izostavljuju kvantifikatori i piše samo  $\mathcal{A}$ , podrazumevajući da se misli na univerzalno zatvorene formule  $\mathcal{A}$ .

**Teorema 9.5.** *Neka je formula  $\mathcal{B}$  (u klauzalnoj formi) dobijena od rečenice  $\mathcal{A}$  uzastopnom primenom sledećih postupaka:*

- transformisanje formule u preneks normalnu formu;
- transformisanje dela formule bez kvantifikatora u konjunktivnu normalnu formu;
- skolemizacija.

*Tada je formula  $\mathcal{A}$  zadovoljiva ako i samo ako je  $\mathcal{B}$  zadovoljiva.*

Navedena teorema kaže da transformisanje formule u klauzalnu formu ne daje njoj logički ekvivalentnu formulu, već formulu koja joj je samo slabo ekvivalentna. Međutim, to je dovoljno u kontekstu ispitivanja

zadovoljivosti: ako se ispituje zadovoljivost rečenice  $\mathcal{A}$ , dovoljno je ispitati zadovoljivost formule  $\mathcal{B}$  koja je u klauzalnoj formi i slabo ekvivalentna sa formulom  $\mathcal{A}$ . To daje i put za dokazivanje da je neka formula  $\mathcal{A}$  valjana: dovoljno je dokazati da je formula  $\neg\mathcal{A}$  nezadovoljiva, pa je dovoljno i dokazati da je klauzalna forma formule  $\neg\mathcal{A}$  nezadovoljiva.

**Primer 9.25.** Formula  $\mathcal{A} = (\forall x)p(x, x) \Rightarrow (\forall y)p(y, y)$  nad signaturom  $\mathcal{L}$  je valjana. To se može dokazati na sledeći način.

Formula  $\neg\mathcal{A}$  je jednaka  $\neg((\forall x)p(x, x) \Rightarrow (\forall y)p(y, y))$  i njena preneks normalna forma je  $(\exists y)(\forall x)(p(x, x) \wedge \neg p(y, y))$ . Skolemizacijom dobijamo formulu  $p(x, x) \wedge \neg p(c, c)$ , gde je  $c$  novi simbol konstante. Neka je  $\mathcal{L}'$  signatura dobijena proširivanjem signature  $\mathcal{L}$  simbolom  $c$ . Može se pokazati da je formula  $p(x, x) \wedge \neg p(c, c)$  nezadovoljiva – ili neposredno koristeći definiciju zadovoljivosti ili koristeći metod rezolucije opisan u nastavku. Kako je formula  $p(x, x) \wedge \neg p(c, c)$  nezadovoljiva, polazna formula  $\mathcal{A}$  je valjana.

### 9.3.3 Unifikacija

Problem unifikacije je problem ispitivanja da li postoji supstitucija koja čini dva izraza (dva terma ili dve formule) jednakim.

**Definicija 9.14** (Unifikabilnost i unifikator). Ako za izraze  $e_1$  i  $e_2$  postoji supstitucija  $\sigma$  takva da važi  $e_1\sigma = e_2\sigma$ , onda kažemo da su izrazi  $e_1$  i  $e_2$  unifikabilni i da je supstitucija  $\sigma$  unifikator (eng. unifier) za ta dva izraza.

**Primer 9.26.** Neka je term  $t_1$  jednak  $g(x, z)$ , neka je term  $t_2$  jednak  $g(y, f(y))$  i neka je  $\sigma$  supstitucija  $[y \mapsto x, z \mapsto f(x)]$ . Tada je i  $t_1\sigma$  i  $t_2\sigma$  jednako  $g(x, f(x))$ , pa su termovi  $t_1$  i  $t_2$  unifikabilni, a  $\sigma$  je (jedan) njihov unifikator. Unifikator termova  $t_1$  i  $t_2$  je na primer, i  $[x \mapsto a, y \mapsto a, z \mapsto f(a)]$ . Termovi  $g(x, x)$  i  $g(y, f(y))$  nisu unifikabilni.

Dva unifikabilna izraza mogu da imaju više unifikatora. Za dva unifikatora  $\sigma_1$  i  $\sigma_2$  kažemo da su jednakia do na preimenovanje promenljivih ako postoji supstitucija  $\lambda$  koja je oblika  $[v'_1 \mapsto v''_1, v'_2 \mapsto v''_2, \dots, v'_n \mapsto v''_n]$ , pri čemu su  $v'_i$  i  $v''_i$  simboli promenljivih i važi  $\sigma_1\lambda = \sigma_2$ .

**Primer 9.27.** Naredni parovi unifikatora izraza  $g(x, z)$  i  $g(y, f(y))$  jednakci su do na preimenovanje promenljivih:

- $\sigma_1 = [y \mapsto x, z \mapsto f(x)]$ ,  $\sigma_2 = [y \mapsto x, z \mapsto f(x)]$ .
- $\sigma_1 = [y \mapsto x, z \mapsto f(x)]$ ,  $\sigma_2 = [x \mapsto y, z \mapsto f(y)]$ .
- $\sigma_1 = [y \mapsto x, z \mapsto f(x)]$ ,  $\sigma_2 = [x \mapsto u, y \mapsto u, z \mapsto f(u)]$ .

**Definicija 9.15** (Najopštiji unifikator). Supstitucija  $\sigma$  je najopštiji unifikator (eng. most general unifier) za izraze  $e_1$  i  $e_2$  ako svaki unifikator  $\tau$  izraza  $e_1$  i  $e_2$  može biti predstavljen kao kompozicija supstitucije  $\sigma$  i neke supstitucije  $\mu$ .

Svaka dva unifikabilna izraza imaju najopštiji unifikator, jedinstveno do na preimenovanje promenljivih.

Na slici 9.5 dat je opis opšteg algoritma za određivanje najopštijeg unifikatora za niz parova izraza. Algoritam unifikacije ili vraća traženu supstituciju ili se zaustavlja sa neuspocom, ukazujući na to da tražena supstitucija ne postoji.

Primetimo da je korak 6 algoritma (*application*) (kao i neke druge korake algoritma) moguće u opštem slučaju primeniti na više načina. Bilo koji od tih načina vodi istom rezultatu — neuspelu (ako ne postoji traženi unifikator) ili jednom od unifikatora koji se mogu razlikovati samo do na preimenovanje promenljivih.

**Primer 9.28.** Ilustrujmo rad algoritma Najopštiji unifikator na primeru sledeća dva para termova:  
 $(g(y), x)$ ,  $(f(x, h(x), y), f(g(z), w, z))$

**Algoritam:** Najopštiji unifikator

**Ulaz:** Niz parova izraza  $(s_1, t_1), (s_2, t_2), \dots, (s_n, t_n)$

**Izlaz:** Najopštiji unifikator (ako on postoji)  $\sigma$  takav da važi  $s_1\sigma = t_1\sigma, s_2\sigma = t_2\sigma, \dots, s_n\sigma = t_n\sigma$ .

- 1: **dok god** je moguće primeniti neko od navedenih pravila **radi**
- 2: {Korak 1 (*factoring*):}
- 3: **ako** postoji par koji ima više od jednog pojavljivanja **onda**
- 4: obriši sva njegova pojavljivanja osim jednog.
- 5: {Korak 2 (*tautology*):}
- 6: **ako** postoji par  $(t, t)$  **onda**
- 7: obriši ga.
- 8: {Korak 3 (*orientation*):}
- 9: **ako** postoji par  $(t, x)$ , gde je  $x$  promenljiva, a  $t$  nije promenljiva **onda**
- 10: zameni par  $(t, x)$  parom  $(x, t)$ .
- 11: **ako** postoji par  $(s, t)$ , gde ni  $s$  ni  $t$  nisu promenljive **onda**
- 12: **ako** je  $s$  jednako  $\varphi(u_1, u_2, \dots, u_k)$  i  $t$  je jednako  $\varphi(v_1, v_2, \dots, v_k)$  (gde je  $\varphi$  funkcijski ili predikatski simbol) **onda**
- 13: {Korak 4(a) (*decomposition*):}
- 14: dodaj parove  $(u_1, v_1), (u_2, v_2), \dots, (u_k, v_k)$  i obriši par  $(s, t)$ ;
- 15: **inace**
- 16: {Korak 4(b) (*collision*):}
- 17: zaustavi rad i kao rezultat vrati *neuspeh*;
- 18: {Korak 5 (*cycle*): }
- 19: **ako** postoji par  $(x, t)$  takav da je  $x$  promenljiva i  $t$  term koji sadrži  $x$  **onda**
- 20: zaustavi rad i kao rezultat vrati *neuspeh*;
- 21: {Korak 6 (*application*): }
- 22: **ako** postoji  $(x, t)$ , gde je  $x$  promenljiva a  $t$  term koji ne sadrži  $x$  i  $x$  se pojavljuje i u nekim drugim parovima **onda**
- 23: primeni supstituciju  $[x \mapsto t]$  na sve druge parove.
- 24: vrati tekući skup parova kao najopštiji unifikator.

Slika 9.5: Algoritam Najopštiji unifikator.

Primenom koraka 3 (orientation) dobijamo

$(x, g(y)), (f(x, h(x), y), f(g(z), w, z))$ .

Primenom koraka 4(a) (decomposition) dobijamo

$(x, g(y)), (x, g(z)), (h(x), w), (y, z)$ .

Korak 6 (application) moguće je primeniti na više načina. Primenom za par  $(y, z)$  dobijamo  $(x, g(z)), (x, g(z)), (h(x), w), (y, z)$ .

Primenom koraka 1 (factoring) dobijamo

$(x, g(z)), (h(x), w), (y, z)$ .

Primenom koraka 3 (orientation) dobijamo

$(x, g(z)), (w, h(x)), (y, z)$ .

Primenom koraka 6 (application) dobijamo

$(x, g(z)), (w, h(g(z))), (y, z)$ .

Ovaj niz parova određuje traženi najopštiji unifikator  $\sigma = [x \mapsto g(z), w \mapsto h(g(z)), y \mapsto z]$ .

**Primer 9.29.** Razmotrimo sledeći par izraza  $(g(x, x), g(y, f(y)))$ .

Primenom koraka 4(a) (decomposition) dobijamo dva para izraza:  
 $(x, y), (x, f(y))$ .

Korak 6 (application) može se primeniti na samo dva načina:

- primenom za par  $(x, y)$ ; tada se dobija  $(x, y), (y, f(y))$ , odakle se, primenom koraka 5 (cycle) dolazi do neuspeha.
- primenom za par  $(x, f(y))$ ; tada se dobija  $(f(y), y), (x, f(y))$ , odakle se, primenom koraka 3 (orientation) i koraka 5 (cycle) dolazi do neuspeha.

Dakle, izrazi  $g(x, x)$  i  $g(y, f(y))$  nisu unifikabilni.

**Teorema 9.6** (Korektnost algoritma Najopštiji unifikator). *Algoritam Najopštiji unifikator zadovoljava sledeće uslove:*

- zaustavlja se;
- ako vrati supstituciju, onda je ona najopštiji unifikator za dati niz parova izraza;
- ako se algoritam zaustavi sa neuspehom, onda ne postoji unifikator za dati niz parova izraza.

Navedeni algoritam za unifikaciju nije efikasan. Postoje znatno efikasniji algoritmi i mnogi od njih zasnovani su na korišćenju pogodnih struktura podataka i implicitnom primenjivanju supstitucije (iz koraka 6). Neki od tih algoritama imaju linearnu složenost (po broju polaznih parova) ali, u opštem slučaju, najopštiji unifikator može imati eksponencijalnu dužinu (po broju polaznih parova), te ga nije moguće eksplisitno predstaviti u linearном vremenu. To ilustruje sledeći primer.

**Primer 9.30.** Za skup parova

$$\begin{aligned} & (x_1, f(x_0, x_0)) \\ & (x_2, f(x_1, x_1)) \\ & \dots \\ & (x_n, f(x_{n-1}, x_{n-1})) \end{aligned}$$

Najopštiji unifikator sadrži zamenu  $x_n \mapsto t$ , gde je  $t$  term koji sadrži samo simbole  $x_0$  i  $f$ , pri čemu ima  $2^n - 1$  pojavljivanja simbola  $f$ .

Unifikacija ima mnoge primene. Neke od njih su u transformisanju izraza korišćenjem raspoloživih pravila, a neke u metodu rezolucije.

**Primer 9.31.** U primeru 8.17, obrazloženo je zašto se u C programu uslov

`if (!(!a && !b) || c)`

može zameniti sledećim redom:

`if (a || b || c)`

Kompilatori vrše takve izmene tokom prevodenja, na način pojednostavljenog opisan u nastavku. Kompilator može da ima raspoloživa pravila poput

`!(!x && !y) —> x || y.`

Kompilator razmatra izraz  $!(!a \&\& !b) \mid\mid c$  i njegove podizraze i ispituje da li na njih može da primeni neko od raspoloživih pravila. Pravilo je primenljivo ako se njegova leva strana može unifikovati sa izrazom koji se razmatra. Pri tome, promenljive koje figurišu u izrazu iz programa smatraju se konstantama (zato se ova vrsta unifikacije zove jednosmerna unifikacija). U našem primeru,  $!(!x \&\& !y)$  može da se unifikuje sa podizrazom  $!(!a \&\& !b)$  za unifikator  $\sigma = [x \mapsto a, y \mapsto b]$ . Važi  $(x \mid\mid y)\sigma = a \mid\mid b$ , te se primenom zamene od izraza `if (!(!a && !b) || c)` dobija izraz `if (a || b || c)`.

Ovim je, zapravo, opisan i mehanizam primene logičkih ekvivalencija u algoritmima PRENEX i KNF.

### 9.3.4 Metod rezolucije

Metod rezolucije (eng. *resolution method*) formulisao je Alen Robinson (Alan Robinson) 1965. godine, sledeći mnogobrojne prethodne rezultate. To je postupak za ispitivanje (ne)zadovoljivosti formule logike prvog reda u

klauzalnoj formi, tj. za ispitivanje (ne)zadovoljivosti skupa klauza logike prvog reda. Metod se može pojednostaviti tako da je primenljiv za ispitivanje (ne)zadovoljivosti skupa klauza iskazne logike.

Formula koja je u konjunktivnoj normalnoj formi može da ima konjunkte koji se ponavljaju, a njeni konjunkti mogu da imaju literale koji se ponavljaju. Međutim, na osnovu asocijativnosti i komutativnosti konjunkcije i disjunkcije, kao i na osnovu logičkih ekvivalencija  $A \wedge A \equiv A$  i  $A \vee A \equiv A$ , takva ponavljanja mogu da se eliminisu i formula koja je u konjunktivnoj normalnoj formi može da se zameni (logički ekvivalentnom) formulom koja je konjunkcija različitih klauza od kojih je svaka disjunkcija različitih literalova. Dakle, formula se može opisati skupom klauza i, dalje, skupom skupova literalova. Takva formula je zadovoljiva ako i samo ako postoji interpretacija u kojoj su sve njene klauze tačne. Klauza je zadovoljiva ako postoji interpretacija u kojoj je bar jedan literal iz te klauze tačan, pa se smatra da prazna klauza, u oznaci  $\square$ , nije zadovoljiva.

Može se osigurati da se logičke konstante  $T$  i  $\perp$  ne pojavljuju u skupu klauza. Zaista, klauza koja sadrži literal  $T$  je u svakoj valuaciji tačna, pa može biti eliminisana (jer ne utiče na zadovoljivost polaznog skupa klauza). Ako klauza  $C$  sadrži literal  $\perp$ , onda taj literal može biti obrisan, dajući novu klauzu  $C'$  (jer je u svakoj interpretaciji klauza  $C$  tačna ako i samo ako je tačna klauza  $C'$ ).

U slučaju iskazne logike, ako je literal  $l$  jednak iskaznom slovu  $p$ , onda sa  $\bar{l}$  označavamo literal  $\neg p$ ; ako je literal  $l$  jednak negaciji iskaznog slova  $p$  (tj. literalu  $\neg p$ ), onda sa  $\bar{l}$  označavamo literal  $p$ . Za literale  $l$  i  $\bar{l}$  kažemo da su međusobno *komplementni*. U slučaju logike prvog reda, ako je literal  $l$  jednak  $p(t_1, t_2, \dots, t_n)$ , onda sa  $\bar{l}$  označavamo literal  $\neg p(t_1, t_2, \dots, t_n)$ , a ako je literal  $l$  jednak  $\neg p(t_1, t_2, \dots, t_n)$ , onda sa  $\bar{l}$  označavamo literal  $p(t_1, t_2, \dots, t_n)$ . Za literale  $l$  i  $\bar{l}$  kažemo da su (međusobno) *komplementni*.

Metod rezolucije (i za iskaznu i za logiku prvog reda) proverava da li je dati skup klauza (ne)zadovoljiv. Ako je potrebno ispitati da li je formula  $\Phi$  valjana, dovoljno je metodom rezolucije utvrditi da li je formula  $\neg\Phi$  nezadovoljiva, pri čemu je potrebno najpre formulu  $\neg\Phi$  transformisati u skup klauza. Ako se izvede prazna klauza, onda to znači da je formula  $\neg\Phi$  nezadovoljiva, pa je  $\Phi$  valjana; ako u nekom koraku ne može da se izvede nijedna nova klauza, onda to znači da je formula  $\neg\Phi$  zadovoljiva, pa  $\Phi$  nije valjana. Za razliku od iskaznog slučaja, u logici prvog reda moguće je i ishod da nove klauze mogu da se izvode beskonačno, a da se pri tome ne izvede prazna klauza. Ovaj vid dokazivanja da je formula  $\Phi$  valjana zovemo *dokazivanje pobijanjem*.

**Metod rezolucije za iskaznu logiku.** U metodu rezolucije za iskaznu logiku primenjuje se pravilo rezolucije sledećeg oblika:

$$\frac{C' \vee l \quad C'' \vee \bar{l}}{C' \vee C''}$$

Klauzu  $C' \vee C''$  zovemo *rezolventom* klauza  $C' \vee l$  i  $C'' \vee \bar{l}$ , a klauze  $C' \vee l$  i  $C'' \vee \bar{l}$  *roditeljima rezolvente*. Kažemo da klauze  $C' \vee l$  i  $C'' \vee \bar{l}$  *rezolviramo* pravilom rezolucije.

Pravilo rezolucije može se zapisati i na sledeći, intuitivniji način:

$$\frac{\neg C' \Rightarrow l \quad l \Rightarrow C''}{\neg C' \Rightarrow C''}$$

Ilustrujmo taj oblik pravila jednostavnim primerom nad formulama iskazanim prirodnim jezikom:

$$\frac{\begin{array}{c} \text{„vedro je“} \Rightarrow \text{„pada kiša“} \quad \text{„pada kiša“} \Rightarrow \text{„meč se otkazuje“} \\ \hline \text{„vedro je“} \Rightarrow \text{„meč se otkazuje“} \end{array}}{\quad}$$

U polaznoj formi pravila rezolucije, navedeno zaključivanje izgleda ovako:

$$\frac{\begin{array}{c} \text{„vedro je“} \vee \text{„pada kiša“} \quad \neg \text{„pada kiša“} \vee \text{„meč se otkazuje“} \\ \hline \text{„vedro je“} \vee \text{„meč se otkazuje“} \end{array}}{\quad}$$

*Metod rezolucije* je postupak za ispitivanje zadovoljivosti skupa klauza koji se sastoji od uzastopnog primeњivanja pravila rezolucije (slika 9.6).

U primeni metoda rezolucije, niz klauza (polaznih i izvedenih) označavaćemo obično sa  $C_1, C_2, C_3, \dots$ . Iza izvedene klauze zapisivaćemo oznake klauza iz kojih je ona izvedena, kao i redne brojeve literalova nad kojim je primenjeno pravilo rezolucije. Literale u klauzama razdvajaćemo obično simbolom ',' (umesto simbolom ' $\vee$ ').

**Primer 9.32.** Metodom rezolucije se iz skupa  $\{\{\neg p, \neg q, r\}, \{\neg p, q\}, \{p\}, \{\neg r\}\}$  može izvesti prazna klauza:

**Algoritam:** Metod rezolucije

**Ulaz:** Skup kluza  $S$

**Izlaz:** Odgovor zadovoljiv/nezadovoljiv

- 1: **ponavljam**
- 2:    **ako** u tekućem skupu kluza postoji prazna kluza ( $\square$ ) **onda**
- 3:        vratи odgovor da je skup kluza  $S$  nezadovoljiv;
- 4:    **ako** se pravilom rezolucije može izvesti neka nova kluza **onda**
- 5:        primeni pravilo rezolucije na taj način (pri tome, roditelji rezolvente se ne zamenjuju rezolventom, već se rezolventa dodaje u tekući skup kluza);
- 6:    **inače**
- 7:        vratи odgovor da je skup kluza  $S$  zadovoljiv.

Slika 9.6: Algoritam Metod rezolucije.

$$\begin{array}{ll}
 C_1 : & \neg p, \neg q, r \\
 C_2 : & \neg p, q \\
 C_3 : & p \\
 C_4 : & \neg r \\
 \hline
 C_5 : & \neg p, r \quad (C_1, 2; C_2, 2) \\
 C_6 : & \neg p \quad (C_4, 1; C_5, 2) \\
 C_7 : & \square \quad (C_3, 1; C_6, 1)
 \end{array}$$

Skup kluza  $\{\{\neg p, \neg q, r\}, \{\neg p, q\}, \{p\}, \{\neg r\}\}$  je, dakle, nezadovoljiv.

**Primer 9.33.** Metodom rezolucije se iz skupa  $\{\{\neg p, \neg q, r\}, \{\neg p, q\}, \{p\}\}$  ne može izvesti prazna kluza. Ovaj skup kluza je, dakle, zadovoljiv.

U iskaznom slučaju, nad konačnim skupom promenljivih koje se pojavljuju u zadatom skupu kluza postoji konačno mnogo kluza, pa samim tim i mogućih rezolventi tokom primene metoda rezolucije. Zbog toga se metod rezolucije zaustavlja za svaku ulaznu formulu (tj. za svaki ulazni skup kluza). Može se dokazati i više, da je metod rezolucije *procedura odlučivanja* za zadovoljivost skupova kluza iskazne logike:

**Teorema 9.7** (Teorema o algoritmu Metod rezolucije). *Metod rezolucije zaustavlja se za svaku iskaznu formulu i u završnom skupu kluza postoji prazna kluza ako i samo ako je polazna formula nezadovoljiva.*

Metod rezolucije može na razne načine biti modifikovan tako da bude efikasniji.

**Metod rezolucije za logiku prvog reda.** U iskaznoj logici pravilo rezolucije može da se primeni na sledeće načine:

$$\frac{\text{,,Sokrat je filozof"} \Rightarrow \text{,,Sokrat je čovek"} \quad \text{,,Sokrat je čovek"} \Rightarrow \text{,,Sokrat je smrtan"}}{\text{,,Sokrat je filozof"} \Rightarrow \text{,,Sokrat je smrtan"}}$$

$$\frac{\text{,,Platon je filozof"} \Rightarrow \text{,,Platon je čovek"} \quad \text{,,Platon je čovek"} \Rightarrow \text{,,Platon je smrtan"} }{\text{,,Platon je filozof"} \Rightarrow \text{,,Platon je smrtan"}}$$

U logici prvog reda pravilo rezolucije (koje će tek biti uvedeno precizno) može da se primeni na sledeći način:

$$\frac{\text{,,}x \text{ je filozof"} \Rightarrow \text{,,}x \text{ je čovek"} \quad \text{,,}x \text{ je čovek"} \Rightarrow \text{,,}x \text{ je smrtan"} }{\text{,,}x \text{ je filozof"} \Rightarrow \text{,,}x \text{ je smrtan"}}$$

Ovakvo zaključivanje je opštije (pa u tom smislu i bolje) jer iz jednog zaključka „ $x$  je filozof“  $\Rightarrow$  „ $x$  je smrtan“ primenom supstitucija  $[x \mapsto \text{Sokrat}]$  i  $[x \mapsto \text{Platon}]$ , možemo da dobijemo specifična tvrđenja:

„Sokrat je filozof“  $\Rightarrow$  „Sokrat je smrtan“;

„Platon je filozof“  $\Rightarrow$  „Platon je smrtan“

pa, primenom supstitucije  $[x \mapsto \text{Cezar}]$  i

„Cezar je filozof“  $\Rightarrow$  „Cezar je smrtan“ (iako Cezar nije filozof) i mnoga druga slična. Ovakvo zaključivanje, koje uključuje promenljive logike prvog reda, ne može se opravdati u terminima iskazne logike i za njega je potrebno pravilo rezolucije za logiku prvog reda, motivisano prethodnim primerima. Njegov specijalni slučaj je:

$$\frac{\neg\Gamma' \Rightarrow \mathcal{A} \quad \mathcal{A} \Rightarrow \Gamma''}{\neg\Gamma' \Rightarrow \Gamma''}$$

U navedenom pravilu, veznik negacije koji postoji u formuli  $\neg\Gamma' \Rightarrow \mathcal{A}$  tu je samo radi pogodnijeg zapisa pravila do kojeg ćemo doći kasnije. I formule bez te negacije mogu da se rezolviraju analogno.

Napravimo korak dalje. U navedenom pravilu specijalnog oblika, ista formula  $\mathcal{A}$  pojavljuje se i u formuli  $\neg\Gamma' \Rightarrow \mathcal{A}$  i u formuli  $\mathcal{A} \Rightarrow \Gamma''$ . Ona može da bude instance formula  $\mathcal{A}'$  i  $\mathcal{A}''$  koje nisu jednake ali su unifikabilne, tj. formula  $\mathcal{A}$  može da bude jednaka formulama  $\mathcal{A}'\sigma$  i  $\mathcal{A}''\sigma$ . To ilustruje naredni primer.

**Primer 9.34.** Formule  $\neg p(a, y) \Rightarrow q(a, y)$  i  $q(x, b) \Rightarrow r(x, b)$  mogu da se rezolviraju tek ako se na njih primeni supstitucija  $\sigma$  takva da važi  $q(a, y)\sigma = q(x, b)\sigma$ , na primer,  $\sigma = [x \mapsto a, y \mapsto b]$ . Formule  $\neg p(a, b) \Rightarrow q(a, b)$  i  $q(a, b) \Rightarrow r(a, b)$  se mogu rezolvirati i daju zaključak  $\neg p(a, b) \Rightarrow r(a, b)$ .

Kao što je nagovešteno, i formule bez veznika negacije koji postoji u formuli  $\neg\Gamma' \Rightarrow \mathcal{A}$  mogu da se rezolviraju analogno i to ilustruje naredni primer.

**Primer 9.35.** Prepostavimo da su date formule  $otac(Aleksa, y) \Rightarrow roditelj(Aleksa, y)$  i  $roditelj(x, Branko) \Rightarrow predak(x, Branko)$ . One mogu da se rezolviraju iako formule  $roditelj(Aleksa, y)$  i  $roditelj(x, Branko)$  nisu jednake. Naime, one su unifikabilne i jedan unifikator je  $\sigma = [x \mapsto Aleksa, y \mapsto Branko]$ . Formule  $otac(Aleksa, Branko) \Rightarrow roditelj(Aleksa, Branko)$  i  $roditelj(Aleksa, Branko) \Rightarrow predak(Aleksa, Branko)$  se mogu rezolvirati i daju zaključak  $otac(Aleksa, Branko) \Rightarrow predak(Aleksa, Branko)$ . Primetimo da se na osnovu datih formula ne može zaključiti ništa opštije (na primer, ne može se izvesti formula  $otac(Aleksa, y) \Rightarrow predak(Aleksa, y)$ ).

Pokazano je kako se rezolviraju formule  $\neg\Gamma' \Rightarrow \mathcal{A}'$  i  $\mathcal{A}'' \Rightarrow \Gamma''$ , kada su formule  $\mathcal{A}'$  i  $\mathcal{A}''$  unifikabilne. Zapravo se rezolviraju formule  $(\neg\Gamma' \Rightarrow \mathcal{A}')\sigma$  i  $(\mathcal{A}'' \Rightarrow \Gamma'')\sigma$  tj. formule  $\neg\Gamma'\sigma \Rightarrow \mathcal{A}'\sigma$  i  $\mathcal{A}''\sigma \Rightarrow \Gamma''\sigma$ , pri čemu je  $\sigma$  neki unifikator za formule  $\mathcal{A}'$  i  $\mathcal{A}''$ , tj. važi  $\mathcal{A}'\sigma = \mathcal{A}''\sigma$ :

$$\frac{\neg\Gamma'\sigma \Rightarrow \mathcal{A}'\sigma \quad \mathcal{A}''\sigma \Rightarrow \Gamma''\sigma}{\neg\Gamma'\sigma \Rightarrow \Gamma''\sigma}$$

Navedeno pravilo može se zapisati i na sledeći način:

$$\frac{(\Gamma' \vee \mathcal{A}')\sigma \quad (\Gamma'' \vee \neg\mathcal{A}'')\sigma}{(\Gamma' \vee \Gamma'')\sigma}$$

ili, kao što je uobičajeno:

$$\frac{\Gamma' \vee \mathcal{A}' \quad \Gamma'' \vee \neg\mathcal{A}''}{(\Gamma' \vee \Gamma'')\sigma}$$

uz uslov da je  $\sigma$  unifikator za  $\mathcal{A}'$  i  $\mathcal{A}''$ .

**Primer 9.36.** Klauze  $\neg otac(x, y) \vee roditelj(x, y)$  i  $\neg roditelj(x, y) \vee predak(x, y)$  rezolviraju se na sledeći način:

$$\frac{\neg otac(x, y) \vee roditelj(x, y) \quad \neg roditelj(x, y) \vee predak(x, y)}{\neg otac(x, y) \vee predak(x, y)}$$

Za  $\sigma = [x \mapsto Aleksa, y \mapsto Branko]$  dobija se rezolventa  $\neg otac(Aleksa, Branko) \vee predak(Aleksa, Branko)$ . Ipak, mnogo opštiji zaključak je rezolventa  $\neg otac(x, y) \vee predak(x, y)$ , koja se dobija za unifikator  $\sigma = []$ .

Kao što pokazuje i prethodni primer, bolje je da su zaključci što opštiji, te se u pravilu zahteva da je  $\sigma$  najopštiji unifikator za  $\mathcal{A}'$  i  $\mathcal{A}''$ . Zato pravilo rezolucije za logiku prvog reda u nešto opštijem, ali i dalje specijalnom obliku (tzv. binarna rezolucija) glasi ovako:

$$\frac{\Gamma' \vee \mathcal{A}' \quad \Gamma'' \vee \neg\mathcal{A}''}{(\Gamma' \vee \Gamma'')\sigma}$$

gde su  $\Gamma'$  i  $\Gamma''$  klauze, a  $\sigma$  je najopštiji unifikator za  $\mathcal{A}'$  i  $\mathcal{A}''$ . Opšte pravilo rezolucije omogućava rezolviranje više literala odjednom. Ono može biti opisano na sledeći način:

$$\frac{\Gamma' \vee \mathcal{A}'_1 \vee \mathcal{A}'_2 \vee \dots \vee \mathcal{A}'_m \quad \Gamma'' \vee \neg \mathcal{A}''_1 \vee \neg \mathcal{A}''_2 \vee \dots \vee \neg \mathcal{A}''_n}{(\Gamma' \vee \Gamma'')\sigma}$$

gde je  $\sigma$  najopštiji unifikator za formule (sve istovremeno)  $\mathcal{A}'_1, \mathcal{A}'_2, \dots, \mathcal{A}'_m, \mathcal{A}''_1, \mathcal{A}''_2, \dots, \mathcal{A}''_n$ .

**Primer 9.37.** Primenjeno nad klauzama  $r(g(y)) \vee p(x) \vee p(f(y))$  i  $q(x) \vee \neg p(f(g(a)))$ , uz unifikator  $[y \mapsto g(a), x \mapsto f(g(a))]$ , opšte pravilo rezolucije daje rezolventu  $r(g(g(a))) \vee q(f(g(a)))$ .

Obe klauze na koje se primenjuje pravilo rezolucije su (implicitno) univerzalno kvantifikovane. Zbog toga se svaka od njihovih varijabli može preimenovati (jer su formule  $\forall x \mathcal{A}(x)$  i  $\forall x' \mathcal{A}(x')$  logički ekvivalentne). Štaviše, to je neophodno uraditi za sve deljene varijable, jer bi, inače, neke primene pravila rezolucije bile (pogrešno) onemogućene (jer odgovarajući literali ne bi bili unifikabilni).

**Primer 9.38.** Nad klauzama  $\neg p(x, y) \vee \neg p(z, y) \vee p(x, z)$  i  $\neg p(b, a)$  može se primeniti pravilo rezolucije, jer su literali  $p(x, z)$  i  $p(b, a)$  unifikabilni (uz najopštiji unifikator  $\sigma = [x \mapsto b, z \mapsto a]$ ). Rezolventa ove dve klauze je klauza

$$\neg p(b, y) \vee \neg p(a, y).$$

Ako se pravilo rezolucije primenjuje dalje, onda u dobijenoj klauzi sve promenljive treba da budu preimenovane (treba da dobiju imena koja do tada nisu korišćena):

$$\neg p(b, y') \vee \neg p(a, y').$$

Metod rezolucije za logiku prvog reda ima isti opšti oblik kao metod rezolucije za iskaznu logiku (slika 9.6), s tim što se koristi opšte pravilo rezolucije za logiku prvog reda.

**Primer 9.39.** Dokažimo da je formula  $p(a) \Rightarrow (\exists x)p(x)$  valjana. Negacija date formule je logički ekvivalentna formuli  $p(a) \wedge (\forall x)\neg p(x)$ . Metod rezolucije primenjuje se na skup klauza  $\{p(a), \neg p(x)\}$ . Pravilo rezolucije moguće je primeniti samo na jedan način – literali  $p(a)$  i  $\neg p(x)$  se unifikuju supstitucijom  $[x \mapsto a]$  i njime se dobija prazna klauza. Odatle sledi da je formula  $p(a) \Rightarrow (\exists x)p(x)$  valjana.

**Primer 9.40.** Formula  $(\forall x)(\exists y)p(x, y) \Rightarrow (\exists y)(\forall x)p(x, y)$  nije valjana. Negacija date formule je logički ekvivalentna sa formulom  $(\forall x)(\exists y)(p(x, y) \wedge (\forall y)(\exists x) \neg p(x, y))$  i sa formulom  $(\forall x)(\exists y)(\forall u)(\exists v)(p(x, y) \wedge \neg p(v, u))$ . Skolemizacijom se dobija skup od dve klauze:  $\{p(x, f(x)), \neg p(g(x, u), u)\}$ . Pravilo rezolucije nije moguće primeniti na ove dve klauze, odakle sledi da je formula  $(\forall x)(\exists y)(p(x, y) \wedge (\forall y)(\exists x) \neg p(x, y))$  zadovoljiva, tj. polazna formula nije valjana.

U primenama metoda rezolucije, niz klauza (polaznih i izvedenih) označavaćemo često sa  $C_i$  ( $i = 1, 2, \dots$ ). Iza izvedene klauze zapisivaćemo oznake klauza iz kojih je ona izvedena, redne brojeve literala u tim klauzama, iskorišćeni najopštiji unifikator, kao i supstituciju kojom se preimenuju promenljive.

**Primer 9.41.** Dokažimo da je formula

$$(\forall x)(\exists y)q(x, y)$$

logička posledica skupa formula

$$\{(\forall x)(\exists y)p(x, y), (\forall x)(\forall y)(p(x, y) \Rightarrow q(x, y))\}.$$

Dovoljno je dokazati da je formula

$$\mathcal{A} = ((\forall x)(\exists y)p(x, y) \wedge (\forall x)(\forall y)(p(x, y) \Rightarrow q(x, y))) \Rightarrow (\forall x)(\exists y)q(x, y)$$

valjana. Preneks normalna forma negacije ove formule je

$$(\exists w)(\forall x)(\exists y)(\forall u)(\forall v)(\forall z)(p(x, y) \wedge (\neg p(u, v) \vee q(u, v)) \wedge \neg q(w, z)).$$

Nakon skolemizacije, ova formula dobija oblik:

$$(\forall x)(\forall u)(\forall v)(\forall z)(p(x, g(x)) \wedge (\neg p(u, v) \vee q(u, v)) \wedge \neg q(c, z)),$$

pri čemu je  $c$  nova Skolemova konstanta, a  $g$  nova Skolemova funkcija. Konjunktivna normalna forma formule

$$p(x, g(x)) \wedge (\neg p(u, v) \vee q(u, v)) \wedge \neg q(c, z)$$

je ista ta formula, pa početni skup čine sledeće kluze:

$$C_1 : p(x, g(x)) \quad (\text{prvi deo hipoteze})$$

$$C_2 : \neg p(u, v), q(u, v) \quad (\text{drugi deo hipoteze})$$

$$C_3 : \neg q(c, z) \quad (\text{zaključak})$$

Prazna kluza izvodi se na sledeći način:

$$C_4 : q(x', g(x')) \quad (C_1, 1; C_2, 1), [v \mapsto g(x), u \mapsto x];$$

preimenovanje:  $[x \mapsto x']$

$$C_5 : \square \quad (C_3, 1; C_4, 1), [x' \mapsto c, z \mapsto g(c)]$$

**Primer 9.42.** Dokažimo da je formula

$$(\forall x)(\forall y)(\forall z)(\text{above}(y, x) \wedge \text{below}(z, x) \Rightarrow \text{above}(y, z))$$

logička posledica skupa formula

$$\{ (\forall x)(\forall y)(\text{above}(x, y) \Rightarrow \neg \text{above}(y, x)), (\forall x)(\forall y)(\text{above}(x, y) \Leftrightarrow \text{below}(y, x)), (\forall x)(\forall y)(\forall z)(\text{above}(x, y) \wedge \text{above}(y, z) \Rightarrow \text{above}(x, z)) \}$$

(videti primere 9.1 i 9.19). Dovoljno je dokazati da je formula

$$(\forall x)(\forall y)(\text{above}(x, y) \Rightarrow \neg \text{above}(y, x)) \wedge$$

$$(\forall x)(\forall y)(\text{above}(x, y) \Leftrightarrow \text{below}(y, x)) \wedge$$

$$(\forall x)(\forall y)(\forall z)(\text{above}(x, y) \wedge \text{above}(y, z) \Rightarrow \text{above}(x, z))$$

$\Rightarrow$

$$(\forall x)(\forall y)(\forall z)(\text{above}(y, x) \wedge \text{below}(z, x) \Rightarrow \text{above}(y, z))$$

valjana. Odgovarajući skup kluza je:

$$C_1 : \neg \text{above}(x_1, y_1) \vee \neg \text{above}(y_1, x_1) \quad (\text{prvi deo hipoteze})$$

$$C_2 : \neg \text{above}(x_2, y_2) \vee \text{below}(y_2, x_2) \quad (\text{drugi deo hipoteze})$$

$$C_3 : \neg \text{below}(x_3, y_3) \vee \text{above}(y_3, x_3) \quad (\text{drugi deo hipoteze})$$

$$C_4 : \neg \text{above}(x_4, y_4) \vee \neg \text{above}(y_4, z_4) \vee \text{above}(x_4, z_4) \quad (\text{treći deo hipoteze})$$

$$C_5 : \text{above}(c_y, c_x) \quad (\text{prvi deo zaključka})$$

$$C_6 : \text{below}(c_z, c_x) \quad (\text{drugi deo zaključka})$$

$$C_7 : \neg \text{above}(c_y, c_z) \quad (\text{treći deo zaključka})$$

Prazna kluza se izvodi na sledeći način:

$$C_8 : \neg \text{above}(c_y, y_5) \vee \neg \text{above}(y_5, c_z) \quad (C_7, 1; C_4, 3), [x_4 \mapsto c_y, z_4 \mapsto c_z];$$

preimenovanje:  $[y_4 \mapsto y_5]$

$$C_9 : \neg \text{above}(c_x, c_z) \quad (C_8, 1; C_5, 1), [y_5 \mapsto c_x];$$

$$C_{10} : \neg \text{below}(c_z, c_x) \quad (C_3, 2; C_9, 1), [y_3 \mapsto c_x, x_3 \mapsto c_z];$$

$$C_{11} : \square \quad (C_6, 1; C_{10}, 1), []$$

**Primer 9.43.** Formula  $\forall x \forall y (p(x, y) \Rightarrow p(y, x))$  je logička posledica formula  $\forall x p(x, x)$  i  $\forall u \forall v \forall w (p(u, v) \wedge p(w, v) \Rightarrow p(u, w))$ , jer je formula

$$\mathcal{A} = (\forall x p(x, x)) \wedge (\forall u \forall v \forall w (p(u, v) \wedge p(w, v) \Rightarrow p(u, w))) \Rightarrow$$

$$(\forall x \forall y (p(x, y) \Rightarrow p(y, x)))$$

valjana:

$C_1 : p(x, x)$	
$C_2 : \neg p(u, v), \neg p(w, v), p(u, w)$	
$C_3 : p(a, b)$	
$C_4 : \neg p(b, a)$	
$C_5 : \neg p(u', b), p(u', a)$	$(C_2, 2; C_3, 1) [w \mapsto a, v \mapsto b];$ preimenovanje: $[u \mapsto u']$
$C_6 : \neg p(b, b)$	$(C_4, 1; C_5, 2) [u' \mapsto b]$
$C_7 : \square$	$(C_1, 1; C_6, 1) [x \mapsto b]$

Kao što je ranije rečeno, da bismo dokazali da važi  $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n \models \mathcal{B}$ , dovoljno je dokazati da je formula  $\mathcal{A}_1 \wedge \mathcal{A}_2 \wedge \dots \wedge \mathcal{A}_n \Rightarrow \mathcal{B}$  valjana, tj. da je formula  $\mathcal{A}_1 \wedge \mathcal{A}_2 \wedge \dots \wedge \mathcal{A}_n \wedge \neg \mathcal{B}$  nezadovoljiva. Ako formule  $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n, \mathcal{B}$  nemaju slobodne promenljive, onda ne moramo da u klauzalnu formu transformišemo navedenu formulu, dovoljno je da u klauzalne forme transformišemo formule  $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n, \mathcal{B}$  pojedinačno. Time će se dobiti isti skup klauza, ali na jednostavniji način, primenljiv i na prethodne primere.

**Teorema 9.8** (Saglasnost i potpunost metoda rezolucije). *Metod rezolucije je sagasan: ako je primenom metoda dobijena prazna klauza, onda je i polazni skup klauza nezadovoljiv (ili, drugim rečima, iz zadovoljivog skupa klauza može se dobiti samo zadovoljiv skup klauza).*

*Metod rezolucije je potpun za pobijanje: iz svakog nezadovoljivog skupa klauza moguće je izvesti praznu klauzu.*

Navedena teorema odnosi se na opšte pravilo rezolucije. Navedena svojstva važe i za binarno pravilo ako se pored njega koristi i dodatno pravilo — faktorizacija (ili grupisanje) koje grapiše i unifikabilne literale u okviru jedne klauze.

Problem ispitivanja valjanosti u logici prvog reda nije odlučiv, ali metod rezolucije pruža najviše što se može dobiti: njime se iz svakog nezadovoljivog skupa klauza može izvesti prazna klauza (čime je dokazano da je nezadovoljiv), ali ne može se za svaki zadovoljiv skup klauza dokazati da je zadovoljiv (naime, moguće je izvesti beskonačno mnogo rezolventi). Dualno, pobijanjem se metodom rezolucije može za svaku valjanu formulu dokazati da je valjana, ali ne može se za svaku formulu koja nije valjana utvrditi da nije valjana.

Primetimo da u opisu metoda rezolucije nije zadan način na koji se biraju klauze nad kojim se primenjuje pravilo rezolucije. Takođe, teorema 9.8 tvrdi da se iz svakog nezadovoljivog skupa klauza može izvesti prazna klauza, a ne tvrdi da se iz svakog nezadovoljivog skupa klauza mora izvesti prazna klauza bez obzira na izbor klauza za rezolviranje. Naime, u zavisnosti od izbora klauza na koje se primenjuje pravilo rezolucije moguće je da se i za nezadovoljiv skup klauza metod rezolucije ne zaustavlja. Dakle, primena metoda rezolucije može se razmatrati i kao problem pretrage. Način na koji se biraju klauze na koje se primenjuje pravilo rezolucije čini strategiju za upravljanje metoda rezolucije.

Jedna od mogućnosti za obezbeđivanje potpunosti metoda rezolucije u strožijem smislu (da postoji strategija za upravljanje metoda rezolucije takva da se iz svakog nezadovoljivog skupa klauza nužno izvodi prazna klauza u konačno mnogo koraka) je sistematsko izvođenje svih rezolventi iz skupa klauza koji se širi tokom primene metoda. *Sistematski metod rezolucije* može se definisati na sledeći način: metod se primenjuje u stupnjevima; prvi stupanj čini kreiranje početnog skupa klauza; neka pre  $i$ -tog stupnja tekući skup klauza čine klauze  $C_1, C_2, \dots, C_n$ ,  $i$ -ti stupanj sastoji se od izvođenja (i dodavanja tekućem skupu klauza) svih mogućih rezolventi iz po svake dve klauze iz skupa  $C_1, C_2, \dots, C_n$  (broj tih klauza je konačan); metod se zaustavlja ako se u nekom koraku izvede prazna klauza ili ako se u nekom stupnju ne može izvesti nijedna nova klauza. Ako je razmatrani skup klauza  $\Gamma$  nezadovoljiv, onda se, na osnovu teoreme 9.8, iz njega metodom rezolucije može izvesti prazna klauza, tj. postoji niz rezolventi  $R_1, R_2, \dots, R_n$  (koje se izvode iz početnih i izvedenih klauza) od kojih je poslednja u nizu prazna klauza. Ako se na skup klauza  $\Gamma$  primeni sistematski metod rezolucije, u nekom stupnju biće (ako već pre toga nije izvedena prazna klauza) izvedene sve klauze iz skupa  $R_1, R_2, \dots, R_n$ , pa i prazna klauza. Odakle sledi naredna teorema.

**Teorema 9.9** (Potpunost sistematskog metoda rezolucije). *Ako je  $\Gamma$  nezadovoljiv skup klauza, onda se iz njega sistematskim metodom rezolucije mora izvesti prazna klauza.*

Očigledno je da je sistematski metod rezolucije izuzetno neefikasan. Postoji više strategija koje obezbeđuju nužno izvođenje prazne klauze iz nezadovoljivog skupa klauza (tj. sprečavaju beskonačne petlje), ali na efikasniji način. Smanjivanje izvođenja nepotrebnih klauza jedan je od najvažnijih ciljeva u dizajniranju strategija za metod rezolucije.

### 9.3.5 Prirodna dedukcija

Pojam valjanosti je semantičke prirode, a koncept dokazivanja i sistema za dedukciju vodi do pojma teoreme koji je sintaksički-deduktivne prirode. Pojam *teoreme* je deduktivni pandan semantičkog pojma *valjane formule*. Između ova dva pojma postoji veza i deduktivni sistemi obično imaju svojstvo potpunosti i saglasnosti: ako je neka formula valjana, onda ona može biti dokazana u okviru deduktivnog sistema, a ako za neku formulu postoji dokaz u okviru deduktivnog sistema, onda je ona sigurno valjana.

Sistemi za dedukciju su čisto sintaksičke prirode — primenjuju se kroz kombinovanje simbola, ne razmatrajući semantiku formula. Sisteme za dedukciju zovemo i *račun* — (*iskazni račun* u slučaju iskazne logike i *predikatski račun* u slučaju logike prvog reda). Postoji više različitih deduktivnih sistema, a u nastavku će biti opisan samo jedan — *prirodna dedukcija* (eng. *natural deduction*).

Sistem prirodne dedukcije (račun prirodne dedukcije) uveo je, 1935. godine, Gerhard Gencen (Gerhard Gentzen) sa namerom da prirodnije opiše uobičajeno zaključivanje matematičara.

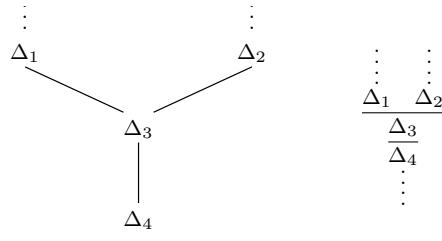
U prirodnoj dedukciji koriste se logički veznici  $\neg$ ,  $\wedge$ ,  $\vee$ ,  $\Rightarrow$ , kao i logička konstanta  $\perp$ . Formula  $\mathcal{A} \Leftrightarrow \mathcal{B}$  je kraći zapis za  $(\mathcal{A} \Rightarrow \mathcal{B}) \wedge (\mathcal{B} \Rightarrow \mathcal{A})$ , a formula  $\top$  kraći zapis za  $\mathcal{A} \Rightarrow \mathcal{A}$ . Skup formula definiše se na uobičajeni način.

Pravila izvođenja sistema prirodne dedukcije data su na slici 9.7. Primetimo da za svaki logički veznik i svaki kvantifikator postoje pravila koja ga uvode (pravila *I*-tipa) i pravila koja ga eliminisu (pravila *E*-tipa). Pravilo *efq* (*Ex falso quodlibet*) je jedino pravilo koje ne uvodi niti eliminiše neki logički veznik. Skup pravila sistema prirodne dedukcije za iskaznu logiku čine sva pravila sa slike 9.7 izuzev onih koja uključuju kvantifikatore.

$$\begin{array}{c}
 \frac{[\mathcal{A}]^u}{\perp} \neg I, u \\
 \frac{\mathcal{A} \quad \neg \mathcal{A}}{\perp} \neg E \\
 \frac{\mathcal{A} \quad \mathcal{B}}{\mathcal{A} \wedge \mathcal{B}} \wedge I \qquad \frac{\mathcal{A} \wedge \mathcal{B}}{\mathcal{A}} \wedge E \quad \frac{\mathcal{A} \wedge \mathcal{B}}{\mathcal{B}} \wedge E \\
 \frac{\mathcal{A} \quad \mathcal{B}}{\mathcal{A} \vee \mathcal{B}} \vee I \quad \frac{\mathcal{B} \quad \mathcal{A}}{\mathcal{A} \vee \mathcal{B}} \vee I \qquad \frac{\mathcal{A} \vee \mathcal{B} \quad C \quad C}{C} \vee E, u, v \\
 \frac{[\mathcal{A}]^u}{\mathcal{A} \Rightarrow \mathcal{B}} \Rightarrow I, u \qquad \frac{\mathcal{A} \quad \mathcal{A} \Rightarrow \mathcal{B}}{\mathcal{B}} \Rightarrow E \\
 \frac{\mathcal{A}[x \mapsto y] \quad (\forall x)\mathcal{A}}{(\forall x)\mathcal{A}} \forall I \qquad \frac{(\forall x)\mathcal{A}}{\mathcal{A}[x \mapsto t]} \forall E \\
 \text{uz dodatni uslov} \\
 \frac{[\mathcal{A}[x \mapsto y]]^u}{\frac{(\exists x)\mathcal{A} \quad \mathcal{B}}{\mathcal{B}} \exists E, u} \exists I \qquad \frac{(\exists x)\mathcal{A}}{\mathcal{B}} \exists E, u \\
 \text{uz dodatni uslov} \\
 \frac{\perp}{D} \text{ efq}
 \end{array}$$

Slika 9.7: Pravila izvođenja sistema prirodne dedukcije.

U pravilima izvođenja prikazanim na slici 9.7, simbol  $t$  označava proizvoljan term. Simbol  $y$  označava tzv. *pravu promenljivu, ajgenvarijablu* (eng. *eigenvariable*) — simbol promenljive za koju važi dodatni, tzv. *ajgenvarijabla* uslov. Ovaj uslov za pravilo  $\forall I$  je da važi da je  $x = y$  ili da promenljiva  $y$  nije slobodna u  $\mathcal{A}$ , kao i da važi da  $y$  nije slobodna ni u jednoj neoslobodenoj prepostavci u izvođenju formule  $\mathcal{A}[x \mapsto y]$ . Ajgenvarijabla



Slika 9.8: Deo dokaza i njegov pojednostavljeni prikaz.

uslov za pravilo  $\exists E$  je da važi da je  $x = y$  ili da promenljiva  $y$  nije slobodna u  $\mathcal{A}$ , kao i da važi da  $y$  nije slobodna u  $\mathcal{B}$  niti u bilo kojoj neoslobođenoj prepostavci u izvođenju formule  $\mathcal{B}$  osim, eventualno, u formuli  $\mathcal{A}[x \mapsto y]$ .

Postoji sistem prirodne dedukcije za klasičnu logiku, sistem NK, i sistem prirodne dedukcije za intuicionističku logiku, sistem NJ. U sistemu NK postoji jedna aksiomska shema –  $\mathcal{A} \vee \neg\mathcal{A}$  (*tertium non datur*), a sistem NJ nema aksioma.

Tokom izvođenja dokaza u sistemu prirodne dedukcije mogu se koristiti (nedokazane) prepostavke, ali one moraju biti eliminisane („oslobođene“) pre kraja izvođenja. U zapisu pravila,  $[\mathcal{F}]$  označava da se nekoliko (možda i nula) pojavljivanja prepostavke  $\mathcal{F}$  oslobada (kao nedokazane, neraspoložive prepostavke) neposredno nakon primene pravila. Pri tome, može ostati i nekoliko neoslobođenih pojavljivanja prepostavke  $\mathcal{F}$ . Prepostavkama su pridružene oznake (obično prirodni brojevi), koje se zapisuju i u okviru zapisa primjenjenog pravila (kako bi se znalo koja prepostavka je oslobođena u kojem koraku).

U sistemu prirodne dedukcije, *dokaz* je stablo čijem je svakom čvoru pridružena formula, a svakom listu ili prepostavka ili aksioma. Formula  $\mathcal{A}$  je *teorema* prirodne dedukcije ako postoji dokaz u čijem je korenju  $\mathcal{A}$  i koji nema neoslobođenih prepostavki i tada pišemo  $\vdash \mathcal{A}$  i kažemo da je formula  $\mathcal{A}$  *dokaziva* u sistemu prirodne dedukcije. Ako postoji dokaz u čijem je korenju formula  $\mathcal{A}$  i koji ima neoslobođene prepostavke koje pripadaju nekom skupu  $\Gamma$ , onda kažemo da je formula  $\mathcal{A}$  *deduktivna posledica* skupa  $\Gamma$  i tada pišemo  $\Gamma \vdash \mathcal{A}$ . Elemente skupa  $\Gamma$  tada zovemo i *premisama* ili *hipotezama* dokaza. Ako je skup  $\Gamma$  jednak  $\{\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_n\}$ , onda pišemo i  $\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_n \vdash \mathcal{A}$ .

Dokaz u sistemu prirodne dedukcije obično se prikazuje u vidu stabla čiji su listovi na vrhu, a koren na dnu. To stablo se prikazuje pojednostavljeni, stilizованo (videti sliku 9.8).

U narednim primerima prikazani su dokazi nekoliko teorema. To su teoreme i klasične i intuicionističke logike jer se u dokazima ne koriste aksiome *tertium non datur*, tj.  $\mathcal{A} \vee \neg\mathcal{A}$ . Ako se neka formula može dokazati samo uz korišćenje takvih aksioma, onda je ona teorema klasične, ali ne i intuicionističke logike.

**Primer 9.44.** Formula  $(\mathcal{A} \vee \mathcal{B}) \Rightarrow (\mathcal{B} \vee \mathcal{A})$  je teorema sistema prirodne dedukcije, tj. važi  $\vdash (\mathcal{A} \vee \mathcal{B}) \Rightarrow (\mathcal{B} \vee \mathcal{A})$ :

$$\frac{[\mathcal{A} \vee \mathcal{B}]^1 \quad \frac{[\mathcal{A}]^2}{\mathcal{B} \vee \mathcal{A}} \vee I \quad \frac{[\mathcal{B}]^3}{\mathcal{B} \vee \mathcal{A}} \vee I}{\frac{\mathcal{B} \vee \mathcal{A}}{(\mathcal{A} \vee \mathcal{B}) \Rightarrow (\mathcal{B} \vee \mathcal{A})} \vee E, 2, 3} \Rightarrow I, 1$$

**Primer 9.45.** U sistemu prirodne dedukcije važi:  $\mathcal{A} \Rightarrow \mathcal{B}, \mathcal{B} \Rightarrow \mathcal{C} \vdash \mathcal{A} \Rightarrow \mathcal{C}$ :

$$\frac{[\mathcal{A}]^1 \quad \frac{\mathcal{A} \Rightarrow \mathcal{B}}{\mathcal{B}} \Rightarrow E \quad \frac{\mathcal{B} \Rightarrow \mathcal{C}}{\mathcal{C}} \Rightarrow E}{\frac{\mathcal{C}}{\mathcal{A} \Rightarrow \mathcal{C}} \Rightarrow I, 1} \Rightarrow E$$

**Primer 9.46.** U sistemu prirodne dedukcije važi  $\vdash \mathcal{A} \Rightarrow (\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C})$ :

$$\frac{\frac{[\mathcal{A}]^1}{\mathcal{A} \vee \mathcal{B}} \vee I \quad \frac{[\mathcal{A}]^1}{\mathcal{A} \vee \mathcal{C}} \vee I}{(\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C})} \wedge I \\ \mathcal{A} \Rightarrow (\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C}) \Rightarrow I, 1$$

U prethodnom dokazu, primenom pravila  $\Rightarrow I$  nisu morala da budu oslobođena sva pojavljivanja pretpostavke  $\mathcal{A}$ . Na primer:

$$\frac{\frac{[\mathcal{A}]^1}{\mathcal{A} \vee \mathcal{B}} \vee I \quad \frac{\mathcal{A}}{\mathcal{A} \vee \mathcal{C}} \vee I}{(\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C})} \wedge I \\ \mathcal{A} \Rightarrow (\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C}) \Rightarrow I, 1$$

Ovaj dokaz je dokaz tvrđenja  $\mathcal{A} \vdash \mathcal{A} \Rightarrow (\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C})$  (što je slabije tvrđenje od tvrđenja  $\vdash \mathcal{A} \Rightarrow (\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C})$ ).

**Primer 9.47.** U sistemu prirodne dedukcije važi  $\forall x \mathcal{A}, \forall x (\mathcal{A} \Rightarrow \mathcal{B}) \vdash \forall x \mathcal{B}$ :

$$\frac{\frac{\forall x \mathcal{A} \quad \forall x (\mathcal{A} \Rightarrow \mathcal{B})}{\mathcal{A} \Rightarrow \mathcal{B}} \Rightarrow E}{\frac{\mathcal{B}}{\forall x \mathcal{B}}} \forall I$$

**Primer 9.48.** Formula  $\neg(\exists x)p(x) \Rightarrow (\forall y)\neg p(y)$  je teorema sistema prirodne dedukcije:

$$\frac{\frac{\frac{[p(z)]^1}{(\exists x)p(x)} \exists I \quad [\neg(\exists x)p(x)]^2}{\frac{\perp}{\neg p(z)}} \neg I, 1}{\frac{(\forall y)\neg p(y)}{\neg(\exists x)p(x) \Rightarrow (\forall y)\neg p(y)}} \forall I \\ \neg(\exists x)p(x) \Rightarrow (\forall y)\neg p(y) \Rightarrow I, 2$$

**Primer 9.49.** Formula  $(\exists x)(\forall y)p(x, y) \Rightarrow (\forall y)(\exists x)p(x, y)$  je teorema sistema prirodne dedukcije (i za klasičnu i za intuicionističku logiku). Neki matematičar bi ovu formulu (neformalno) dokazao na sledeći način:

1. Prepostavimo da važi  $(\exists x)(\forall y)p(x, y)$ .
2. Prepostavimo da važi  $(\forall y)p(x', y)$  za neko  $x'$ .
3. Neka je  $y'$  proizvoljni objekat. Tada važi  $p(x', y')$ .
4. Iz  $p(x', y')$  sledi da važi  $(\exists x)p(x, y')$ .
5. Objekat  $y'$  je proizvoljan, pa važi  $(\forall y)(\exists x)p(x, y)$ .
6. Iz  $(\exists x)(\forall y)p(x, y)$  i iz toga što pretpostavka  $(\forall y)p(x', y)$  ima za posledicu  $(\forall y)(\exists x)p(x, y)$  (a  $(\forall y)p(x, y)$  ne zavisi od  $x'$ ), sledi  $(\forall y)(\exists x)p(x, y)$ .
7. Iz pretpostavke  $(\exists x)(\forall y)p(x, y)$  sledi  $(\forall y)(\exists x)p(x, y)$ , pa važi  $(\exists x)(\forall y)p(x, y) \Rightarrow (\forall y)(\exists x)p(x, y)$ .

Ovaj dokaz može se precizno opisati u vidu dokaza u sistemu prirodne dedukcije:

$$\begin{array}{c}
 \frac{[(\forall y)p(x', y)]^1}{p(x', y')} \forall E \\
 \frac{}{(\exists x)p(x, y')} \exists I \\
 \frac{[(\exists x)(\forall y)p(x, y)]^2}{(\forall y)(\exists x)p(x, y)} \forall I \\
 \frac{}{(\exists x)(\forall y)p(x, y)} \exists E, 1 \\
 \frac{}{(\exists x)(\forall y)p(x, y) \Rightarrow (\forall y)(\exists x)p(x, y)} \Rightarrow I, 2
 \end{array}$$

Naredna teorema povezuje semantička i deduktivna svojstva klasične logike (ona važi i za iskaznu i za predikatsku logiku).

**Teorema 9.10.** Formula je teorema sistema prirodne dedukcije za klasičnu logiku ako i samo ako je valjana.

## 9.4 Rešavanje problema svodenjem na problem valjanosti

Ukoliko je potrebno dokazati da je neko tvrđenje  $\mathcal{B}$  logička posledica skupa tvrđenja  $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n$ , onda je dovoljno dokazati da je formula  $\mathcal{A}_1 \wedge \mathcal{A}_2 \wedge \dots \wedge \mathcal{A}_n \Rightarrow \mathcal{B}$  valjana. Dokazivanje valjanosti formule može se sprovesti namenskim programima za to. Tipična situacija za ovakvu primenu je dokazivanje teorema određene teorije, pri čemu su  $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n$  aksiome te teorije, a  $\mathcal{B}$  teorema koju treba dokazati. Međutim, za mnoge teorije, krajnje je nepraktično teoreme dokazivati na ovaj opšti način. Za mnoge teorije logike prvog reda postoje i specijalizovani dokazivači zasnovani na specifičnostima tih teorija.

### 9.4.1 Porodični odnosi

U bazama podataka državne uprave postoje informacije o svim građanima koji uključuju informacije o oba roditelja. Državnoj upravi je ponekad potrebno da proveri da li su neke dve osobe u nekom porodičnom odnosu, ali bilo bi iracionalno da se svi mogući rodbinski odnosi čuvaju eksplicitno u bazama podataka. Umesto toga, moguće je proveriti neke konkretne porodične odnose i neka opšta tvrđenja korišćenjem logičkog rasuđivanja.

Porodične relacije (kao što su „biti otac“, „biti majka“, „biti brat“, „biti tetka“ i slično) mogu se opisati formulama logike prvog reda. Na primer,

$$\mathcal{A} = \forall x \forall y (brat(x, y) \Leftrightarrow musko(x) \wedge x \neq y \wedge \exists z (roditelj(z, x) \wedge roditelj(z, y)))$$

$$\mathcal{B} = \forall x \forall y (brat(x, y) \wedge musko(y) \Leftrightarrow brat(y, x) \wedge musko(x))$$

Navedene formule mogu se interpretirati na različite načine. Ali ako se interpretiraju nad nekim konkretnim skupom osoba a predikatski simboli se interpretiraju na prirođan način (npr. relacijski simbol *roditelj* preslikava se u uobičajenu relaciju „biti roditelj“ nad skupom osoba), njihovo značenje se poklapa sa intuitivnim i može se razmatrati njihova valjanost na osnovu znanja o porodičnim odnosima. Na primer, formula  $\mathcal{B}$  valjana je za svaki skup osoba i prirodno interpretirane predikatske simbole. To se može utvrditi nekakvim „rešavačem za porodične odnose“ koje većina ljudi ima u svojim glavama iako obično ne u nekom eksplicitnom obliku. O valjanosti formule  $\mathcal{B}$  i sličnih možemo da rasuđujemo korišćenjem takvog rešavača (ako ga imamo) ili korišćenjem „univerzalnog rešavača“, tj. dokazivača za logiku prvog reda, kao što je opisano u nastavku.

Formula  $\mathcal{B}$  valjana je za svaki skup osoba i prirodno interpretirane predikatske simbole, ali nije valjana (nije valjana u svakoj  $\mathcal{L}$  strukturi). Ona je, međutim, logička posledica formule  $\mathcal{A}$ . „Univerzalni rešavač“ ćemo, dakle, moći da upotrebimo da proverimo da li je formula  $\mathcal{B}$  logička posledica formule  $\mathcal{A}$ , tj. da proverimo da li je važi  $\mathcal{A} \models \mathcal{B}$ , tj. da proverimo da li je formula  $\mathcal{A} \Rightarrow \mathcal{B}$  valjana. Na opisani način proveravamo tvrđenja oblika  $\Gamma \models \mathcal{B}$ . Ovim pristupom razmatra se „opšta“ valjanost ali skup  $\Gamma$  ima ulogu da efektivno ograniči skup interpretacija koje se razmatraju: ovakvim razmatranjem pokrivene su samo interpretacije u kojima su formule skupa  $\Gamma$  valjane. Formule  $\Gamma$  imaju ulogu svojevrsnih aksioma i matematičke teorije se grade u istom duhu. Da bi važilo  $\Gamma \models \mathcal{B}$  za širok skup formula  $\mathcal{B}$ , u skup  $\Gamma$  dodaćemo više jednostavnih svojstava porodičnih odnosa, kao što su:

$$\forall x \forall y (sestra(x, y) \Leftrightarrow zensko(x) \wedge x \neq y \wedge \exists z (roditelj(z, x) \wedge roditelj(z, y)))$$

$$\forall x \forall y (majka(x, y) \Leftrightarrow zensko(x) \wedge roditelj(x, y))$$

$$\forall x \forall y (otac(x, y) \Leftrightarrow musko(x) \wedge roditelj(x, y))$$

$$\forall x \forall y (tetka(x, y) \Leftrightarrow \exists z (sestra(x, z) \wedge roditelj(z, y)))$$

Ako u skup  $\Gamma$  dodamo, na primer, i formule:

$$musko(MarkoJankovic)$$

$$musko(LazarJankovic)$$

*otac(JovanJankovic, MarkoJankovic)*

*otac(JovanJankovic, LazarJankovic)*

onda može da se pokaže da važi i  $\Gamma \models brat(MarkoJankovic, LazarJankovic)$ . Dakle, opisanim mehanizmom može se rasuđivati o porodičnim odnosima među konkretnim osobama.

#### 9.4.2 Verifikacija softvera

U poglavlju 8.4.5 bilo je reči o verifikaciji softvera i razmatrano je da li je narednu funkciju  $f$  napisanu na programskom jeziku C:

```
int f(int y)
{
    int x;
    x = x ^ y;
    y = x ^ y;
    x = x ^ y;
    return x * x;
}
```

moguće zameniti narednom funkcijom  $g$  (to je uvek tako, bez obzira na to što promenljiva  $x$  nije inicijalizovana u funkciji  $f$  i njena inicijalna vrednost mogla bi da bude bilo šta):

```
int g(int n)
{
    return n * n;
}
```

Pokazano je da u ovom konkretnom slučaju može da se razmatra dejstvo bitovskih operatora nad pojedinačnim bitovima i sredstvima iskazne logike pokazano je da je svaki bit programske promenljive  $x$  pre naredbe  $return x * x;$  u funkciji  $f$  jednak odgovarajućem bitu programske promenljive  $n$  pre naredbe  $return n * n;$  u funkciji  $g$ , čime je dokazana ekvivalentnost dve funkcije.

Sredstvima logike prvog reda to tvrđenje može se dokazati na drugačiji način, primenljiv u širem skupu situacija. Dokaz će važiti za bilo koju širinu tipa `int`. Kao u iskaznom slučaju, pogodno je u razmatranje uvesti i promenljive za medurezultate:

```
x1 = x ^ y;
y1 = x1 ^ y;
x2 = x1 ^ y1;
```

Da bi se dokazalo da su funkcije  $f$  i  $g$  ekvivalentne, dovoljno je dokazati da je formula

$$\forall x \forall x_1 \forall x_2 \forall y \forall y_1 \quad x_1 = (x \wedge y) \wedge y_1 = (x_1 \wedge y) \wedge x_2 = (x_1 \wedge y_1) \Rightarrow x_2 = y$$

valjana u strukturi koju čine moguće vrednosti tipa `int`. Ukoliko nemamo rešavač specijalizovan za tu strukturu, moramo da se okrenemo „univerzalnom rešavaču“ za logiku prvog reda. Ali tada moramo da u igru uvedemo uslove  $\Gamma$  koji opisuju strukturu vrednosti tipa `int` sa operatom  $\wedge$ . Može se dokazati (što nećemo ovde uraditi) da za tu strukturu i operator  $\wedge$  važe sledeća tvrđenja koje ćemo koristiti kao aksiome:

$$\begin{aligned} \forall u \forall v \quad (u \wedge v) &= (v \wedge u) \\ \forall u \forall v \quad ((u \wedge v) \wedge v) &= u \end{aligned}$$

Primenom metoda rezolucije može se pokazati da je navedena formula logička posledica navedenih aksioma i aksiomu jednakosti (videti poglavlje 9.4.5).

#### 9.4.3 Rezanje ploča

Nameštaj se često pravi od pločastih materijala i tada je od velike važnosti raspoređivanje komponenti nameštaja po pojedinačnim pločama. Što manje upotrebljenih ploča – veća isplativost. Razmotrimo jednostavan primerak ovog problema: da li je moguće od ploče dimenzija  $100 \times 60$  napraviti komade dimenzija  $80 \times 30$  i  $50 \times 40$  (prepostavimo, jednostavnosti radi, da duže stranice moraju da budu paralelne dužim stranicama osnovne ploče)? Dokazaćemo da to nije moguće.

Smestimo sve elemente u koordinatni sistem: neka je donji levi ugao ploče u tački  $(0, 0)$  i neka su  $(x_1, y_1)$  i  $(x_2, y_2)$  donji levi uglovi dva željena pločasta dela. Da bi delovi bili na ploči mora da važi formula  $\mathcal{A}$ :

$$\begin{aligned} 0 \leq x_1 \wedge x_1 + 80 &\leq 100 \wedge \\ 0 \leq y_1 \wedge y_1 + 30 &\leq 60 \wedge \end{aligned}$$

$$\begin{aligned} 0 \leq x_2 \wedge x_2 + 50 \leq 100 \wedge \\ 0 \leq y_2 \wedge y_2 + 40 \leq 60 \end{aligned}$$

Dva elementa nemaju presek samo ako postoji horizontalna ili vertikalna prava koja ih razdvaja, tj. ako važi formula  $\mathcal{B}$ :

$$(x_1 + 80 \leq x_2 \vee x_2 + 50 \leq x_1 \vee y_2 + 40 \leq y_1 \vee y_1 + 30 \leq y_2)$$

Da bismo dokazali da traženo rezanje nije moguće, dovoljno je da dokažemo da važi  $\mathcal{A} \models \neg\mathcal{B}$ , tj. da dokažemo da je univerzalno zatvorene formule  $\mathcal{A} \Rightarrow \neg\mathcal{B}$  valjana formula. Ova formula, međutim, nije valjana. No, ona je valjana u strukturi realnih brojeva sa prirodno interpretiranim simbolima  $+$  i  $<$ , a to je jedina interpretacija koja nas zanima. Valjanost u toj strukturi može da dokaže SMT rešavač za linearu aritmetiku nad realnim brojevima. Valjanost univerzalnog zatvorenja formule  $\mathcal{A} \Rightarrow \neg\mathcal{B}$  biće tada dokazana time što će biti pokazano da je formula  $\mathcal{A} \wedge \mathcal{B}$  nezadovoljiva (poglavlje 9.4.5).

#### 9.4.4 PROLOG

Jezik PROLOG najznačajniji je predstavnik paradigme logičkog programiranja, u kojoj se koristi logika kao deklarativni jezik za opisivanje problema, a dokazivač teorema kao mehanizam za rešavanje. Ime PROLOG dolazi od engleskih reči „PROGramming in LOGic“. Jezik PROLOG i prvi interpretator za njega razvijeni su na Univerzitetu u Marseju 1972. godine. Vreme najveće popularnosti PROLOG-a je prošlo, ali se on i dalje široko koristi, uglavnom za probleme iz oblasti veštacke inteligencije: od medicinskih sistema pa do istraživanja podataka. Pogodan je za brz razvoj prototipova jer se obrada ulaza i izlaza, parsiranje i druge slične operacije implementiraju jednostavno, a mehanizam za netrivijalno rasudivanje je jednostavno raspoloživ.

Mehanizam izvođenja zaključaka u PROLOG-u zasniva se na metodu rezolucije i na korišćenju Hornovih kluza — kluza u kojima postoji najviše jedan literal koji nije pod negacijom. Za ispitivanje zadovoljivosti skupova kluza, zahvaljujući njihovoj specifičnoj formi, koristi se algoritam koji je polinomske složenosti. Četiri vrste Hornovih kluza i odgovarajuće formule logike prvog reda prikazani su u narednoj tabeli (formule  $\mathcal{A}_i$  su atomičke). Može se dokazati da svaki nezadovoljiv skup Hornovih kluza mora da sadrži bar jednu činjenicu i bar jednu ciljnju kluzu.

Vrsta	logika prvog reda	PROLOG
implikaciona kluza	$\neg\mathcal{A}_1 \vee \dots \vee \neg\mathcal{A}_n \vee \mathcal{A}$	$\mathcal{A} : -\mathcal{A}_1, \dots, \mathcal{A}_n.$
ciljna kluza	$\neg\mathcal{A}_1 \vee \dots \vee \neg\mathcal{A}_n$	? - $\mathcal{A}_1, \dots, \mathcal{A}_n.$
činjenica	$\mathcal{A}$	$\mathcal{A}.$
prazna kluza	$\square$	false

PROLOG sistemi obično sadrže interaktivni interpretator a neki sistemi omogućavaju i kompiliranje kôda. Komunikacija sa PROLOG interpretatorom odvija se kroz komandni prozor, a prompt interpretatora obično izgleda ovako: ?-. PROLOG konvencija je da se konstante zapisuju malim početnim slovom, a promenljive velikim početnim slovom.

**Primer 9.50.** Prepostavimo da je zadata činjenica

man(sokrat).

(nova činjenica može se učitati iz datoteke, kao deo programa a može se zadati i interaktivno, na sledeći način:  
?- assert(man(sokrat)).) Nakon ovoga, upit

?- man(sokrat).

uspeva, tj. daje rezultat Yes. Naime, da bi ovaj upit bio zadovoljen, činjenica man(sokrat) rezolvira se sa kluzom  $\neg \text{man}(\text{sokrat})$  (dobijenom iz upita) i daje praznu kluzu, kao što je i trebalo. Time je, praktično, dokazano da je  $\text{man}(\text{sokrat}) \Rightarrow \text{man}(\text{sokrat})$  valjana formula.

Prepostavimo da je (na primer sa ?- assert(mortal(X) :- man(X)).) zadato i pravilo:

mortal(X) :- man(X).

U ovom pravilu, predikat mortal(X) je glava pravila a (jednočlani) niz predikata man(X) je rep pravila. Upit:

?- mortal(sokrat).

uspeva (daje odgovor Yes). Da bi ovaj upit bio zadovoljen, klauza  $\neg\text{man}(X) \vee \text{mortal}(X)$  (dobijena iz zadataog pravila) rezolvira se sa klauzom  $\neg\text{mortal}(\text{sokrat})$  (dobijenom iz upita) i daje rezolventu, tj. novi cilj  $\neg\text{man}(\text{sokrat})$ . On uspeva, jer sa klauzom  $\text{man}(\text{sokrat})$  (dobijenom iz zadate činjenice) daje praznu klauzu. Time je, praktično, dokazano da je

$(\text{man}(\text{sokrat}) \wedge \forall x (\text{man}(x) \Rightarrow \text{mortal}(x))) \Rightarrow \text{mortal}(\text{sokrat})$

valjana formula.

Ako se zada upit:

? -  $\text{mortal}(X)$ .

onda se metodom rezolucije pokušava dokazivanje nezadovoljivosti skupa klauza:

$\text{man}(\text{sokrat})$   
 $\neg\text{man}(X) \vee \text{mortal}(X)$   
 $\neg\text{mortal}(Y)$

Primetimo da je u trećoj klauzi promenljiva preimenovana u Y, da ne bi došlo do poklapanja imena promenljive u dve klauze. Ciljna (treća) klauza može da se rezolvira sa drugom klauzom, korišćenjem unifikatora  $[Y \mapsto X]$  dajući novi cilj

$\neg\text{man}(X)$

tj. nakon preimenovanja promenljive X:

$\neg\text{man}(X')$

Rezolviranjem ove klauze sa prvom klauzom iz početnog skupa, korišćenjem unifikatora  $[X' \mapsto \text{sokrat}]$  dobija se prazna klauza, pa je dokazana nezadovoljivost datog skupa klauza i PROLOG vraća rezultat:

Yes

i daje odgovor:

X = sokrat

To je jedino moguće rešenje i ako ukucamo simbol ; (čime tražimo druge moguće unifikatore) dobićemo odgovor No.

Naravno, upiti

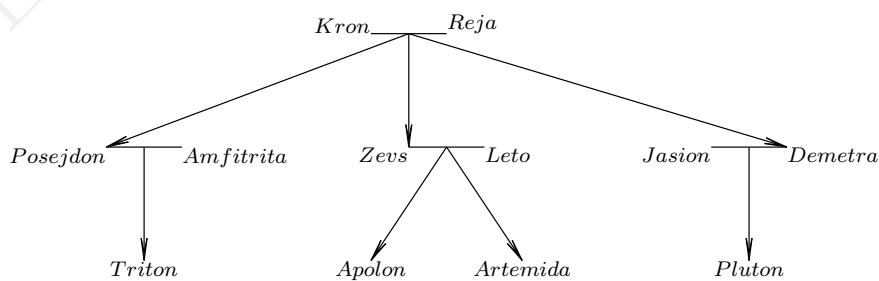
?-  $\text{man}(\text{platon})$ .

i

? -  $\text{mortal}(\text{platon})$ .

ne uspevaju i daju odgovor No (sem ako nije zadata i činjenica  $\text{man}(\text{platon})$ ).

**Primer 9.51.** Definisanje odnosa u PROLOG-u može se ilustrovati na primeru porodičnih odnosa (kao što su otac, majka, brat, tetka i slično) (videti poglavlje 9.4.1) i to nad skupom starogrčkih božanstava, čije su veze prikazane na narednoj slici (u vidu porodičnog stabla):



musko(kron).  
musko(posejdon).  
musko(zevs).  
musko(jasion).  
musko(triton).

```

musko(apolon).
musko(pluton).

zensko(reja).
zensko(amfitrita).
zensko(leto).
zensko(demetra).
zensko(artemida).

roditelj(kron,posejdon).
roditelj(reja,posejdon).
roditelj(kron,zevs).
roditelj(reja,zevs).
roditelj(kron,demetra).
roditelj(reja,demetra).
roditelj(posejdon,triton).
roditelj(amfitrita,triton).
roditelj(zevs,apolon).
roditelj(leto,apolon).
roditelj(zevs,artemida).
roditelj(leto,artemida).
roditelj(jasion,pluton).
roditelj(demetra,pluton).

predak(X,Y) :- roditelj(X,Y).
predak(X,Y) :- roditelj(X,Z), predak(Z,Y).

majka(X,Y) :- zensko(X), roditelj(X,Y).
otac(X,Y) :- musko(X), roditelj(X,Y).
brat(X,Y) :- musko(X), roditelj(Z,X), roditelj(Z,Y), X\==Y.
sestra(X,Y) :- zensko(X), roditelj(Z,X), roditelj(Z,Y), X\==Y.

tetka(X,Y) :- sestra(X,Z), roditelj(Z,Y).
stric(X,Y) :- brat(X,Z), otac(Z,Y).
ujak(X,Y) :- brat(X,Z), majka(Z,Y).
bratodstrica(X,Y) :- musko(X), otac(Z,X), stric(Z,Y).
sestraodstrica(X,Y) :- zensko(X), otac(Z,X), stric(Z,Y).
bratodujaka(X,Y) :- musko(X), otac(Z,X), ujak(Z,Y).
sestraodujaka(X,Y) :- zensko(X), otac(Z,X), ujak(Z,Y).
bratodtetke(X,Y) :- musko(X), majka(Z,X), tetka(Z,Y).
sestraodtetke(X,Y) :- zensko(X), majka(Z,X), tetka(Z,Y).

```

*U relacijama brat i sestra, predikat  $X \neq Y$  ima vrednost tačno ako je X različito od Y. U suprotnom, ima vrednost netačno. U nastavku je dato nekoliko primera upita i rezultata koje sistem daje.*

```
?- stric(posejdon,apolon).
```

Yes

```
?- ujak(X,Y).
```

```

X = zevs,
Y = pluton
X = zevs,
Y = pluton
X = posejdon,
Y = pluton
X = posejdon,
Y = pluton

```

```
?- sestraodstrica(X,Y).
```

```
X=artemida,
Y=triton
X=artemida,
Y=triton
```

U navedenim primerima, mogu se primetiti ponavljanja istih rešenja. U slučaju upita `ujak(X,Y)`, razlog za to je što zadovoljavanje ovog cilja, zavisi od zadovoljavanja podcilja `brat(X,Z)`, koji zavisi od zadovoljavanja podciljeva `roditelj(W,X)` i `roditelj(W,Z)`. U slučaju da važi `X=zevs`, `Y=pluton` i `Z=demetra`, onda postoje dve mogućnosti za `W`, što su `kron` i `reja`. Kako sistem dva puta nalazi način da zadovolji sve potciljeve u kojima važi `X=zevs` i `Y=pluton`, dva puta navodi tu kombinaciju kao rešenje. Slučaj upita `sestraodstrica(X,Y)` je analogan.

Primetimo da pravilu

```
otac(X,Y) :- musko(X), roditelj(X,Y).
```

odgovara formula

$$\forall x \forall y (\text{musko}(x) \wedge \text{roditelj}(x,y) \Rightarrow \text{otac}(x,y))$$

što je slabije od formule (iz poglavlja 9.4.1):

$$\forall x \forall y (\text{otac}(x,y) \Leftrightarrow \text{musko}(x) \wedge \text{roditelj}(x,y))$$

Zato iz navedenih činjenica PROLOG može da pokaže da važi `otac(kron,zevs)`. ali, da nije data činjenica `roditelj(kron,zevs)`, PROLOG ne bi mogao da je izvede čak i da ima na raspolaganju činjenicu `otac(kron,zevs)`. Naime, navedeni PROLOG program dizajniran je tako da iz informacija o roditeljstvu izvodi sve druge porodične odnose. Ukoliko je potrebno i obratno, onda u program treba dodati dodatna pravila.

Da bi PROLOG mogao da bude upotrebljiv kao programski jezik, u njegovoj semantici postoje odstupanja od semantike logike prvog reda (na primer, umesto negacije koja je kao u logici prvog reda, u PROLOG-u se koristi takozvana „negacija kao neuspeh“).

U PROLOG-u se lako mogu implementirati procedure kao što je DPLL, ali detaljniji prikaz jezika i njegove upotrebe izlazi iz okvira ove knjige.

#### 9.4.5 FOL dokazivači i SMT rešavači i njihovi ulazni formati

Programe koji rešavaju instance problema valjanosti ili zadovoljivosti u logici prvog reda („univerzalne rešavače“) zovemo obično FOL dokazivači (eng. FOL-provers, od *first-order logic*). Većina savremenih FOL dokazivača zasnovana je na metodi rezolucije, obogaćenoj mnogim dodatnim tehnikama i heurstikama. Neki od danas popularnih FOL dokazivača su Vampire, E i Spass.

FOL dokazivači obično očekuju ulaz u nekom od TPTP<sup>3</sup> formata. Jedan od tih formata je FOF format. U ovom formatu, formule se navode jedna po jedna sa oznakom o tome da li se radi o aksiomi ili tvrđenju koje dokazivač treba da dokaže. U slučaju primera 9.42, problem bi mogao da se zapiše na sledeći način.

```
fof(a1, axiom, (! [X,Y] : (above(X,Y) => ~above(Y,X)))).  
fof(a2, axiom, (! [X,Y] : (above(X,Y) <=> below(Y,X)))).  
fof(a3, axiom, (! [X,Y,Z] : ((above(X,Y) & above(Y,Z)) => above(X,Z)))).  
fof(cn, conjecture, (! [X,Y,Z] : ((above(Y,X) & below(Z,X)) => above(Y,Z)))).
```

Za navedeni opis problema (u vidu datoteke `above.tptp`), na primer, dokazivač Vampire vratiće odgovor da je odgovarajući skup klauza nezadovoljiv, tj. da je navedena formula logička posledica navedenih aksioma:

```
% Refutation found.  
% Szs status Theorem for above  
% Szs output start Proof for above  
2. ! [X0,X1] : (above(X0,X1) <=> below(X1,X0)) [input]  
3. ! [X0,X1,X2] : ((above(X1,X2) & above(X0,X1)) => above(X0,X2)) [input]  
4. ! [X0,X1,X2] : ((below(X2,X0) & above(X1,X0)) => above(X1,X2)) [input]  
5. ~! [X0,X1,X2] : ((below(X2,X0) & above(X1,X0)) => above(X1,X2)) [negated conjecture 4]  
7. ! [X0,X1,X2] : (above(X0,X2) | (~above(X1,X2) | ~above(X0,X1))) [ennf transformation 3]  
8. ! [X0,X1,X2] : (above(X0,X2) | ~above(X1,X2) | ~above(X0,X1)) [flattening 7]  
9. ? [X0,X1,X2] : (~above(X1,X2) & (below(X2,X0) & above(X1,X0))) [ennf transformation 5]
```

<sup>3</sup>TPTP (Thousands of Problems for Theorem Provers) je biblioteka problema za automatske dokazivače teorema. U okviru nje definisano je i nekoliko formata za zapis formula logike prvog reda.

```

10. ? [X0,X1,X2] : (~above(X1,X2) & below(X2,X0) & above(X1,X0)) [flattening 9]
11. ! [X0,X1] : ((above(X0,X1) | ~below(X1,X0)) & (below(X1,X0) | ~above(X0,X1))) [nnf transformation 2]
12. ? [X0,X1,X2] : (~above(X1,X2) & below(X2,X0) & above(X1,X0)) => (~above(sK1,sK2) & below(sK2,sK0) & above(sK1,sK0)) [choosing 10]
13. ~above(sK1,sK2) & below(sK2,sK0) & above(sK1,sK0) [skolemisation 10,12]
16. ~below(X1,X0) | above(X0,X1) [cnf transformation 11]
17. ~above(X1,X2) | above(X0,X2) | ~above(X0,X1) [cnf transformation 8]
18. above(sK1,sK0) [cnf transformation 13]
19. below(sK2,sK0) [cnf transformation 13]
20. ~above(sK1,sK2) [cnf transformation 13]
23. above(sK0,sK2) [resolution 16,19]
28. ~above(X1,sK0) | above(X1,sK2) [resolution 17,23]
29. above(sK1,sK2) [resolution 28,18]
30. $false [subsumption resolution 29,20]
% Szs output end Proof for above
% -----
% Version: Vampire 4.2.2 (commit e1949dd on 2017-12-14 18:39:21 +0000)
% Termination reason: Refutation

% Memory used [KB]: 4733
% Time elapsed: 0.103 s

```

Tvrđenje iz poglavlja 9.4.2 može se opisati na sledeći način:

```

fof(a1, axiom, (! [U,V] : ( xor(xor(U,V),V)=U ))).
fof(a1, axiom, (! [U,V] : ( xor(U,V)=xor(V,U)))).
fof(cn, conjecture, (! [X,X1,X2,Y,Y1] :
((X1=xor(X,Y) & Y1=xor(X1,Y) & X2=xor(X1,Y1)) => X2=Y))).
```

Dokazivač Vampire vratiće odgovor da je odgovarajući skup klauza nezadovoljiv, tj. da je navedena formula logička posledica navedenih aksioma i aksioma jednakosti (nije potrebno eksplicitno zadavati aksiome jednakosti).

Za mnoge praktične probleme, umesto „univerzalnog rešavača“, FOL dokazivača, pogodno je koristiti specijalizovane rešavače za neke konkretnе teorije ili konkretnе strukture.

Na primer, za dokazivanje nezadovoljivosti formule iz primera sa rezanjem ploča (poglavlje 9.4.3) pogodno je koristiti specijalizovani SMT rešavač za linearnu aritmetiku i to za fragment bez kvantifikatora (jer su sve promenljive implicitno egzistencijalno kvantifikovane) – ta teorija obično se označava sa QF\_LIA. Potrebno je dokazati da je formula  $\mathcal{A} \wedge \mathcal{B}$  nezadovoljiva i taj zadatak može se opisati u vidu datoteke ploče.smt sa narednim sadržajem u formatu smt-lib:

```

(set-logic QF_LIA)
(declare-const x1 Int)
(declare-const y1 Int)
(declare-const x2 Int)
(declare-const y2 Int)

(assert (<= 0 x1))
(assert (<= 0 (+ x1 80) 100))
(assert (<= 0 y1))
(assert (<= 0 (+ y1 30) 60))
(assert (<= 0 x2))
(assert (<= 0 (+ x2 50) 100))
(assert (<= 0 y2))
(assert (<= 0 (+ y2 40) 60))
(assert (or (<= (+ x1 80) x2) (<= (+ x2 50) x1) (<= (+ y1 30) y2) (<= (+ y2 40) y1)))
(check-sat)
(get-model)
```

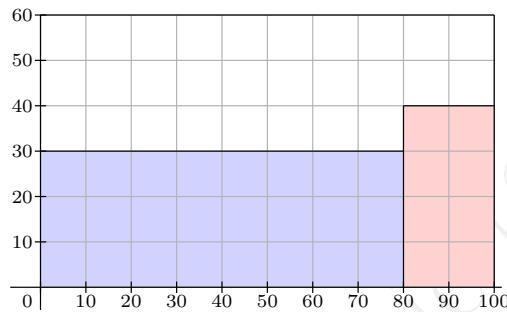
Za navedeni opis problema (u vidu datoteke ploče.smt), na primer, rešavač z3 vratiće odgovor unsat.

unsat

Ukoliko dimenzije drugog komada nisu  $50 \times 40$  nego, na primer,  $20 \times 40$ , onda za odgovarajući smt datoteku (samo je vrednost 50 zamjenjena sa 20), rešavač z3 daje odgovor koji sadrži i jedno rešenje kako treba iseći tražene komade ploče:

```
sat
(model
  (define-fun x2 () Int
    80)
  (define-fun y1 () Int
    0)
  (define-fun x1 () Int
    0)
  (define-fun y2 () Int
    0)
)
```

Ovo rešenje, ovaj model zadate formule govori da postoji (barem jedno) željeno rezanje i u njemu je donji levi ugao prvog komada u tački  $(x_1, y_1) = (0, 0)$ , a donji levi ugao drugog komada u tački  $(x_2, y_2) = (80, 0)$ , kao što je ilustrovano na narednoj slici:



*Deo III*

---

## Mašinsko učenje

---

Elektronsko izdanje (2020)



## Glava 10

# Rešavanje problema korišćenjem mašinskog učenja

Rešavanje problema korišćenjem automatskog rasuđivanja počiva na formalnom i obično potpunom opisu problema koji je potrebno rešiti. Na primer, da bismo rešili problem  $n$  dama, potrebno je precizno definisati sva pravila koja se tiču raspoređivanja figura i njihovog napadanja. Ako je potrebno donositi zaključke o geometrijskim problemima, potrebno je potpuno precizno zadati aksiome geometrije i tvrđenje koje je potrebno dokazati. Međutim, u velikom broju praktičnih problema nije moguće potpuno precizno opisati pravila sveta u kojem se rešava problem, pa ni svojstva rešenja. Na primer, u slučaju prepoznavanja lica nije lako precizno definisati šta je lice. Prirodan pokušaj da se lice definiše u terminima njegovih delova dalje povlači pitanje definicija tih delova, kao i odnosa među njima. Ipak, određena pravilnost nesporno postoji. Sličan je i problem prevodenja rečenica sa jednog jezika na drugi. Iako je poznato da jezik ima jaku strukturu, ona je previše komplikovana i prepuna izuzetaka koji su dovoljni da učine formalno preciziranje pravila previše teškim. Samim tim još je veći problem opisati kako se struktura jednog jezika preslikava u strukturu drugog jezika na koji je potrebno izvršiti prevodenje.

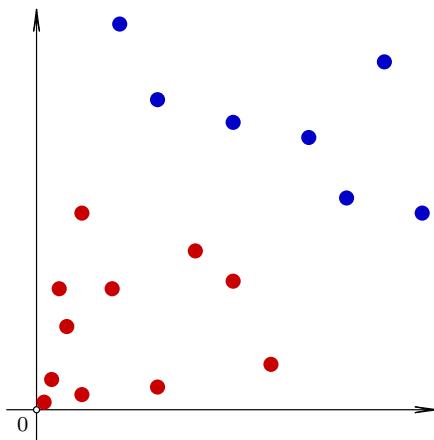
Da bi neki problem bio rešen potrebno je da postoji njegova specifikacija. U prethodnim primerima nije teško dati nekakvu, već dati eksplicitnu i preciznu specifikaciju. Moguće je i dati implicitnu specifikaciju, velikom količinom primera onoga što jeste rešenje i onoga što nije rešenje problema, odnosno kroz konkretnе podatke. Ovakva specifikacija nužno je slabija od formalne, ali nekada je najbolje što možemo uraditi u praksi. Mašinsko učenje bavi se upravo rešavanjem problema za koje je teško dati formalnu specifikaciju, ali je dostupna velika količina podataka koji na neki način ilustruju rešenja pojedinačnih instanci problema. Zadatak metoda mašinskog učenja je da u pruženim podacima uoče relevantne zakonitosti i da ih formulišu u vidu nekog matematičkog modela. Ključni cilj metoda mašinskog učenja je, dakle, dobra generalizacija, tj. dobro ponašanje dobijenog matematičkog modela u situacijama koje nisu bile opisane podacima na osnovu kojih je model dobijen. Drugim rečima, ključni cilj je model koji će biti koristan u rešavanju budućih instanci problema.

U modelovanju podataka greške su nepoželjne ali moguće. Greške modela mogu da postoje u odnosu na podatke iz kojih je model dobijen ili u odnosu na podatke u kasnije primeni. Važnije su ove druge jer je, ključni cilj mašinskog učenja, kao što je rečeno, kreiranje modela koji će biti korisni u rešavanju budućih instanci problema.

Raspoloživi podaci nekada mogu sadržati greške, pa i protivrečnosti. Takođe, mogu davati nepotpunu sliku problema koji se rešava. Otud je jasno da logika nije pogodan formalni okvir za mašinsko učenje, te metode mašinskog učenja nisu deduktivne već induktivne metode zaključivanja, obično zasnovane na verovatnoći i statistici. Verovatnosni okvir pruža im robustnost u odnosu na greške u podacima, a često omogućava i kvantifikaciju pouzdanosti ponuđenog rešenja. Pored toga, metode mašinskog učenja tipično vrše odlučivanje veoma brzo (znatno brže nego, na primer, metode automatskog rasudivanja koje svoju egzaktnost plaćaju visokom računskom zahtevnošću prilikom odlučivanja). Otud je njihova primena preferirana i u nekim slučajevima kada jeste moguće u potpunosti precizirati problem, ali je prostor pretrage ogroman. Takav je primer igranja igre go u kojoj je metodama mašinskog učenja pobeden ljudski svetski šampion. Ipak, u rešenjima zasnovanim na mašinskom učenju greške se uvek mogu očekivati te taj pristup nije primeren domenima u kojima greška nije dozvoljena (na primer, u automatskoj kontroli metroa).

Osnovni koncepti mašinskog učenja u nastavku teksta biće ilustrovani kroz naredni primer.

**Primer 10.1.** *Prepostavimo da je potrebno napraviti specijalizovani pretraživač interneta koji omogućuje*



Slika 10.1: Plavi krugovi predstavljaju računarske članke, a crveni ostale.  $x$  koordinata predstavlja frekvenciju reči „računar“, a koordinata  $y$  predstavlja frekvenciju reči „datoteka“.

*korisnicima da pretražuju samo računarske članke. Pored mnoštva elemenata koje ovaj sistem mora da ima, poput onih vezanih za komunikaciju na internetu, skladištenje informacija, interfejs prema korisniku i tako dalje, potrebno je da ima specijalizovan modul koji se bavi razvrstavanjem, odnosno klasifikacijom članaka na članke iz oblasti računarstva i na članke iz svih ostalih oblasti. Razmotrimo kako bi elementarna varijanta ovakvog modula mogla biti osmišljena.*

*Apstraktni aspekti teksta, poput semantike, stila i sličnih, iako vrlo relevantni za navedeni problem, ne dopuštaju laku analizu od strane mašine. Otud bi se elementarna varijanta sistema za razvrstavanje članaka morala zasnivati na neposredno dostupnim aspektima teksta. To su pre svega reči od kojih se tekst sastoji. Na sreću, reči mogu biti vrlo indikativne po pitanju oblasti kojoj tekst pripada, pošto mogu predstavljati stručne termine. Otud je zamislivo da se pomenuti problem može u zadovoljavajućoj meri rešiti razmatranjem frekvencija pojavljivanja pojedinih, pogodno odabranih reči. U pojednostavljenom pristupu, moguće je osloniti se na frekvenciju stručnih reči „računar“ i „datoteka“, čime se svaki tekst predstavlja kao tačka u dvodimenzionoj ravni. Očekuje se da će računarski tekstovi u tom slučaju biti predstavljeni tačkama koje su daleko od koordinatnog početka, a da će ostali tekstovi biti blizu koordinatnog početka, kao što je prikazano na slici 10.1. U idealnoj situaciji, te dve grupe tačaka su potpuno razdvojene i između njih postoji prava koja ih razdvaja. U tom slučaju, kasnije je moguće jednostavno odgovoriti na pitanje da li je članak računarski samo na osnovu toga sa koje strane te prave se nalazi tačka koja odgovara tom članku. Ukoliko ne postoji prava koja razdvaja dve grupe tačaka, potrebno je naći neku pravu koja većinu tačaka razdvaja ispravno. Važno pitanje je kako se i u takvom slučaju može odrediti prava koja najbolje razdvaja dve grupe članaka. Takvih algoritama ima više, a intuitivno, osnovna ideja je da se krene od bilo koje prave i da se ona po malo pomera dok se ne pozicionira najbolje moguće (po nekom preciznom kriterijumu) između dve grupe tačaka, tj. između dve klase. Detalji jednog takvog algoritma biće dati kasnije.*

**Primeri primena mašinskog učenja.** Mašinsko učenje uspešno se primenjuje u mnoštву praktičnih problema. Jedan od najstarijih, a još uvek zanimljivih praktičnih rezultata postignut je od strane sistema ALVINN zasnovanog na neuronskoj mreži, krajem osamdesetih godina dvadesetog veka, koji je naučen da bez ljudske pomoći vozi automobil auto-putem, u prisustvu drugih vozila i brzinom od oko 110km/h. Sistem je uspešno vozio na putu dužine oko 140km. Sa razvojem dubokih neuronskih mreža, sredinom prve decenije ovog veka, projekat razvoja autonomnih vozila dobio je novi zamah. Mnoge kompanije trenutno razvijaju vozila koja treba da budu u stanju da samostalno učestvuju u gradskoj vožnji, koja je značajno komplikovanija od vožnje na auto-putu. Izazovi za tehnike mašinskog učenja u ovom problemu uključuju kako prepoznavanje puta i učesnika u saobraćaju, tako i donošenje odluka. Slične metode koriste se i za učenje upravljanja kvadrotorima (malim letilicama sa četiri propelera) u cilju prenošenja predmeta. Kompanija Amazon razmatra mogućnost ovakvog načina dostavljanja svojih pošiljki u gradskim sredinama.

Jedan od najpoznatijih ranih primera primene mašinskog učenja je i sistem TD-Gammon za igranje igre *Backgammon* konstruisan devedesetih godina prošlog veka. Igrajući protiv sebe više od milion partija i nastavljajući da uči u igri sa ljudskim igračima, sistem je dostigao nivo igre u rangu svetskog šampiona. Na sličnim principima, ali koristeći modernije algoritme učenja konstruisan je sistem AlfaGo koji je 2015. i 2016. ubedljivo pobedio evropskog, a zatim i svetskog šampiona u igri go. Ova igra poznata je kao jedan od, do sada, najozbiljn-

nijih izazova veštačkoj inteligenciji u domenu igranja igara, pošto po broju mogućih stanja daleko prevazilazi i šah, što drastično otežava primenu tradicionalnih tehnika veštačke inteligencije poput algoritma minimaks sa alfa-beta odsecanjem.

Kompanije poput Amazona, koje se bave internet prodajom, odavno koriste sisteme koji na osnovu primera kupovnih transakcija korisnika uče kako da budućim korisnicima preporučuju proizvode koji bi ih mogli interesovati. Ovakvi sistemi i odgovarajući algoritmi učenja nazivaju se sistemima za preporučivanje (eng. *recommender systems*).

Sistemi za prepoznavanje govora takođe koriste mašinsko učenje u nekoj formi. Sistem Sphinx, takođe iz kraja osamdesetih, bio je u stanju da prepozna izgovorene reči uz prilagođavanje izgovoru različitih ljudi, različitim karakteristikama mikrofona, pozadinskoj buci i slično. U međuvremenu su takvi sistemi značajno napredovali, ušli u široku upotrebu na mobilnim telefonima, a u stanju su i da vode dijaloge, daju preporuke relevantne korisniku i slično.

Mašinsko učenje najveće uspehe do sada postiglo je verovatno u obradi slika i video zapisa. Tipični problemi su prepoznavanje i lokalizacija objekata na slikama i video zapisima, praćenje objekata u video zapisima, generisanje realističnih slika i video zapisa visoke rezolucije, automatsko opisivanje slika prirodnim jezikom i slično.

Sveprisutnost društvenih mreža dala je veliki impuls razvoju metoda mašinskog učenja nad grafovima. Društvena mreža može se razmatrati kao graf čiji čvorovi predstavljaju učesnike mreže, a grane postoje između učesnika koji su povezani u mreži (poput prijateljstva na mreži *Facebook*). Metode mašinskog učenja se u ovom kontekstu koriste za predviđanje budućih veza među učesnicima, recimo prilikom preporučivanja učesnicima mreže sa kime se mogu povezati. Razvijene su i metode za otkrivanje postojećih, ali neopaženih veza u društvenim mrežama (koje ne moraju biti samo mreže na internetu, već i u realnom životu). Jedna od motivacija za razvoj ovih metoda je otkrivanje povezanosti u terorističkim i kriminalnim grupama.

**Faze rešavanja problema mašinskim učenjem.** Tök rešavanja problema pomoću mašinskog učenja obično obuhvata sledeće osnovne korake:

- Modelovanje problema;
- Rešavanje problema, odnosno treniranje modela;
- Evaluacija dobijenog rešenja, odnosno modela.

Kod pretrage i automatskog rasudivanja, treći korak može biti važan, ali je nekada i trivijalan ili nepostojeći. Mašinsko učenje uključuje evaluaciju dobijenog rešenja kao suštinski važan korak zbog pomene mogućnosti grešaka. Ukoliko je proces rešavanja dao nezadovoljavajuće rezultate, kroz sve navedene korake prolazi se iznova, primenjujući druge moguće pristupe.

U ostatku teksta, biće dosledno korišćena naredna notacija. Običnim malim slovima poput  $x$  označavaju se skaliari. Podebljanim malim slovima poput  $\mathbf{x}$  označavaju se vektori, a podebljanim velikim poput  $\mathbf{X}$  označavaju se matrice. U kontekstu u kojem se javljaju podebljano slovo i obično slovo sa indeksima, podrazumeva se da obično slovo predstavlja koordinatu vektora ili element matrice označene odgovarajućim podebljanim slovom. Na primer,  $x_i$  će označavati  $i$ -tu koordinatu vektora  $\mathbf{x}$ , a  $x_{ij}$  odgovarajući element matrice  $\mathbf{X}$ . S druge strane  $\mathbf{x}_i$  označava  $i$ -ti vektor  $\mathbf{x}$ , a ne  $i$ -tu koordinatu vektora  $\mathbf{x}$  i slično za matrice.

## 10.1 Modelovanje

Modelovanje problema polazni je deo procesa mašinskog učenja i on često zahteva najviše znanja i inventivnosti. Konkretni koraci variraju od problema do problema, ali postoje neki koraci zajednički za većinu slučajeva i to su:

- Uočavanje opštег okvira učenja adekvatnog za dati problem, koji je pre svega uslovjen vrstom informacije datih u primerima iz kojih se uči;
- Izbor reprezentacije podataka iz kojih se uči;
- Izbor forme modela zakonitosti u podacima, odnosno skupa dopustivih modela;
- Izbor funkcije greške koja meri koliko model odgovara podacima na kojima se vrši trening.

Ovi koraci biće objašnjeni u nastavku.

### 10.1.1 Nadgledano, nenadgledano i učenje potkrepljivanjem

Koliko god primene mašinskog učenja bile raznovrsne, postoje određene zajedničke karakteristike zadataka i procesa učenja koje se često sreću. Postoje tri glavna okvira problema učenja: *nadgledano učenje* (eng. *supervised learning*), *nenadgledano učenje* (eng. *unsupervised learning*) i *učenje potkrepljivanjem* (eng. *reinforcement learning*).

Nadgledano učenje odnosi se na situacije u kojima se algoritmu, zajedno sa podacima iz kojih uči, daju i željeni izlazi, to jest vrednosti takozvane *ciljne promenljive*. Algoritam treba da nauči da za date podatke pruži odgovarajuće izlaze. Očekuje se da izlazi dati za podatke na kojima nije vršeno učenje takođe budu dobri. Neki primeri problema nadgledanog učenja su predviđanje da li je članak računarski ili nije, da li određeno elektronsko pismo predstavlja neželjenu poštu (eng. *spam*) ili ne i predviđanje cene nekih akcija u zavisnosti od kretanja cene tih akcija u prošlosti i kretanja cena drugih akcija.

Nenadgledano učenje odnosi se na situacije u kojima se algoritmu koji uči pružaju samo podaci bez izlaza. Od algoritma koji uči očekuje se da sâm uoči neke zakonitosti u podacima koji su mu dati. Primer nenadgledanog učenja je takozvano klasterovanje – uočavanje grupe na neki način sličnih objekata kada ne postoji prethodno znanje o tome koliko grupe postoji ili koje su njihove karakteristike. Jedan primer primene klasterovanja je redukcija skupa boja slike. Pikseli slike se mogu grupisati klasterovanjem po njihovoј blizini u RGB prostoru boja, a potom se iz svakog klastera može izabrati po jedna boja koja bi ga predstavljala i kojom bi bili obojeni svi pikseli koji pripadaju tom klasteru. Drugi primer primene klasterovanja je grupisanje (prirodnih) jezika.

Učenje potkrepljivanjem odnosi se na situacije u kojima je nizom akcija potrebno postići neki cilj pri čemu u podacima nije poznato koja akcija je bila dobra u kojoj situaciji, već samo finalni ishod čitavog niza akcija. Cilj je naučiti koje su akcije dobre u kojim stanjima. Tipičan primer problema pogodnih za ovu vrstu učenja je igranje igara. Naime, u igrama često nije uvek jasno koji potez je bio dobar (ili ključan), a koji nije, ali je poznato da li je partija na kraju dobijena ili izgubljena.

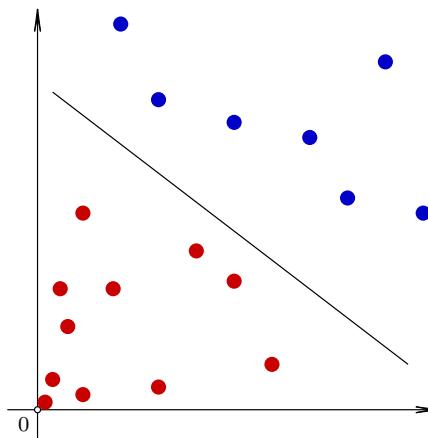
**Primer 10.2.** *Problem prepoznavanja računarskih članaka očito spada u okvir nadgledanog učenja. Naime, postoji konkretna ciljna promenljiva — da li je članak računarski ili nije i nju je potrebno predvideti, i moguće je zajedno sa podacima iz kojih se uči (člancima) pružiti i vrednosti ove ciljne promenljive. Ovaj problem ne uklapa se u druga dva opisana okvira. Nemoguće je očekivati da bi se nenadgledanim pristupom mogli izdiferencirati računarski i ostali članci. Iako nenadgledano učenje često omogućava grupisanje, teško je obezbediti da grupisanje bude izvršeno baš u skladu sa jednim aspektom teme članka — njenom pripadnošću oblasti računarstva. Sasvim je moguće da bi neki postojeći algoritam grupisao članke po nekom drugom kriterijumu, koji najverovatnije ne bi ni bio lako razumljiv čoveku ili dovoljno precisan za potrebe ovog problema. Na kraju, ovaj problem se ne uklapa ni u okvir učenja potkrepljivanjem, pošto rešenje predstavlja jednostavan odgovor da li je članak računarski, a ne niz akcija čijim se zajedničkim efektom postiže neki cilj (kao u igranju igara).*

*Kako je prepoznato da se radi o problemu nadgledanog učenja i da podaci pripadaju dvema klasama, za uspešno rešavanje problema potrebno je obezbediti određeni skup primera koji sadrži kako računarske, tako i članke iz drugih oblasti.*

### 10.1.2 Reprezentacija podataka

Kao što je rečeno na početku, podacima se opisuju konkretni primerci, tj. *instance* nekog problema. Primera radi, iako možemo govoriti o opštem problemu prepoznavanja računarskih članaka, u praksi se uvek susrećemo sa njegovim konkretnim primercima – uvek je za neke konkretnе članke potrebno ustanoviti da li su računarski ili nisu. Otuda se, kada se govorи о podacima, često govorи о instancama podataka pri čemu se podrazumeva da jedna instanca podataka opisuje jedan konkretni primerak problema koji treba rešiti.

Instance koje čine podatke treba da budu zapisane u obliku koji je pogodan za primenu algoritama učenja. Najpogodniji i najčešće korišćeni način koji se koristi u algoritmima mašinskog učenja je predstavljanje instanci pomoću nekih njihovih relevantnih *svojstava*, tj. *atributa odlika* ili *obeležja* (eng. *feature, attribute*). Svojstva ili atributi predstavljaju karakteristike instanci kao što su boja, veličina, težina i slično. Svako od izabralih svojstava može imati vrednost koja pripada nekom unapred zadatom skupu. Te vrednosti su nekad numeričke, kao u slučaju težine, koja je skalarna veličina i koja se najbolje opisuje brojem. Primer numeričke vrednosti može biti i frekvencija reči u nekom članku, kao što je to bio slučaj u primeru sa klasifikacijom članaka. Svojstva takođe mogu biti i kategorička — mogu predstavljati imena nekih kategorija kojima se ne mogu dodeliti smislene numeričke vrednosti ili pridružiti uređenje. Primer kategoričkog svojstva može biti grad u kome osoba živi, pol, nacionalnost i slično. Skup svojstava koji će se koristiti u zapisu instance generalno nije unapred zadat, već ga je potrebno odabratи u skladu sa time koje su karakteristike instanci bitne za dati problem učenja. Na primer, za predviđanje ocene na ispitу nije bitna boja očiju studenta. Kada su izabrana svostva pomoću kojih se instance



Slika 10.2: Osim krugova koji predstavljaju članke, prikazana je i prava koja ih razdvaja.

opisuju, svaka instanca može se predstaviti vektorom vrednosti svojstava koja joj odgovaraju.

S druge strane, podaci se nekada mogu dati algoritmu učenja i u sirovoj formi, poput matrice piksela koji čine sliku bez opisivanja slike nekim specifičnim svojstvima. Nisu svi algoritmi učenja dizajnirani da dobro rade sa takvim reprezentacijama.

**Primer 10.3.** U slučaju prepoznavanja računarskih članaka, instanca je jedan članak. Instance će biti u računaru predstavljene pomoću nekih podataka koji ih opisuju. Ciljna promenljiva  $y$ , koja predstavlja klasu može biti 1 za računarske članke i -1 za ostale. Iako je predstavljena brojem, ovo je kategorička vrednost, pošto se radi o dve kategorije za koje su ovi brojevi proizvoljno izabrani.

Da bi bilo sprovedeno učenje, potrebno je raspoložive članke predstaviti u nekom obliku koji je pogodan za algoritam učenja i koji bi mogao da na neki način sažme osnovne karakteristike na osnovu kojih se članci iz ove dve kategorije mogu razlikovati. Kao što je ranije zapaženo, očekivano je da će u člancima iz računarstva biti češće pominjani računarski pojmovi nego u ostalim člancima. To svojstvo bi se moglo iskoristiti za razlikovanje članaka. U skladu sa ovim, mogu se nabrojati sve reči iz nekog rečnika računarske terminologije. Svaki članak može biti predstavljen vektorom frekvencija ovih reči (frekvencija neke reči u članku se računa tako što se broj pojavljivanja te reči podeli ukupnim brojem pojavljivanja svih reči u članku). Ako je  $\mathbf{x}$  vektor koji odgovara nekom članku, onda će  $x_i$  označavati frekvencije izabranih pojedinačnih reči.

Opisani vektori frekvencija predstavljaju tačke u euklidiskom prostoru. Kao što je diskutovano ranije, ako se u pojednostavljenom rečniku nalaze samo dve reči — „računar“ i „datoteka“, ove tačke mogu izgledati kao na slici 10.1. Ukoliko su u člancima iz jedne kategorije ovi računarski termini visokofrekventni, a u drugim niskofrekventni, tačke koje odgovaraju računarskim člancima će se grupisati dalje od koordinatnog početka, dok će se ostale grupisati bliže njemu.

### 10.1.3 Skup dopustivih modela

Proces učenja može se razmatrati kao proces pronalaženja zakonitosti u podacima ili, preciznije, zavisnosti među promenljivim koje ih opisuju, a koje čine svojstva i ciljna promenljiva. Kako bi se učenje moglo automatizovati, potrebno je da opšta forma tih zavisnosti bude matematički definisana. Matematičke reprezentacije zavisnosti među promenljivim nazivamo *modelima*. Ovaj pojam vrlo je blizak pojmu modela u empirijskim naukama, koji takođe ustanavljava zavisnosti između veličina koje su relevantne za proučavani fenomen (na primer, zavisnosti između brzine, puta i vremena).

Obično modeli koji se razmatraju imaju unapred određenu opštu formu i moguće je uočiti *skup dopustivih modela*. Na primer, u primeru prepoznavanja računarskih članaka, kao skup dopustivih, mogućih modela koristi se skup pravih  $\{w_1x_1 + w_2x_2 + w_3 \mid w_1, w_2, w_3 \in \mathbb{R}\}$ . Forme modela mogu biti raznovrsne: modeli mogu biti pravila oblika IF . . . THEN, linearne funkcije svostava, nelinearne funkcije svojstava i tako dalje.

**Primer 10.4.** U slučaju prepoznavanja računarskih članaka, prepostavimo da se članci prepoznaju na osnovu frekvencija dve reči. Između dve grupe tačaka možda postoji prava koja ih razdvaja, kao na slici 10.2. Ako je ova prava poznata, onda neki nov, nepoznati članak može biti prepoznat kao članak iz oblasti

računarstva ukoliko se tačka koja mu odgovara nalazi sa iste strane prave kao i vektori računarskih članaka koji su nam poznati. U suprotnom, smatra se da članak nije iz oblasti računarstva. U ovom slučaju dobar izbor modela je funkcija

$$w_1x_1 + w_2x_2 + w_3$$

na osnovu čijeg znaka se odlučuje o tome kojoj klasi pripada članak.

Izbor skupa dopustivih modela od fundamentalnog je značaja za kvalitet učenja. Ukoliko ovaj skup nije dovoljno bogat, onda učenje, jasno, ne može biti dovoljno dobro. Naizgled paradoksalno, i preterano bogatstvo skupa dopustivih modela može da dovede (i često dovodi) do loših rezultata. Ovaj fenomen biće diskutovan u delu 11.3.

#### 10.1.4 Funkcija greške

U mašinskom učenju greška je nužno prisutna, ali je i nepoželjna. Stoga je potrebno meriti kolike su greške koje model pravi. Tome služe funkcije greške koje se razlikuju od problema do problema.

U slučaju nadgledanog učenja, ova funkcija treba da kvantifikuje odstupanje predviđene od stvarne vrednosti ciljne promenljive za neku instancu. U slučaju nenadgledanog učenja, ako je potrebno, na primer, izvršiti identifikaciju grupa u podacima, greška treba da kvantifikuje raznolikost podataka unutar grupa, pošto je poželjno da podaci unutar jedne grupe budu što sličniji. U slučaju učenja potkrepljivanjem, obično se govori o nagradi umesto o grešci, a nagrada je utoliko veća ukoliko je posao bolje obavljen nizom akcija koje se preduzimaju. Funkcija greške se definiše za jednu instancu, ali je poželjno da model dobijen treningom pravi malu ukupnu grešku, koja se obično definiše kao suma ili prosek grešaka na pojedinačniminstancama.

Funkcije greške obično se biraju tako da pored smislenosti u odnosu na problem imaju i neka poželjna svojstva poput neprekidnosti, diferencijabilnosti, konveksnosti i slično.

**Primer 10.5.** U primeru prepoznavanja računarskih članaka, mogući su različiti izbori funkcije greške. Jedan izbor bi mogao biti 1 ako je instance loše klasifikovana, a 0 ako je dobro klasifikovana, ali ta funkcija nema dobra tehnička svojstva i otud se ne koristi. Drugi prirodan izbor bi se mogao voditi idejom da je greška utoliko veća ukoliko je instance dalje od prave koja razdvaja klase i sa pogrešne njene strane. Ukoliko je tačka sa ispravne strane („ispravne“ u smislu ovog modela), znak funkcije

$$w_1x_1 + w_2x_2 + w_3$$

se poklapa sa znakom ciljne promenljive  $y$  (1 ili -1) i proizvod  $y \cdot (w_1x_1 + w_2x_2 + w_3)$  je pozitivan. Ukoliko je tačka sa pogrešne strane, taj proizvod je negativan i utoliko veći što je tačka dalje od prave na pogrešnoj strani. Otud je greška, kada postoji, proporcionalna izrazu

$$-y \cdot (w_1x_1 + w_2x_2 + w_3)$$

Kako bi se izbegla negativna greška u slučaju tačne klasifikacije, ovaj izraz može se zameniti sledećim

$$\max(0, -y \cdot (w_1x_1 + w_2x_2 + w_3))$$

tako da je u slučaju tačne klasifikacije greška jednaka nuli. Onda prava koja ispravno razdvaja sve članke pravi grešku 0 na svim člancima. Dodatno, ova funkcija je neprekidna, konveksna i diferencijabilna svugde osim u jednoj tački.

Ponovimo da su oznake klase 1 i -1 zapravo kategoričke vrednosti. Ovakav izbor vrednosti daje tehničku pogodnost i za njega ne postoji drugi razlozi.

## 10.2 Rešavanje

Rešavanje problema u mašinskom učenju svodi se na pronalaženje modela koji pravi najmanju grešku (prema izabranom kriterijumu) na datim podacima. To rešavanje se onda može razumeti i kao pretraga kroz skup dopustivih modela koja je vođena podacima, a koju realizuje algoritam učenja. Za taj algoritam mogući su i drugačiji izbori, ali on se često svodi na neki optimizacioni metod poput gradijentnog spusta nad funkcijom greške definisanom prilikom modelovanja. Proces pronalaženja adekvatnog modela naziva se *treningom*.

**Algoritam:** Algoritam za klasifikaciju članaka

**Ulaz:** Trening skup  $T$ , brzina učenja  $\eta$  i preciznost  $\varepsilon$

**Izlaz:** Koeficijenti modela  $\mathbf{w} = (w_1, w_2, w_3)$

- 1: postavi  $\mathbf{w}$  na  $(0, 0, 0)$ ;
- 2: **ponavljam**
- 3:     postavi  $\mathbf{w}'$  na  $\mathbf{w}$ ;
- 4:     **za** svaku instancu  $(x_1, x_2, y) \in T$  **radi**
- 5:         **ako**  $y(w_1x_1 + w_2x_2 + w_3) \leq 0$  **onda**
- 6:             na  $w_1$  dodaj  $\eta yx_1$ ;
- 7:             na  $w_2$  dodaj  $\eta yx_2$ ;
- 8:             na  $w_3$  dodaj  $\eta y$ ;
- 9: **dok nije ispunjen** uslov  $\|\mathbf{w} - \mathbf{w}'\| \leq \varepsilon$ ;
- 10: **vrati**  $\mathbf{w}$  kao rešenje.

Slika 10.3: Algoritam za klasifikaciju članaka.

**Primer 10.6.** U primeru prepoznavanja računarskih članaka, pronađenje željene prave može se izvesti „pomeranjem“ neke polazne prave dok ona ne bude pozicionirana između tačaka koje treba da razdvaja, odnosno dok greška ne postane jednaka nuli ili, ako je to nemoguće, što manja moguća. Ovo pomeranje vrši se postepenim modifikacijama koeficijenata  $w_1, w_2$  i  $w_3$ . Jedan takav algoritam dat je na slici 10.3.

Važno pitanje je da li ažuriranje koeficijenata u predloženom algoritmu vodi poboljšanju naučene funkcije (tj. tekućeg modela). Pre svega, algoritam ne menja vrednosti parametara  $w$  u slučaju da je instanca tačno klasifikovana, već samo ako nije, što deluje dobro. Dalje, vrednost  $\eta$ , koja kontroliše veličinu korekcije ( $\eta yx_1, \eta yx_2, \eta y$ ), mora biti mala kako bi korekcije bile male i postepene. Vrednosti  $x_1$  i  $x_2$  su uvek nenegativne i stoga znak korekcije zavisi samo od znaka vrednosti  $y$ . Ukoliko je vrednost  $y$  negativna, koeficijenti se smanjuju čime se i vrednost  $w_1x_1 + w_2x_2 + w_3$  smanjuje. Ako pritom ta vrednost postane negativna, greška na toj instanci je eliminisana. Analogno u slučaju da je vrednost  $y$  pozitivna. Stoga, naslućujemo da se ovim algoritmom vrednosti  $w_1x_1 + w_2x_2 + w_3$  po znaku približavaju vrednostima  $y$ . Korekcije su proporcionalne vrednostima  $x_1$  i  $x_2$ , odnosno veće su za koeficijente čija promena može više doprineti promeni vrednosti  $w_1x_1 + w_2x_2 + w_3$  za dati primer. Ovaj postupak liči na gradijentni spust kojim se minimizuje ranije definisana funkcija greške:

$$\max(0, -y \cdot (w_1x_1 + w_2x_2 + w_3)).$$

### 10.3 Evaluacija

Učenje uvek polazi od nekih podataka. Podaci na osnovu kojih se vrši generalizacija, nazivaju se *podacima za trening*, a njihov skup *trening skup*. Evaluacija naučenog znanja na podacima na osnovu kojih je učeno obično dovodi do značajno boljih rezultata od onih koji se mogu kasnije dobiti u primenama, što ne bi trebalo da iznenađuje. Naime, i u slučaju ljudskog učenja, lakše je rešavati probleme koji su već poznati u procesu učenja, nego potpuno nove probleme. Stoga je pre upotrebe potrebno proceniti kvalitet naučenog znanja na novim, nepoznatim podacima. To se obično radi tako što se razmatra koliko je naučeno znanje kvalitetno u odnosu na neke unapred date *podatke za testiranje* čija je uloga da u evaluaciji simuliraju (nepoznate) podatke na kojima će model biti primenjivan u budućnosti. Podaci za testiranje čine *test skup*. Test skup treba da bude disjunktan sa trening skupom. Kvalitet modela na test skupu može se meriti računanjem različitih statistika čija je uloga da što bolje opišu poklapanje predviđanja modela sa stvarnim podacima. U praksi, trening i test skup se dobijaju tako što se raspoloživi podaci podele na dva dela. Naravno, kako različite podele na trening i test skup mogu urodit različitim rezultatima, slučajno deljenje nije najbolji način formiranja trening i test skupa, osim u slučaju ogromne količine podataka i mogući su i drugi načini evaluacije, o kojima će biti reči kasnije.

**Primer 10.7.** U primeru prepoznavanja računarskih članaka, pre treninga mogli smo odvojiti, na primer,

*trećinu ukupnih raspoloživih podataka za testiranje, a trening sproveduti nad preostale dve trećine. Kad su članci klasifikovani, kao mera kvaliteta učenja može se na test skupu izračunati udeo dobro klasifikovanih članaka u ukupnom broju članaka. Poželjno je da on bude što veći.*

Elektronsko izdanie (2020)

## Glava 11

# Nadgledano mašinsko učenje

Nadgledano mašinsko učenje karakteriše se time da su za sve unapred raspoložive podatke poznate vrednosti ciljne promenljive. U nastavku su opisani osnovni problemi nadgledanog učenja – klasifikacija i regresija, potom pojmovi zajednički za većinu algoritama nadgledanog mašinskog učenja i, na kraju, konkretni modeli nadgledanog učenja i algoritmi za njihovo treniranje.

### 11.1 Klasifikacija i regresija

Problemi nadgledanog mašinskog učenja najčešće se mogu svrstati u jednu od dve kategorije – u probleme klasifikacije ili u probleme regresije.

Klasifikacija se sastoji u razvrstavanju nepoznate instance u jednu od unapred ponuđenih kategorija tj. klase. Neki od primera klasifikacije su razvrstavanje bankovnih transakcija u rizične koje mogu predstavljati prevaru ili u nerizične koje predstavljaju uobičajene transakcije, određivanje autorstva tekstova pri čemu se tekstu nepoznatog autora pridružuje jedan od nekoliko unapred ponuđenih autora, razvrstavanje elektronske pošte u željenu i neželjenu i slično. U navedenim primerima, svakoj instanci (bankovna transakcija, tekst, elektronska poruka) odgovara vektor vrednosti nekih izabranih svojstava. Svakoj instanci takođe odgovara i oznaka klase kojoj ta instanca pripada. Problem klasifikacije sastoji se onda u određivanju vrednosti klase na osnovu preostalih svojstava instance. Ključno zapažanje je da je ciljna promenljiva u ovom problemu diskretna i da se, u opštem slučaju, oznakama klasa ne mogu smisleno dodeliti numeričke vrednosti niti uređenje. Dakle, klasa, čiju je vrednost potrebno odrediti, kategoričko je svojstvo.

Problem regresije predstavlja problem u kojem pored vrednosti svojstava svakoj instanci odgovara i neka neprekidna numerička vrednost koju je potrebno predvideti. Primene regresije su mnogobrojne. One uključuju procenu rizika u finansijskim ulaganjima, modelovanje zagađenja životne sredine, procenu smrtnosti u zavisnosti od životnih navika, određivanje pozicija objekata na slikama i slično.

### 11.2 Minimizacija greške

Mnogi algoritmi klasifikacije i regresije zasnivaju se na minimizaciji greške. Čak i kada ona ne figuriše eksplicitno u formulaciji, često se može uočiti pažljivom analizom. Diskusija u nastavku prepostavlja eksplicitno formulisanu funkciju greške.

Kao što je već rečeno, zavisnosti između promenljivih u mašinskom učenju opisuju se parametrizovanim funkcijama  $f_{\mathbf{w}}(\mathbf{x})$  koje se nazivaju modelima. Već pominjani primer su linearne funkcije. Vektor parametara će se nadalje uvek označavati  $\mathbf{w}$  i obično prepostavljamo da je njime model potpuno određen.

Kako modeli mašinskog učenja mogu da greše, potrebno je izabrati *funkciju gubitka* čija je uloga da kvantificuje odstupanje koje model pravi u odnosu na tačnu vrednost ciljne promenljive. Na primer, kvadrat razlike predviđene realne vrednosti  $f_{\mathbf{w}}(\mathbf{x})$  i ciljne vrednosti  $y$

$$L(f_{\mathbf{w}}(\mathbf{x}), y) = (f_{\mathbf{w}}(\mathbf{x}) - y)^2$$

predstavlja čest izbor za funkciju gubitka. Naravno, za njenu primenu potrebno je da su nad vrednostima ciljne promenljive definisane aritmetičke operacije, što ne mora da važi u opštem slučaju. Ipak, tamo gde je primenljiva, ova funkcija ponaša se intuitivno — velike razlike između predviđene i stvarne vrednosti proizvode veliku vrednost gubitka, a takođe je i matematički pogodna zbog svoje diferencijabilnosti. Zbog toga predstavlja čest izbor funkcije gubitka, mada nije najbolji izbor ni uvek kada je primenljiva.

Kada su definisani forma modela i funkcija gubitka, nije teško formulisati kriterijume za izbor najboljeg modela — to je model koji pravi najmanju grešku, to jest, pravi najmanji očekivani gubitak na podacima na kojima će biti korišćen. Dakle, potrebno je rešiti sledeći problem minimizacije:

$$\min_{\mathbf{w}} \mathbb{E}_{(\mathbf{x},y)} L(f_{\mathbf{w}}(\mathbf{x}), y)$$

gde je  $\mathbb{E}_{(\mathbf{x},y)}$  matematičko očekivanje po svojstvima i ciljnoj promenljivoj koje se naziva *rizikom*. Za izračunavanje matematičkog očekivanja  $\mathbb{E}_{(\mathbf{x},y)}$ , potrebno je poznavati raspodelu promenljivih po kojima se očekivanje računa. U ovom slučaju, radi se o zajedničkoj raspodeli promenljivih  $\mathbf{x}$  i  $y$ , koja obično nije poznata. Zato ovaj kriterijum nije lako upotrebiti. Međutim, ono što u praksi jeste poznato jeste uzorak podataka iz trening skupa  $\{(\mathbf{x}_i, y_i) \mid i = 1, \dots, N\}$ . Zbog toga se očekivanje funkcije gubitka aproksimira njenim uzoračkim prosekom koji se naziva *empirijskim rizikom* ili *greškom modela*:

$$E(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N L(f_{\mathbf{w}}(\mathbf{x}_i), y_i)$$

**Primer 11.1.** U primeru klasifikacije članaka, greška modela data je funkcijom:

$$E(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \max(0, -y \cdot (w_1 x_1 + w_2 x_2 + w_3))$$

Može se dokazati da, pod određenim uslovima, vrednosti parametara  $\mathbf{w}$  dobijene minimizacijom greške modela dobro aproksimiraju vrednosti parametara koje bi bile dobijene minimizacijom (stvarnog) rizika. Stoga se vrednosti parametara  $\mathbf{w}$  i biraju minimizacijom vrednosti greške modela. Kako prosek i suma imaju isti minimum, obično se prilikom minimizacije ne vodi računa o tome da li je izvršeno deljenje brojem instanci. To će biti vidljivo i u minimizacionim problemima u nastavku.

Prethodno izlaganje pokazuje da je potrebno minimizovati grešku modela. Za to se koriste metode matematičke optimizacije. Jedna od klasičnih metoda korišćenih u ovom kontekstu je gradijentni spust (poglavlje 4.1). Taj metod zahteva da je funkcija diferencijabilna. Postoje mnoge druge optimizacione metode koje mogu biti pogodnije za optimizacioni problem koji se razmatra. Često se za novi problem formuliše i nova metoda optimizacije koja je posebno pogodna za njega. Temeljnije upućivanje u metode matematičke optimizacije izlazi iz okvira ove knjige.

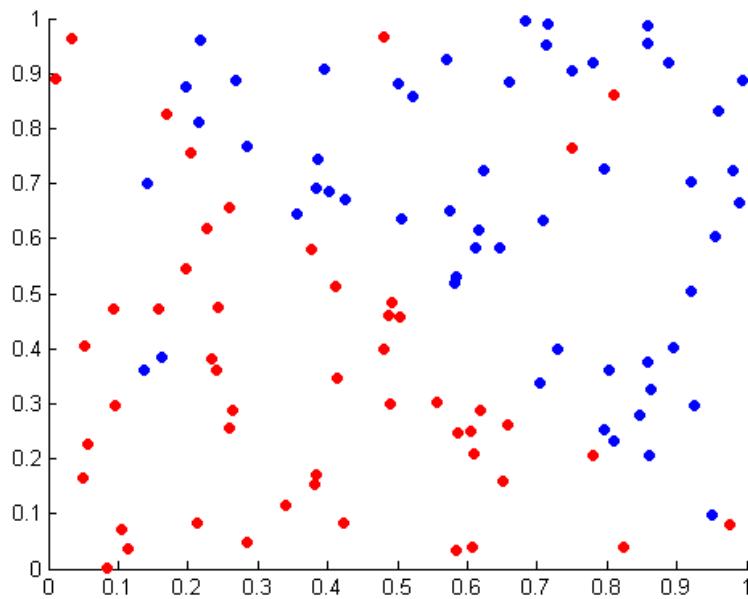
### 11.3 Preprilagođavanje i regularizacija

Očigledno, što je greška modela manja, to je prilagođenost modela podacima za trening veća i obratno. Dakle, vrednost greške modela može da ima ulogu mere prilagođenosti modela podacima za trening.

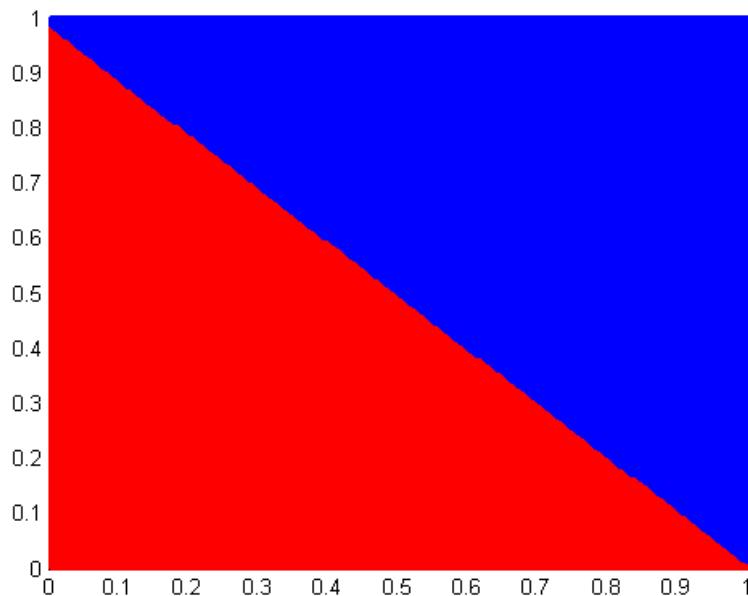
Uslovi pod kojima se minimizacijom greške dobro aproksimira polazni problem nisu uvek ispunjeni, pa prethodno opisani postupak minimizacije i izbora modela ne vodi nužno dobrim rezultatima učenja. Razmatranje ovog problema predstavlja najplodniji aspekt teorije mašinskog učenja i pruža najdublje uvide u prirodu procesa generalizacije, ali prevazilazi okvire ove knjige. Ipak, osnovni zaključak može se kratko formulisati: osnovna prepreka dobroj aproksimaciji optimalnih vrednosti parametara, a time i uspešnoj generalizaciji, je preterano bogatstvo skupa dopustivih modela. Ukoliko je taj skup toliko bogat da u njemu za svaki trening skup postoji model koji ga savršeno opisuje, onda ne postoje garancije za uspešno učenje. Dodatno, što je skup dopustivih modela bogatiji, to je potrebno više podataka za uspešno učenje. Ilustrovaćemo ovaj uvid kroz dva primera, prvi koji se odnosi na klasifikaciju i drugi koji se odnosi na regresiju.

**Primer 11.2.** Neka je dat trening skup instanci koje predstavljaju članke, od kojih su neki računarski, a neki ne. Taj skup je prikazan na slici 11.1. U tom skupu postoje i neki računarski članci sa niskom frekvencijom reči iz specifično računarske terminologije, ali i neki članci koji nisu računarski, a ipak imaju visoku frekvenciju računarskih termina. U praksi je česta situacija da, iz različitih razloga, određeni broj instanci odstupa od očekivanja. Takvih primera obično nema mnogo.

Prepostavimo da je forma modela linearne, kao i do sada —  $f_{\mathbf{w}}(\mathbf{x}) = w_1 x_1 + w_2 x_2 + w_3$  i da su koeficijenti  $\mathbf{w}$  određeni minimizacijom greške pri čemu je za funkciju gubitka korišćen kvadrat razlike ciljne i predviđene vrednosti (u ovom konkretnom problemu postoje i pogodnije funkcije gubitka, ali će za potrebe ovog primera i kvadrat razlike biti dovoljno dobra funkcija). Slika 11.2 ilustruje rešenje u tom slučaju. Može se primetiti da model nije saglasan sa svim instancama, odnosno da postoje računarski članci koji



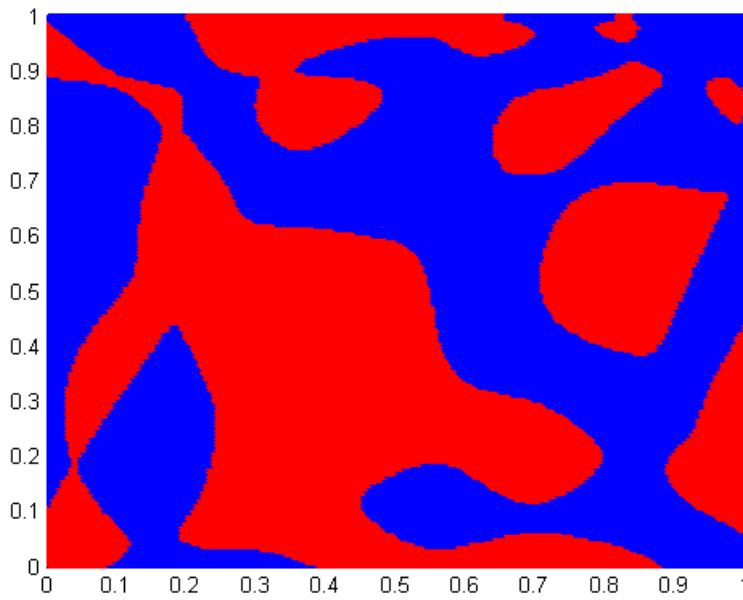
Slika 11.1: Trening skup. Plave tačke označavaju računarske, a crvene ostale članke.



Slika 11.2: Prikaz linearog modela minimalne greške. Tačke u ravni za koje model daje pozitivnu vrednost označene su plavo, a tačke za koje daje negativnu vrednost označene su crveno.

*nisu prepoznati i članci koji su prepoznati kao računarski, a to nisu. Međutim, to ne bi trebalo da bude zabrinjavajuće, pošto je za većinu članaka klasifikacija ispravna. Članci koji su pogrešno klasifikovani odstupaju od trenda učestalog korišćenja reči računar i datoteka u takvim člancima, ali ih nema dovoljno da bi sugerisali da i drugi članci u kojima se te reči retko koriste treba da budu klasifikovani kao računarski.*

*Slika 11.3 prikazuje klasifikaciju datih podataka korišćenjem modela iz skupa svih polinoma dve promen-*



Slika 11.3: Polinomski model minimalne greške. Tačke u ravni za koje model daje pozitivnu vrednost označene su plavo, a tačke za koje daje negativnu vrednost, označene su crveno.

ljive proizvoljnog stepena, odnosno u slučaju da se za modele koristi sledeća forma:

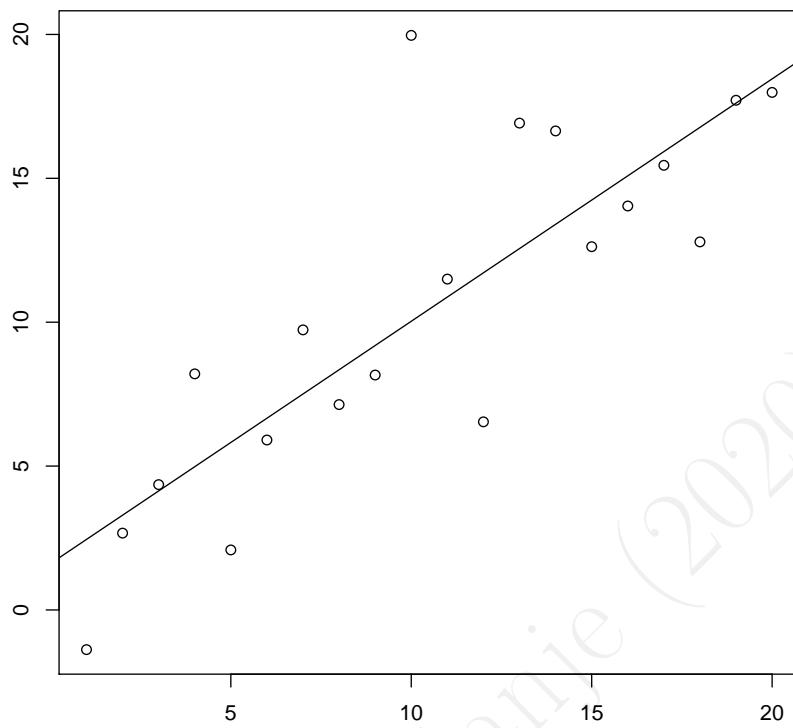
$$f_{\mathbf{w}}(\mathbf{x}) = \sum_{i=0}^n \sum_{j=0}^i w_{ij} x_1^j x_2^{i-j}$$

dok je funkcija gubitka ista. Izabrani model je saglasan sa svim instancama iz trening skupa i stoga je greška jednaka nuli. Međutim, zakonitost koju on opisuje ne izgleda uverljivo. Naime, intuitivno je da su računarski članci koji ne sadrže računarske termine redak izuzetak, a ne da postoje velike oblasti prostora svojstava koje odgovaraju niskim frekvencijama računarskih termina, a ipak se odnose na računarske članke. Vredi primetiti i da se u oblasti za koju bi se očekivalo da je plava, nalazi veliki potprostor obojen crvenom bojom, a u kojem među treninginstancama ne postoji nijedna crvena tačka. Ovakve „zakonitosti“ čine korišćenje ovakvog modela u predikciji potpuno nepouzdanim i sigurno je da je stvarni rizik daleko veći od nule.

**Primer 11.3.** Prepostavimo da je dat trening skup od 20 instanci koje se sastoje od jednog svojstva i vrednosti ciljne promenljive. Prepostavimo da je forma razmatranih modela linearna —  $f_{\mathbf{w}}(\mathbf{x}) = w_1 x + w_2$  i da su koeficijenti  $\mathbf{w}$  određeni minimizacijom greške, pri čemu je za funkciju gubitka korišćen kvadrat razlike ciljne i predviđene vrednosti. Slika 11.4 ilustruje takav pristup. Može se primetiti da model nije u potpunosti saglasan ni sa jednom instancom, odnosno za svaku trening instancu postoji manja ili veća greška u predviđanju. Dakle, jednostavan linearни model nije dovoljno fleksibilan da se može potpuno prilagoditi podacima za trening. S druge strane, očigledno je da on dobro opisuje opšti linearni trend koji u podacima postoji i, posebno važno, za očekivati je da greška na novim podacima iz iste raspodele bude približno jednaka grešci na poznatim, trening podacima.

Slika 11.5 prikazuje aproksimaciju datih podataka korišćenjem modela iz skupa svih polinoma proizvoljnog stepena, odnosno u slučaju da se za modele koristi forma  $f_{\mathbf{w}}(\mathbf{x}) = \sum_{i=0}^n w_i x^i$ . Izabrani model saglasan je sa svim instancama iz trening skupa i stoga je greška modela jednaka nuli. Međutim, posmatrajući globalni izgled izabranog modela, vidi se da on zapravo ne opisuje nikakvu zakonitost u podacima. Oscilacije koje pravi između tačaka čine njegovo korišćenje u budućem predviđanju potpuno nepouzdanim i sigurno je da je stvarni rizik daleko veći od nule.

Problem koji se u prethodnim primerima javlja proistiće upravo iz toga što skup svih polinoma čini previše bogat skup mogućih modela. Za svaki trening skup može se naći model koji ga savršeno opisuje. Međutim,



Slika 11.4: Linearni model telesne mase minimalne greške.

prilagođavajući se trening podacima do krajnosti, gubi se svaka moć generalizacije. Takav zaključak važi i za druge previše bogate skupove dopustivih modela, a ne samo za polinome. Ilustrovani fenomen naziva se *preprilagođavanje* (eng. *overfitting*) i predstavlja glavnu opasnost u mašinskom učenju.

U svetlu prethodnog zaključka, teži se ograničavanju bogatstva skupa dopustivih modela, što se može postići smanjenjem fleksibilnosti forme modela. Primera radi, linearna forma modela sa ograničenim brojem koeficijenata može se smatrati nefleksibilnom.

Zanimljivo je da za smanjenje fleksibilnosti modela nije neophodno unapred izabrati skup dopustivih modela tako da bude siromašan, već je dovoljno modifikovati funkciju koja se minimizuje tako da veliki broj modela ima visoku vrednost te funkcije. Često korišćen i sistematičan način da se to postigne je postupak *regularizacije*, koji može biti sproveden na različite načine a ovde će biti prikazan najstariji i najuobičajeniji. Umesto minimizacije greške, vrši se minimizacija regularizovane greške, odnosno, rešava se problem

$$\min_{\mathbf{w}} E(\mathbf{w}) + \lambda \Omega(\mathbf{w})$$

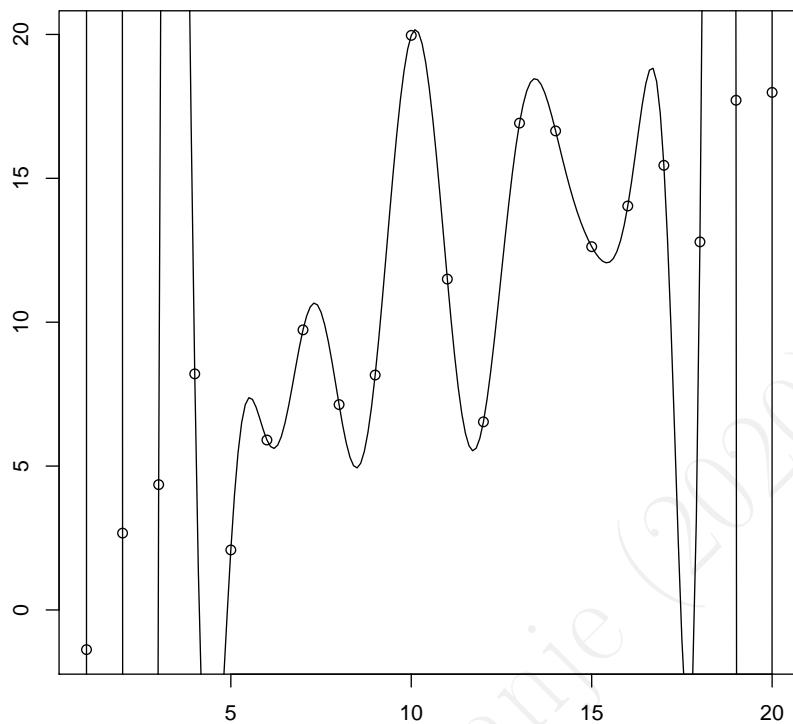
gde je  $\Omega(\mathbf{w})$  *regularizacioni izraz* i pri čemu važi  $\lambda \geq 0$ . Regularizacioni izrazi obično su zasnovani na normama, pa su uobičajeni izbori poput

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_{i=1}^n w_i^2$$

ili

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_{i=1}^n |w_i|$$

ali se koriste i mnogi drugi. Minimizacija greške, koja meri prilagođenost modela podacima, zahteva odstupanje koeficijenata  $w_i$  od nule. Međutim, dodavanjem regularizacionog izraza, takvo odstupanje kažnjava se utoliko više što je odstupanje veće. Time se otežava preveliko prilagođavanje modela podacima, tj. fleksibilnost modela se smanjuje. Mera u kojoj regularizacioni izraz umanjuje fleksibilnost modela kontroliše se izborom *hiperparametara*.



Slika 11.5: Polinomski model minimalne greške.

metra  $\lambda$ .<sup>1</sup> Visoke vrednosti nisu poželjne ni za ovaj parametar jer se nefleksibilni modeli koji se time dobijaju ne mogu dovoljno prilagoditi podacima, pa je kvalitet učenja u tom slučaju loš.

**Primer 11.4.** Neka se u primeru klasifikacije članaka koristi forma modela koja odgovara polinomu dve promenljive, kao funkcija gubitka neka se koristi kvadrat razlike ciljne i predviđene vrednosti i neka se koristi regularizacija. Tada je potrebno rešiti sledeći problem minimizacije:

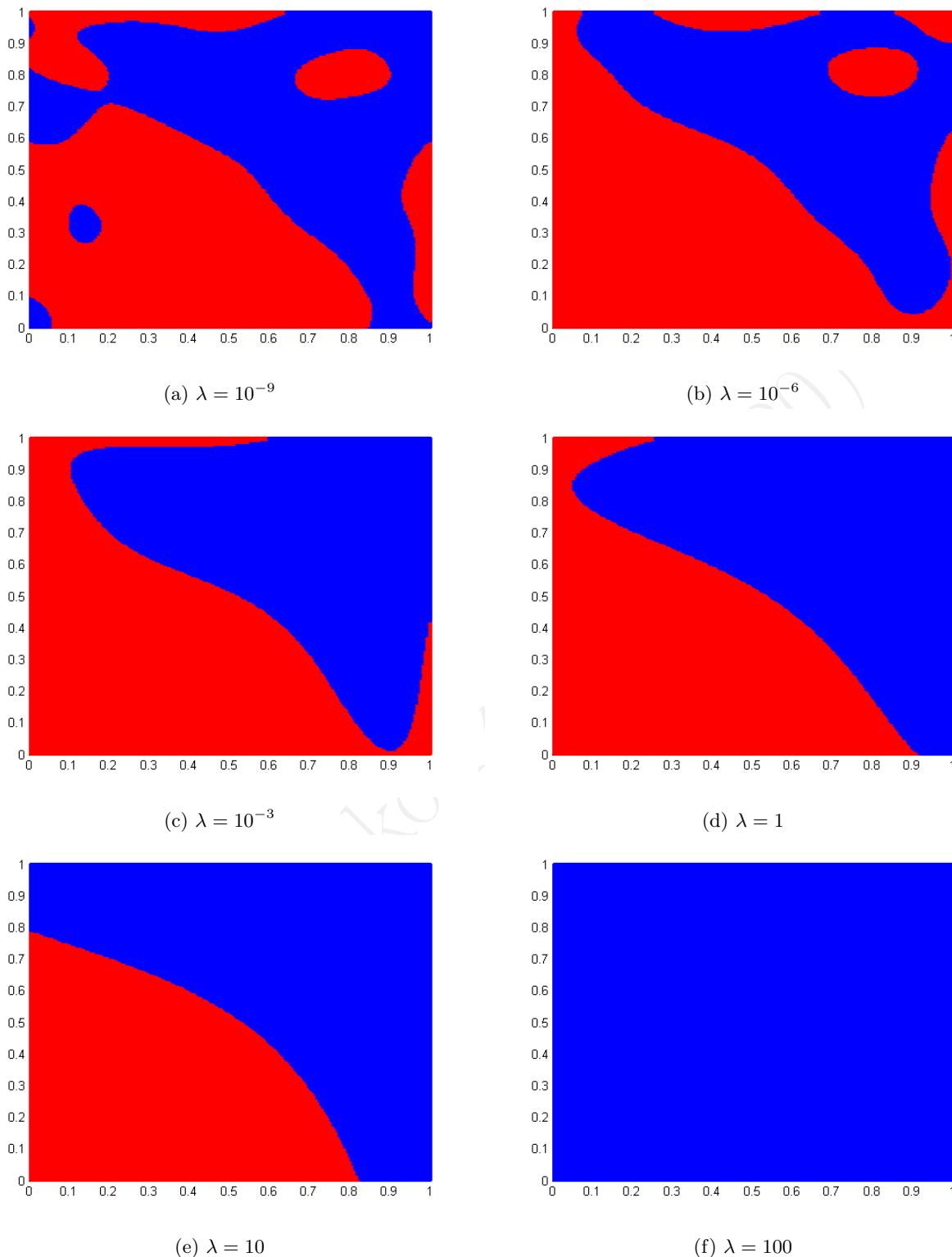
$$\min_{\mathbf{w}} \sum_{i=1}^N \left( \sum_{j=0}^n \sum_{k=0}^j w_{jk} x_{i1}^k x_{i2}^{j-k} - y \right)^2 + \lambda \sum_{j=0}^n \sum_{k=0}^j w_{jk}^2$$

Za vrednosti regularizacionog hiperparametra  $\lambda = 10^{-9}, 10^{-6}, 10^{-3}, 1, 10, 100$ , dobijaju se modeli prikazani na slici 11.6. Očigledno je da povećavanje regularizacionog hiperparametra  $\lambda$  smanjuje mogućnost preprilagođavanja modela, ali i da njegovo preterano povećavanje vodi njegovoj potpunoj neprilagodljivosti, usled čega, za vrednost  $\lambda = 100$ , svi članci bivaju klasifikovani kao računarski samo zato što ih u trening skupu ima više.

Na slici 11.7 prikazane su tri krive koje ilustruju uobičajeno ponašanje modela u zavisnosti od vrednosti hiperparametra  $\lambda$ . Jedna, rastuća, predstavlja grešku na trening skupu u zavisnosti od vrednosti hiperparametra  $\lambda$ . Kako ta greška predstavlja ocenu rizika, riziku se može pridružiti interval poverenja<sup>2</sup> u odnosu na tu ocenu. Širina intervala poverenja predstavljena je drugom, opadajućom, krivom. U slučaju visoke greške na trening podacima, na osnovu uskog intervala poverenja, možemo biti relativno sigurni da će i rizik biti visok. U slučaju vrlo niske vrednosti greške, na osnovu širokog intervala poverenja, nemamo nikakve garancije da će i rizik biti nizak. Treća kriva je zbir prethodne dve i predstavlja gornju granicu rizika. Kao što je već rečeno, i premale i

<sup>1</sup>Terminom *hiperparametar* pravi se razlika između parametara koji se uče poput parametara  $\mathbf{w}$  i parametara koji učestvuju u definiciji optimizacionog problema, poput parametra  $\lambda$ .

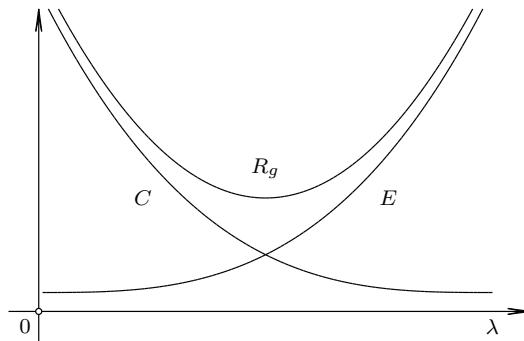
<sup>2</sup>Za ovu diskusiju nije bitna konkretna verovatnoća pridružena intervalu poverenja (na primer, 95% ili 90%).



Slika 11.6: Polinomski modeli dobijeni za različite vrednosti regularizacionog hiperparametra.

prevelike vrednosti hiperparametra  $\lambda$ , koji kontroliše fleksibilnost modela, vode lošim rezultatima. Prve usled preprilagođavanja, a druge zbog nefleksibilnosti. Više o načinu na koji se vrednost ovog parametra može birati u praksi biće reči kasnije.

Pored opisanog načina regularizacije postoje i drugi. Termin *regularizacija* često se koristi i slobodnije tako da obuhvata i druge modifikacije optimizacionog problema kojima se kontroliše fleksibilnost modela, čime se omogućava izbor modela koji nije preprilagođen i dobro generalizuje.



Slika 11.7: Ponašanje greške  $E$ , širine intervala poverenja  $C$  i gornje granice rizika  $R_g$  u zavisnosti od vrednosti regularizacionog hiperparametra  $\lambda$ .

## 11.4 Linearni modeli

Linearni modeli polaze od pretpostavke da se veza između ciljne promenljive i svojstava može približno modelovati linearnim modelom, uz eventualnu jednostavnu nelinearnu transformaciju povrh toga. Odnosno, forma modela je

$$f_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x}) = g\left(\sum_{i=1}^m w_i x_i\right) \quad (11.1)$$

za neku funkciju  $g$  koja zavisi od konkretnе primene. Pritom, kada se govori o linearном modelu bitno je naglasiti da se izraz „linearни“ odnosi na linearost relacije po parametrima  $w_i$ , a da sâma svojstva mogu biti nelinearno transformisana.

**Primer 11.5.** Primeri linearnih modela su:

- $f_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2$
- $f_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 \cos(x_1) + w_2 x_2^2 + w_3 e^{x_3}$
- $f_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2$

Sledeći primjeri ne predstavljaju linearne modele:

- $f_{\mathbf{w}}(\mathbf{x}) = \frac{w_1 x_1}{w_2 + x_2}$
- $f_{\mathbf{w}}(\mathbf{x}) = \frac{\cos(w_1 x_1) w_2 x_2}{e^{w_3 x_3}}$

U nastavku diskutujemo osnovne varijante linearnih modela za regresiju i klasifikaciju.

### 11.4.1 Linearna regresija

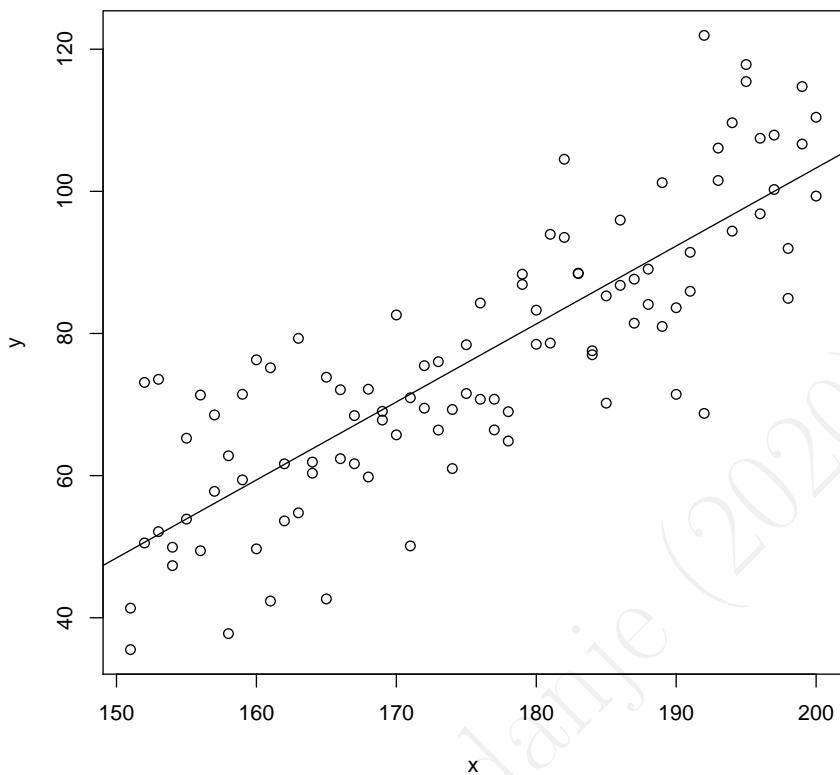
Linearna regresija predstavlja metod regresije zasnovan na linearnom modelu. Pretpostavka je da je funkcija  $g$  iz jednakosti (11.1) identitet tj. da se predviđanje dobija kao linearna kombinacija vrednosti svojstava:

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^m w_i x_i$$

Zadatak linearne regresije je određivanje vrednosti parametara  $\mathbf{w}$  koji najbolje odgovaraju trening podacima, tj. najbolje odgovaraju očekivanjima iz iskustva.

Pored osnovnog zadatka pronalaženja prediktivnog modela, linearna regresija korisna je i za ustanavljanje jačine uticaja nekog svojstva na vrednost ciljne promenljive. Naime, veće apsolutne vrednosti koeficijenata  $w_i$  označavaju jači uticaj svojstava  $x_i$  uz koji stoje. Znak koeficijenta određuje smer uticaja svojstva. Može se meriti i statistička značajnost ovog uticaja ali se, jednostavnosti radi, u nastavku fokusiramo samo na osnovni problem određivanja optimalnih vrednosti koeficijenata  $\mathbf{w}$  i proveru kvaliteta naučenog modela.

Najjednostavniji slučaj linearne regresije je predviđanje vrednosti  $y$  na osnovu samo jednog svojstva  $x$ . Primera radi, možemo razmatrati predviđanje telesne mase na osnovu visine. Primetna je zakonitost da su visoki ljudi uglavnom teži od niskih ljudi. Potrebno je modelovati tu zavisnost. Od nje postoje i odstupanja, ali kao i



Slika 11.8: Primer jednostavne regresije kojom se predviđa telesna težina na osnovu visine.

inače, u mašinskom učenju prihvatamo činjenicu da će greške postojati. Dakle, u ovom slučaju razmatraćemo linearni model oblika

$$f_{\mathbf{w}}(x) = w_0 + w_1 x$$

što je standardna jednačina linearne funkcije.

Na slici 11.8 je prikazano 100 tačaka pri čemu svaka odgovara jednoj instanci, tj. jednom ispitaniku. Koordinata  $x$  predstavlja visinu, a  $y$  telesnu masu. Na slici se može primetiti opšti trend linearog povećanja telesne mase u zavisnosti od visine, koji je prikazan pravom. Takođe, primetno je i da jedan, mali broj tačaka značajno odstupa. Ovakve tačke nazivamo *odudarajućim podacima* (eng. *outliers*). Prikazana prava predstavlja linearni model datih podataka. Metod kojim se do njega dolazi biće prikazan u nastavku.

Ako se koristi opšta jednačina linearnog modela (11.1), moguće je uključiti veći broj svojstava u predviđanje vrednosti ciljne promenljive. Naime, visina nije dovoljna da u potpunosti objasni variranje telesne težine. Dodatna svojstva koji bi vodila ka poboljšavanju predviđanja mogu da se odnose na način života pojedinaca — koliko vremena dnevno provode u sedećem položaju, koliko se bave sportom, koliko kalorija unose dnevno i slično. Umesto prave, u ovakvom slučaju regresioni model bi predstavljao jednu hiperravan.

Osnovni kriterijum izbora vrednosti koeficijenata linearnog modela je smanjivanje odstupanja između vrednosti koje model predviđa i vrednosti koje ciljna promenljiva ima u podacima. Ovaj problem obično se formuliše kao problem minimizacije srednjekvadratne greške. Funkcija gubitka je  $L(f_{\mathbf{w}}(\mathbf{x}), y) = (\mathbf{w} \cdot \mathbf{x} - y)^2$ , pa je odgovarajući minimizacioni problem

$$\min_{\mathbf{w}} \sum_{i=1}^N (\mathbf{w} \cdot \mathbf{x}_i - y_i)^2 + \lambda \Omega(\mathbf{w})$$

pri čemu je  $N$  broj instanci u trening skupu,  $\lambda$  regularizacioni hiperparametar, a  $\Omega(\mathbf{w})$  regularizacioni izraz (videti poglavlje 11.3). Alternativno, u matričnoj notaciji, isti problem može se zapisati kao

$$\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \Omega(\mathbf{w})$$

U slučaju da je  $\Omega(\mathbf{w}) = \|\mathbf{w}\|_2^2$ , ispostavlja se da za postavljeni problem postoji jednostavno rešenje koje ne zahteva korišćenje optimizacionih metoda:

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}$$

pri čemu je  $\mathbf{I}$  jedinična matrica i važi

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1n} \\ 1 & x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & & \vdots \\ 1 & x_{N1} & x_{N2} & \cdots & x_{Nn} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

**Primer 11.6.** Potrebno je modelovati vazdušno zagađenje izraženo u terminima indeksa kvaliteta vazduha (veće vrednosti su nepoželjnije), a u zavisnosti od populacije, maksimalne brzine vetra i indikatora da li je grad u kotlini ili ne. Raspoloživi su (stvarni) podaci za četiri dana – 5, 10, 15. i 20. januar 2020. godine u Beogradu, Sarajevu i Temišvaru. Želimo da dobijemo model kojim je moguće predviđati zagađenje, ali i koji je moguće analizirati. Naime, apsolutna vrednost koeficijenata i njihov znak govore nam koliko je koje svojstvo bitno i u kom smeru njegova promena utiče na promenu ciljne promenljive.

Matrice  $\mathbf{X}$ ,  $\bar{\mathbf{X}}$  i vektor  $\mathbf{y}$ , dati u nastavku sadrže podatke na osnovu kojih treba trenirati model linearne regresije. Primetimo da je prva kolona matrica  $\mathbf{X}$  i  $\bar{\mathbf{X}}$  jednaka jedinici, što omogućava postojanje slobodnog parametra  $w_0$  u linearном modelu (primetite da nije eksplicitno predviđen opštom formom linearog modela 11.1), što je korisna praksa. Takođe, svojstvo koje označava da li se grad nalazi u kotlini ili ne je kategoričke prirode. Pošto je binarno, njegove vrednosti moguće je kodirati brojevima 0 i 1 (u slučaju kategoričkih svojstava sa više promenljivih, bilo bi potrebno nešto komplikovanije modelovanje u koje ovde ne ulazimo). Dalje, promenljive koje koristimo u modelu izražavaju se na različitim skalamama. Veličina koeficijenata u modelu zavisi od skale na kojoj se mere vrednosti svojstava. Kako bi koeficijenti bili uporedivi, sva svojstva treba da uzimaju vrednosti u sličnom rasponu. Jedan način da se to postigne je da se kolone matrice  $\mathbf{X}$  (osim kolone jedinica) standardizuju – od svake vrednosti u koloni se oduzima njen prosek, a potom se sve vrednosti dele standardnom devijacijom kolone. Tako se od matrice  $\mathbf{X}$  dobija matrica  $\bar{\mathbf{X}}$  (uz zaokruživanje na dve decimale):

$$\mathbf{X} = \begin{bmatrix} 1 & 1687132 & 35 & 0 \\ 1 & 1687132 & 13 & 0 \\ 1 & 1687132 & 10 & 0 \\ 1 & 1687132 & 13 & 0 \\ 1 & 555210 & 21 & 1 \\ 1 & 555210 & 5 & 1 \\ 1 & 555210 & 5 & 1 \\ 1 & 555210 & 13 & 1 \\ 1 & 468162 & 37 & 0 \\ 1 & 468162 & 19 & 0 \\ 1 & 468162 & 21 & 0 \\ 1 & 468162 & 16 & 0 \end{bmatrix} \quad \bar{\mathbf{X}} = \begin{bmatrix} 1 & 1.41 & 1.81 & -0.71 \\ 1 & 1.41 & -0.44 & -0.71 \\ 1 & 1.41 & -0.75 & -0.71 \\ 1 & 1.41 & -0.44 & -0.71 \\ 1 & -0.63 & 0.38 & 1.41 \\ 1 & -0.63 & -1.26 & 1.41 \\ 1 & -0.63 & -1.26 & 1.41 \\ 1 & -0.63 & -0.44 & 1.41 \\ 1 & -0.78 & 2.01 & -0.71 \\ 1 & -0.78 & 0.17 & -0.71 \\ 1 & -0.78 & 0.38 & -0.71 \\ 1 & -0.78 & -0.14 & -0.71 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 70 \\ 168 \\ 181 \\ 141 \\ 121 \\ 245 \\ 247 \\ 112 \\ 63 \\ 80 \\ 67 \\ 61 \end{bmatrix}$$

Primenom formule

$$\mathbf{w} = (\bar{\mathbf{X}}^\top \bar{\mathbf{X}})^{-1} \bar{\mathbf{X}}^\top \mathbf{y}$$

dobija se vektor parametara

$$\mathbf{w} = \begin{bmatrix} 129.67 \\ 24.19 \\ -46.11 \\ 29.83 \end{bmatrix}$$

Dobijene vrednosti parametara sugerisu određene zakonitosti – naseljenost grada je pozitivno korelisana sa zagađenjem, vetrovitost negativno, a zatvorenost u kotlinu, takođe pozitivno. Pritom, model sugerise da je vetrovitost najvažniji od pomenuta tri svojstva. Ipak, koliko možemo verovati ovim zaključcima? Najbolje bi bilo primeniti model na podatke na kojima nije treniran, kako bi se videlo koliko precizno predviđa. Takođe, moguće je proveriti da li se model uopšte uspešno prilagodio podacima za trening. O evaluaciji modela će biti reči kasnije, ali možemo proveriti koren srednjekvadratne greške modela. On ima vrednost

29. Poredenja radi, standardna devijacija vrednosti ciljne promenljive je 65. Standardna devijacija je koren srednjekvadratne greške u odnosu na model koji konstantno predviđa prosečnu vrednost promenljive  $y$ , te se može zaključiti da su predviđanja ovde opisanog modela značajno bolja od takvog jednostavnog modela. Ipak, ima još dosta prostora za njegovo unapređenje, što nije ni neobično imajući u vidu da model ne uzima u obzir sve parametre koji utiču na zagodenje.

Osnovni problem pri određivanju optimalnih vrednosti koeficijenata  $\mathbf{w}$  je potencijalna loša uslovjenost matrice  $\mathbf{X}$  (za male promene elemenata polazne matrice, moguće su ogromne promene elemenata inverzne matrice). Naime, moguće je da su neka svojstva linearne zavisna ili da su jako korelirana. U tom slučaju, matrica  $\mathbf{X}$ , pa i matrica  $\mathbf{X}^\top \mathbf{X}$  je neinvertibilna ili loše uslovljena, pa se, ako je  $\lambda = 0$ , optimalne vrednosti koeficijenata  $\mathbf{w}$  ne mogu izračunati ili su previše nestabilne. Stoga se preporučuje da se prilikom linearne regresije uvek koristi regularizacija, pri čemu se kao regularizacioni izraz najčešće koristi kvadrat euklidske norme (eng. *ridge regression*). Moguće je koristiti i druge norme kako u regularizacionom izrazu, tako i u funkciji greške, što dovodi do varijanti metode sa različitim ponašanjima.

U gore navedenom rešenju problema regularizovane linearne regresije

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}$$

moguće je da je dimenzija matrice  $\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}$  velika i da je njeno invertovanje računski previše zahtevno. U takvim situacijama minimizacija se vrši metodama optimizacije poput gradijentnog spusta.

Osnovna i očigledna pretpostavka linearne regresije je da linearni model adekvatno izražava vezu između svojstava i ciljne promenljive. Ipak, ne može se očekivati da vrednosti ciljne promenljive budu uvek jednakе vrednostima modela zbog postojanja šuma, odnosno slučajne greške u podacima. Poreklo šuma može biti nesavršenost opreme kojom se vrši merenje, slučajna priroda samog fenomena ili to što izbor linearne zavisnosti predstavlja svesnu odluku da se inače kompleksna zavisnost donekle pojednostavi radi lakše analize. Stoga, obično se pretpostavlja da ciljna promenljiva ima oblik

$$y = \mathbf{w} \cdot \mathbf{x} + \varepsilon$$

gde je  $\varepsilon \sim \mathcal{N}(0, \sigma^2)$  normalno raspodeljena slučajna promenljiva koja označava šum, pri čemu je standardna devijacija  $\sigma$  konstantna.<sup>3</sup> Neformalno, time se pretpostavlja da se greške „poništavaju“, odnosno da se prebacivanja i podbacivanja javljaju jednako često, da su pritom velike greške vrlo malo verovatne, kao i da veličina greške ne zavisi od vrednosti  $y$  (pošto je  $\sigma$  konstantno).

#### 11.4.2 Logistička regresija

Logistička regresija predstavlja jednu od najkorišćenijih metoda klasifikacije. Glavni razlozi za to su jednostavnost, efikasno treniranje i postojanje verovatnosne interpretacije rezultata. Ograničenje ove metode je da je primenljiva samo na binarnu klasifikaciju — na slučaj kada svaka instanca može pripadati jednoj od dve klase koje se mogu označiti brojevima 1 i -1. Ovakav izbor brojeva samo je tehnička pogodnost i za njega ne postoji nikakav suštinski razlog pošto su oznake klasa zapravo kategoričke vrednosti. Osnovna ideja logističke regresije je da se vrši predviđanje verovatnoće  $P(y = 1|\mathbf{x}) = f_{\mathbf{w}}(\mathbf{x})$  da instanca  $\mathbf{x}$  pripada klasi 1. Očigledno, mora važiti  $f_{\mathbf{w}}(\mathbf{x}) \in [0, 1]$ . Tada je verovatnoća pripadnosti drugoj klasi  $P(y = -1|\mathbf{x}) = 1 - P(y = 1|\mathbf{x}) = 1 - f_{\mathbf{w}}(\mathbf{x})$ .<sup>4</sup>

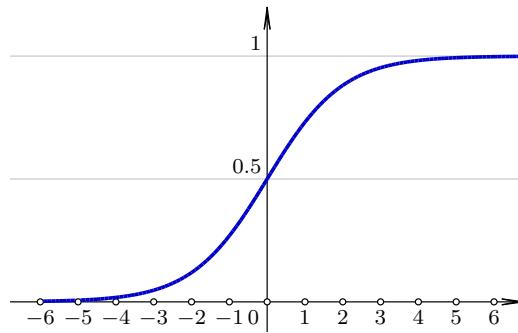
Postavlja se pitanje šta je pogodna forma za modele logističke regresije. Da bi se modelovala verovatnoća, potrebno je da model bude funkcija koja uzima sve vrednosti u intervalu  $[0, 1]$ . Da bi se to postiglo pomoću linearnog modela  $\mathbf{w} \cdot \mathbf{x}$ , potrebno je izabrati funkciju  $g$  (koja figuriše u formi modela 11.1) tako da slika interval  $[-\infty, \infty]$  u interval  $[0, 1]$ . Jedna takva funkcija je *sigmoidna funkcija*:  $\sigma(x) = 1/(1 + e^{-x})$ . Ovo nije jedina funkcija koja zadovoljava pomenuuti zahtev, ali je pogodna i iz tehničkih razloga, poput jednostavnosti izvoda ( $\sigma'(x) = \sigma(x)(1 - \sigma(x))$ ), i često se koristi u mašinskom učenju. Grafik sigmoidne funkcije prikazan je na slici 11.9. Logistički model dobija se komponovanjem prethodne dve funkcije i ima formu:

$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

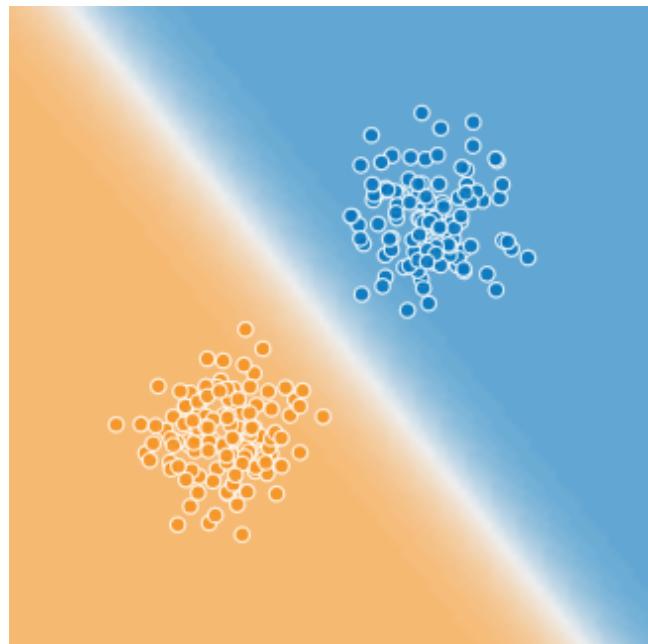
Primer sa kategorizacijom članaka na računarske i ostale je tipičan primer situacije u kojoj je prirodno primeniti logističku regresiju. Pošto logistički model kada važi  $\mathbf{w} \cdot \mathbf{x} > 0$  predviđa verovatnoću veću od 0.5 da

<sup>3</sup> Imajući u vidu da se smatra da važi  $y = \mathbf{w} \cdot \mathbf{x} + \varepsilon$  i pretpostavku  $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ , važi  $y \sim \mathcal{N}(\mathbf{w} \cdot \mathbf{x}, \sigma^2)$  za svaki vektor vrednosti svojstava  $\mathbf{x}$ . Ova konstatacija pruža drugi pogled na linearnu regresiju — da se zapravo radi o izboru normalne raspodele sa promenljivim prosekom koja najbolje opisuje raspodelu podataka.

<sup>4</sup> Kao što je konstatovano da linearna regresija predstavlja izbor normalne raspodele sa promenljivim prosekom koja najbolje opisuje podatke, tako se, na osnovu navedenog, može konstatovati da logistička regresija predstavlja izbor Bernulijeve raspodele  $\mathcal{B}(f_{\mathbf{w}}(\mathbf{x}))$ , takve da važi  $y \sim \mathcal{B}(f_{\mathbf{w}}(\mathbf{x}))$ .



Slika 11.9: Grafik sigmoidne funkcije.



Slika 11.10: Podaci koji se sastoje od dve klase i prava koja ih razdvaja. Boje ukazuju na regione prostora koji pripadaju različitim klasama.

instanca  $\mathbf{x}$  pripada klasi 1, a kada važi  $\mathbf{w} \cdot \mathbf{x} < 0$ , verovatnoća manju od 0.5 (tada je verovatnoća da pripada klasi -1 veća od 0.5), sledi da se, kao i metoda navedena u primeru kategorizacije članaka, i logistička regresija može interpretirati kao metoda koja traži razdvajajuću hiperravnu između instanci dve klase. Pritom, što je neka tačka dalja od razdvajajuće hiperravnih, to je vrednost  $\mathbf{w} \cdot \mathbf{x}$  veća po apsolutnoj vrednosti, a samim tim je i vrednost  $\sigma(\mathbf{w} \cdot \mathbf{x})$  bliža vrednosti 0 ili 1 u zavisnosti od znaka vrednosti  $\mathbf{w} \cdot \mathbf{x}$ . Drugim rečima, što je instanca dublje u oblasti prostora koji pripada nekoj klasi, to model izražava veću sigurnost da ona pripada toj klasi. Ovo ponašanje potpuno je u skladu sa intuicijom. Na slici 11.10 prikazan je primer skupa podataka i prava dobijena logističkom regresijom koja ih klasificuje.

Kao što je navedeno u motivaciji logističke regresije, verovatnoća  $P_{\mathbf{w}}(y = 1|\mathbf{x})$  predviđa se formulom

$$P_{\mathbf{w}}(y = 1|\mathbf{x}) = f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

Takođe, važi

$$P_{\mathbf{w}}(y = -1|\mathbf{x}) = 1 - P_{\mathbf{w}}(y = 1|\mathbf{x}) = \frac{e^{-\mathbf{w} \cdot \mathbf{x}}}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}} = \frac{1}{e^{\mathbf{w} \cdot \mathbf{x}} + 1} = \frac{1}{1 + e^{\mathbf{w} \cdot \mathbf{x}}}$$

Odavde se može izvesti opšti zaključak:

$$P_{\mathbf{w}}(y|\mathbf{x}) = \frac{1}{1 + e^{-y\mathbf{w} \cdot \mathbf{x}}}$$

S obzirom na to da postoji verovatnosna interpretacija, intuitivno je vrednosti parametara izabrati tako da verovatnoća javljanja instanci koje čine trening skup bude maksimalna pri izabranim vrednostima parametara.

Pod standardno korišćenom prepostavkom da instance predstavljaju nezavisne uzorke, ta verovatnoća jednaka je proizvodu

$$\prod_{i=1}^N P_{\mathbf{w}}(y_i | \mathbf{x}_i)$$

koji se naziva *funkcijom verodostojnosti parametra* (eng. *likelihood function*). Kako je korišćenje proizvoda iz tehničkih razloga<sup>5</sup> nepreporučljivo, umesto funkcije verodostojnosti koristi se njen logaritam. Kako je logaritam monotono rastuća funkcija, maksimumi funkcije verodostojnosti i njenog logaritma koincidiraju. Kako je logaritam broja koji je između 0 i 1 negativan, umesto maksimizacije logaritma verodostojnosti, može se minimizovati njegova negativna vrednost

$$\begin{aligned} -\log \prod_{i=1}^N P_{\mathbf{w}}(y_i | \mathbf{x}_i) &= -\sum_{i=1}^N \log P_{\mathbf{w}}(y_i | \mathbf{x}_i) = \\ -\sum_{i=1}^N \log \frac{1}{1 + e^{-y_i \mathbf{w} \cdot \mathbf{x}_i}} &= \sum_{i=1}^N \log(1 + e^{-y_i \mathbf{w} \cdot \mathbf{x}_i}) \end{aligned}$$

Ova veličina (podeljena brojem instanci) je greška modela koja odgovara *logističkoj funkciji gubitka*:  $L(y, f_{\mathbf{w}}(\mathbf{x})) = \log(1 + e^{-y \mathbf{w} \cdot \mathbf{x}})$ . Po dodavanju regularizacije, minimizacioni problem koji se rešava postaje:

$$\min_{\mathbf{w}} \sum_{i=1}^N \log(1 + e^{-y_i \mathbf{w} \cdot \mathbf{x}_i}) + \lambda \Omega(\mathbf{w})$$

Ovaj problem nema jednostavno rešenje kao u slučaju linearne regresije, već se mora sprovesti postupak optimizacije. U tu svrhu moguće je koristiti gradijentni spust, ali postoje i efikasnije metode. Posebna pogodnost za optimizaciju u slučaju logističke regresije je što se minimizuje konveksna funkcija koja ima jedan globalni minimum i ne postoji mogućnost da proces optimizacije završi u nekom neoptimalnom lokalnom minimumu, što je problem sa nekim drugim metodama učenja, poput neuronskih mreža.

Treba imati u vidu da za primenu logističke regresije nije neophodno da klase budu linearno razdvojive. Trening logističke regresije sigurno konvergira zahvaljujući tome što metode optimizacije koje se koriste u ovom problemu sigurno nalaze minimum konveksne funkcije. Naravno, preciznost dobijenog modela ne može biti savršena ako se radi o linearno nerazdvojivom problemu.

## 11.5 Neuronske mreže

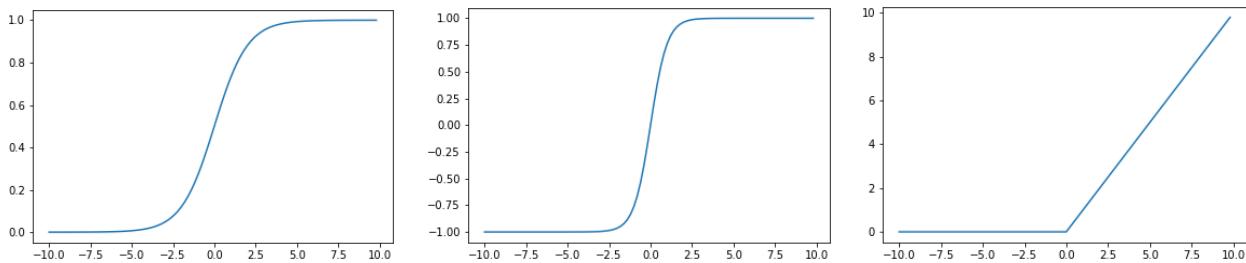
Neuronske mreže predstavljaju jedan od najstarijih i najuspešnijih pristupa mašinskom učenju. Datiraju iz 50-ih godina dvadesetog veka i više puta su u svojoj istoriji bile u prvom planu istraživanja, a potom potiskivane od strane drugih metoda. Njihov poslednji uspon počinje 2006. godine sa pojmom takozvanih *dubokih neuronskih mreža*, a dobija novi zalet 2012. godine kada duboke mreže po prvi put značajno nadilaze postojeće metode u problemima *računarskog vida* (eng. *computer vision*). Od tada su neuronske mreže postigle velike uspehe i u mnogim drugim oblastima, primarno u obradi prirodnog jezika, obradi govora, igranju igara, itd. Trenutno predstavljaju metod mašinskog učenja sa najširim spektrom primena.

Postoje različite vrste neuronskih mreža, često specijalizovanih za konkretnе primene. Ipak, prepoznatljive su po svojoj kompozitnoj strukturi — svaka mreža sastoji se od određenog broja elementarnih jedinica — neurona, koji po gruboj analogiji sa biološkim neuronima u mozgu, jedni drugima prosleđuju signale i izračunavaju nove signale na osnovu onih koji su im prosleđeni. Tačna struktura povezanosti neurona i način na koji oni vrše izračunavanja definiše o kakvoj mreži se radi. Izbori između različitih struktura i načina izračunavanja, koje vrši stručnjak koji dizajnira mrežu, nazivaju se *arhitekturom mreže*. U nastavku će najpre biti opisana struktura pojedinačnih neurona, a potom kako se oni organizuju u različite fundamentalne arhitekture. Na kraju će biti opisano kako se neuronske mreže primenjuju u različitim praktičnim problemima.

### 11.5.1 Neuron

Neuroni su osnovne jedinice izračunavanja čijim se povezivanjem grade neuronske mreže. Po uzoru na biološke neurone, oni kao ulaze primaju signale od drugih neurona i transformišu ih na neki način čime proizvode svoj izlazni signal. Svi signali predstavljeni su realnim brojevima, a njihova obrada vrši se nekom jednostavnom matematičkom funkcijom koja obično uključuje linearnu kombinaciju ulaznih signala, a potom

<sup>5</sup>Proizvod velikog broja vrednosti između 0 i 1 lako može postati 0 usled potkoračenja.



Slika 11.11: Sigmoidna funkcija, hiperbolički tangens i ispravljena linearna jedinica.

nelinearnu transformaciju te linearne kombinacije. Formalno, ako neuron kao ulaz prima  $n$  signala  $x_1, \dots, x_n$ , on vrši sledeće izračunavanje:

$$g\left(\sum_{i=1}^n w_i x_i\right) \quad (11.2)$$

gde su  $w_1, \dots, w_n$  parametri ili težine neurona kojima se vrši linearno kombinovanje ulaza, a  $g$  je nelinearna aktivaciona funkcija kojom se ta linearna kombinacija transformiše.

Aktivaciona funkcija nije jednoznačno definisana i postoji više mogućih izbora. Najčešće su monotone, neprekidne i skoro svuda diferencijabilne. Neke od često korišćenih aktivacionih funkcija su sigmoidna funkcija:

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

hiperbolički tangens:

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1},$$

ispravljena linearna jedinica (eng. *rectified linear unit* – ReLU):

$$\text{ReLU}(x) = \max(0, x)$$

i njene varijacije. Grafici ovih funkcija prikazani su na slici 11.11. U novije vreme najčešće se koriste varijante ispravljene linearne jedinice zbog njenih poželjnih tehničkih svojstava koja ovde neće biti diskutovana.

Zahvaljujući monotonosti, što je vrednost linearne kombinacije veća to je izlazni signal jači. Očito, visoke težine nekih ulaza vode jakom uticaju tih ulaza na izlazni signal, a vrednosti bliske nuli umanjuju uticaj tih ulaza. Znak težine definiše da li neki ulaz potkrepljuje ili inhibira ispoljavanje signala.

Može se primetiti da, u slučaju da se za aktivacionu funkciju uzme sigmoidna funkcija, neuron predstavlja model logističke regresije, dok u slučaju da se za aktivacionu funkciju uzme identitet, neuron predstavlja model linearne regresije. Otud će neuronske mreže predstavljati uopštenja ovih modela, a takođe će biti sposobne da vrše klasifikaciju i regresiju.

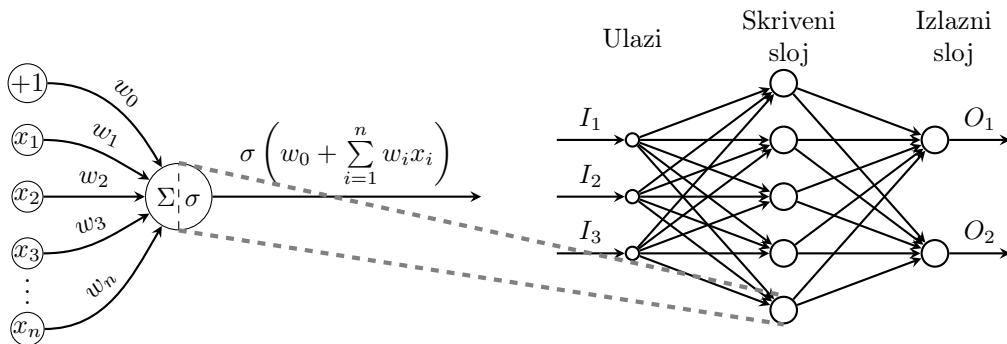
## 11.5.2 Potpuno povezane neuronske mreže

*Potpuno povezane neuronske mreže* (eng. *fully connected neural networks*) predstavljaju najstariji od trenutno korišćenih modela neuronskih mreža. Naziv su dobile po strukturi povezanosti neurona – neuroni se slažu u slojeve, grupe neurona i povezuju se tako što svi neuroni jednog sloja kao ulaze primaju vrednosti svih neurona prethodnog sloja, a svoje vrednosti prosleđuju svim neuronima narednog sloja. Arhitektura potpuno povezane neuronske mreže prikazana je na slici 11.12. Na njoj su prikazani ulazi, koji ne predstavljaju neurone, već numeričke vrednosti, jedan sloj neurona koji se naziva *skrivenim* jer se njegove vrednosti koriste isključivo unutar modela i jedan koji se naziva *izlaznim*, pošto daje predviđanja koja model izračunava. Skrivenih slojeva može biti više od jednog i u tom slučaju se govori od *dubokim neuronskim mrežama* (eng. *deep neural networks*). Mreže koje se danas koriste u praksi su mahom duboke.

Ukoliko mreža ima  $L$  slojeva, ona se može precizno definisati na sledeći način:

$$\begin{aligned} \mathbf{h}_0 &= \mathbf{x} \\ \mathbf{h}_i &= g(\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{w}_{i0}), \quad i = 1, \dots, L \\ f_{\mathbf{w}}(\mathbf{x}) &= \tilde{g}(\mathbf{h}_L) \end{aligned}$$

gde  $\mathbf{h}_i$  za  $i = 1, \dots, L$  predstavlja vektor vrednosti  $i$ -tog sloja neurona,  $\mathbf{W}_i$  je matrica parametara čije vrste predstavljaju vektore parametara neurona  $i$ -tog sloja koji množe ulazne vrednosti,  $\mathbf{w}_{i0}$  vektor slobodnih koeficijenata tih neurona, a  $\mathbf{w}$  predstavlja skup svih tih parametara  $\mathbf{W}_i$  i  $\mathbf{w}_{i0}$ . Pod primenom aktivacione funkcije  $g$



Slika 11.12: Arhitektura potpuno povezane neuronske mreže i struktura neurona.

na vektor podrazumeva se primena funkcije  $g$  na svaku koordinatu vektora pojedinačno. Očito, izračunavanje koje mreža vrši sastoji se od linearnih matričnih transformacija i primena aktivacionih funkcija. Funkcija  $\tilde{g}$  može predstavljati drugačiju aktivacionu funkciju od funkcije  $g$ , što je uobičajeno za poslednji sloj mreže. Ovaj poslednji izbor zavisi od toga u koju svrhu se mreža koristi. Obično se, u slučaju regresije koristi funkcija  $\tilde{g}(x) = x$  pri čemu je  $x$  skalar ( $\mathbf{h}_L$ , u gornjoj formulaciji modela). U ovom slučaju, mreža se može razmatrati kao linearna regresija nad pretposlednjim slojem mreže, a svi slojevi zaključno sa njim se mogu razumeti kao mehanizam konstrukcije novih svojstava iz ulaza.

U slučaju klasifikacije, podrazumeva se da poslednji sloj mreže ima onoliko neurona koliko ima klasa, a na poslednjem sloju koristi se sledeća vektorska aktivaciona funkcija  $\tilde{g}$ :

$$\text{softmax}(a_1, \dots, a_m) = \left( \frac{e^{a_1}}{\sum_{i=1}^m e^{a_i}}, \dots, \frac{e^{a_m}}{\sum_{i=1}^m e^{a_i}} \right) \quad (11.3)$$

Kako su koordinate ovog vektora nenegativne i imaju zbir 1, ovaj vektor se može interpretirati kao raspodela verovatnoće nad  $m$  različitim klasama.

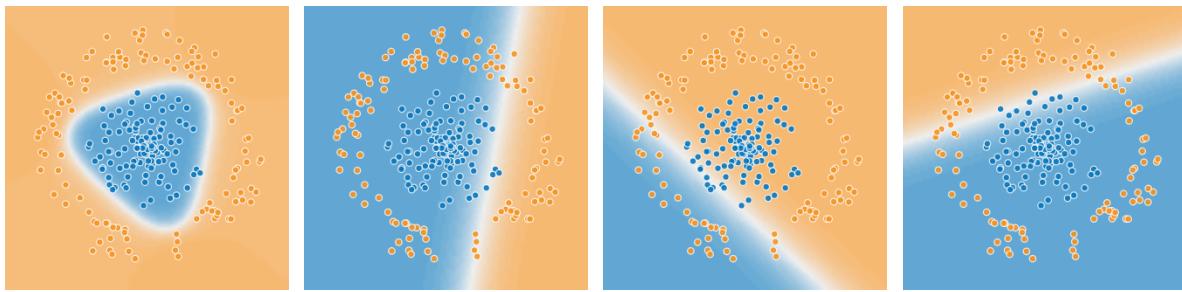
**Primer 11.7.** Neka je potrebno uraditi klasifikaciju neke instance u tri moguće klase. U tom slučaju, neuronska mreža ima tri izlazna neurona od kojih svaki izračunava linearnu kombinaciju svojih ulaza. Neka su te vrednosti recimo  $a_1 = 2$ ,  $a_2 = 0$  i  $a_3 = -1$ . Eksponencijalne vrednosti su onda  $e^{a_1} = 7.389$ ,  $e^{a_2} = 1.000$  i  $e^{a_3} = 0.368$ . Normiranjem ovih vrednosti njihovom sumom dobija se raspodela verovatnoće po klasama (0.844, 0.114, 0.042). Na osnovu modela, najverovatnije je da data instance pripada prvoj klasi, a model nam kroz izračunate verovatnoće daje i procenu pouzdanosti u taj izbor.

Primetimo da je u slučaju binarne klasifikacije umesto dva izlaza i funkcije softmax dovoljno koristiti jedan izlaz i sigmoidnu funkciju, što se može razumeti kao logistička regresija nad pretposlednjim slojem mreže. Ilustrujmo primerom da je neuronska mreža moćnija od logističke regresije.

**Primer 11.8.** Logistička regresija je pogodna za probleme nalik onim ilustrovanim na slici 11.10. Čak i ako ima nekog preklapanja klasa, ako je moguće uočiti dve grupe koje se u velikoj meri mogu razdvojiti pravom, logistička regresija će naći tu pravu. S druge strane, u slučaju skupa podataka datog na slici 11.13, logistička regresija je nemoćna, dok neuronska mreža može da nauči razdvajanje klasa koje je prikazano. Zanimljivo je analizirati kako neuronska mreža to radi.

Mreža korišćena u primeru ima jedan skriveni sloj sa tri neurona i jedan neuron u izlaznom sloju koji poput logističke regresije daje verovatnoću da instance pripada jednoj izabranoj klasi. Svaki od neurona u skrivenom sloju uči neku transformaciju podataka i one su ilustrovane na slici 11.13. Primetimo da svaki od neurona daje visoku vrednost plavim instancama, ali da nije u stanju da se ograniči samo na njih jer klase nisu linearno razdvojive. Ipak, samo su plave instance te kojima sva tri neurona pridružuju visoku vrednost. Ako se te tri vrednosti daju kao ulaz dodatnom neuronu, on može da daje izlaznu vrednost samo za instance za koju su sve tri njene ulazne vrednosti visoke, a da daje nisku vrednost čim je jedna od tih vrednosti niska.

Kako bi se neuronska mreža trenirala da obavlja neki zadatak, potrebno je definisati i funkciju greške, a potom obezbediti algoritam učenja. Funkcije greške mogu varirati od primene do primene, ali postoje neki



Slika 11.13: Podaci koji nisu razdvojivi pravom i razdvajanje naučeno neuronskom mrežom (prva slika) i neuro-nima njenog skrivenog sloja (preostale tri slike). Plava boja predstavlja vrednosti bliske jedinici, a narančasta vrednosti bliske nuli.

običajeni izbori. U slučaju regresije to je obično kvadratna greška

$$L(f_{\mathbf{w}}(\mathbf{x}), y) = (f_{\mathbf{w}}(\mathbf{x}) - y)^2$$

U slučaju klasifikacije u  $m$  klasa, smatra se da je za svaku instancu ispravna klasa definisana vektorom  $\mathbf{y}$  dimenzije  $m$ , koji se sastoji od  $m - 1$  nula i jedne jedinice, pri čemu je jedinica na poziciji  $j$  ako i samo ako je  $j$  tačna klasa. U tom slučaju, kao greška se koristi *unakrsna entropija*:

$$L(f_{\mathbf{w}}(\mathbf{x}), \mathbf{y}) = - \sum_{j=1}^m y_j \log f_{\mathbf{w}}^{(j)}(\mathbf{x})$$

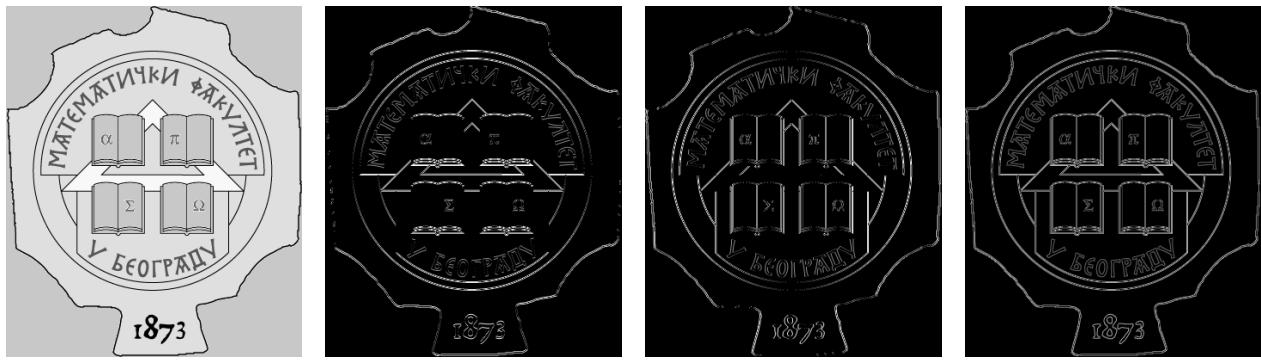
gde  $f_{\mathbf{w}}^{(j)}$  označava  $j$ -tu koordinatu vektorske funkcije  $f_{\mathbf{w}}$ , tj. procenjenu verovatnoću klase  $j$ . Ova funkcija se može izvesti iz statističkog principa maksimalne verodostojnosti, ali ovde neće biti navedeno to izvođenje. Ipak, nije teško intuitivno razumeti ovu funkciju. Naime, ukoliko je verovatnoća stvarne klase  $j$  visoka, logaritam vrednosti  $f_{\mathbf{w}}^{(j)}(\mathbf{x})$  je blizak nuli i doprinos sumi koji mu odgovara je mali. Ostali sabirci jednaki su nuli jer su sve vrednosti vektora  $\mathbf{y}$  osim  $y_j$  jednake nuli (podrazumeva se da važi  $0 \log 0 = 0$ ). Dakle, kada model ispravno proceni verovatnoće klasa, greška je mala. Ukoliko model ispravnoj klasi daje verovatnoću blisku nuli, negativna vrednost logaritma te verovatnoće je velika, množi se vrednošću  $y_j$  koja je za pravu klasu jednaka 1, dok su ostali sabirci kao i pre jednaki nuli, pa je ukupna suma velika. Ovo je upravo ponašanje koje se očekuje od dobre funkcije greške.

Kada je definisana funkcija greške modela, treba je minimizovati po parametrima modela. Ovo se u slučaju neuronskih mreža uvek radi nekom od gradijentnih metoda optimizacije koje obično predstavljaju modifikacije gradijentnog spusta. Algoritam za izračunavanje gradijenta u slučaju neuronskih mreža naziva se algoritmom propagacije unazad (eng. *backpropagation*). Kako je mreža sastavljena od linearnih matričnih transformacija i primena aktivacionih funkcija, algoritam za izračunavanje gradijenta se prosti sastoji od množenja izvoda tih aktivacionih funkcija i odgovarajućih matrica u skladu sa pravilom izvoda složene funkcije. Zato neće biti reči o detaljima ovog algoritma. Iz prethodnog grubog opisa može se naslutiti jedan opšti problem dubokih neuronskih mreža, a to je da u slučaju njihove velike dubine dolazi do velikog broja pomenutih množenja. Zato koordinate gradijenta često bivaju bliske nuli ili ogromne po apsolutnoj vrednosti, što vodi ili previše sporoj konvergenciji gradijentnog spusta ili velikoj nestabilnosti ovog postupka. Postoje načini da se takvo ponašanje ublaži, ali se time nećemo baviti.

Imajući u vidu da je broj ulaza fiksiran, potpuno povezane neuronske mreže su specijalizovane za primenu nad podacima predstavljenim u vidu vektora svojstava fiksne dužine, što nije nužno slučaj sa svim modelima neuronskih mreža. Zbog toga se potpuno povezane neuronske mreže ne mogu jednostavno primeniti na neke vrste podataka, poput slika (koje mogu biti različitih dimenzija), i rečenica prirodnog jezika i vremenskih serija koje prirodno imaju različite dužine. Ipak, korisne su u slučaju modelovanja tabelarnih podataka ili kao komponenta drugih vrsta neuronskih mreža.

### 11.5.3 Konvolutivne neuronske mreže

Konvolutivne neuronske mreže su vrsta neuronskih mreža. Zahvaljujući njima ostvareni su najveći pomaci u praktičnim primenama mašinskog učenja, pre svega u oblasti računarskog vida. Njihova osnovna ideja je da je filtere koji se standardno koriste u obradi signala moguće učiti iz podataka umesto ručno dizajnirati. Stoga će najpre ukratko biti diskutovan jedan primer obrade signala. Različite vrste obrade signala mogu se predstaviti



Slika 11.14: Primer detekcije ivica primenom konvolutivnog filtera. Prikazane su izvorna slika, horizontalne ivice, vertikalne ivice i ukupne ivice.

operacijom konvolucije između dve matrice od kojih jedna predstavlja signal (tj. sliku), a druga filter kojim se definije obrada koja se vrši. Konvolucija matrica  $\mathbf{A} \in \mathbb{R}^{m \times n}$  i  $\mathbf{B} \in \mathbb{R}^{p \times q}$  definiše se na sledeći način:<sup>6</sup>

$$(\mathbf{A} * \mathbf{B})_{ij} = \sum_{k=1}^p \sum_{l=1}^q \mathbf{A}_{i+k,j+l} \mathbf{B}_{kl} \quad i = 1, \dots, m-p, j = 1, \dots, n-q \quad (11.4)$$

Jednostavna primer primene konvolucije je detekcija ivica na slici. Ivice predstavljaju nagle promene boje na slici. Matematički, brzina promene se izražava izvodom ili u slučaju funkcija više promenljivih (kao što je slika), pomoću gradijenta. Njegov pravac predstavlja smer najbrže promene, a norma intenzitet te promene. Otud, ivice se mogu detektovati izračunavanjem norme gradijenta i pronalaženjem njenih visokih vrednosti. Za izračunavanje norme, potrebno je odrediti parcijalne izvode slike po horizontalnom i po vertikalnom pravcu. Time se određuju horizontalna i vertikalna komponenta ivice u svakoj tački. Te komponente ćemo prosto nazivati horizontalnim i vertikalnim ivicama. Rasterizovana slika je diskretna, a ne neprekidna funkcija, umesto parcijalnih izvoda, koriste se razlike vrednosti susednih piksela. Ako je slika u nijansama sive boje data u vidu matrice  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , horizontalne ivice mogu se detektovati oduzimanjem vrednosti piksela koji su neposredno jedan ispod drugog, što se primenom konvolucije može izraziti na sledeći način:

$$\mathbf{H} = \mathbf{A} * \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Vertikalne ivice mogu se dobiti na sličan način:

$$\mathbf{V} = \mathbf{A} * [1 \ -1]$$

Matrice koje se koriste za detekciju horizontalnih i vertikalnih ivica predstavljaju primere filtera. Norma gradijenta, koja predstavlja ivice, onda se računa kao:

$$\mathbf{I}_{ij} = \sqrt{\mathbf{H}_{ij}^2 + \mathbf{V}_{ij}^2} \quad i = 1, \dots, m-1, j = 1, \dots, n-1$$

Rezultat ovog procesa detekcije ivica dat je na slici 11.14.

Jedan pristup mašinskom učenju nad slikama mogao bi biti da se najpre izračunaju izlazi različitih, vešto osmišljenih filtera koji bi imali za cilj da izračunaju različite aspekte slike (poput lokacija ivica) i da se te vrednosti koriste kao svojstva na osnovu kojih bi model mogao da uči. Ovakav pristup, iako moguć nije lak jer traži angažovanje eksperata za slike i inventivnost u izboru relevantnih filtera. Takođe je moguće da ni eksperti nisu u stanju da definisu dovoljno dobre filtere za različite zadatke. Otud se došlo do ideje da se filteri ne moraju unapred definisati, već da se mogu učiti iz podataka tako da greška modela bude mala. Važno pitanje je kako se koncept konvolutivnih filtera uklapa u okvire neuronskih mreža. Ključno zapažanje je da operacija konvolucije data formulom (11.4) za svako  $i$  i  $j$  izračunava skalarni proizvod elemenata matrice  $B$  sa odgovarajućim elementima matrice  $A$  istih dimenzija. Može se zamisliti da se matrica  $B$  pomera duž matrice  $A$  i računa skalarne proizvode na različitim lokacijama i tako daje potpun rezultat konvolucije. Ovo je ilustrovano slikom 11.15. Lako je primetiti da u neuronu, kako je definisan formulom (11.2), upravo figuriše

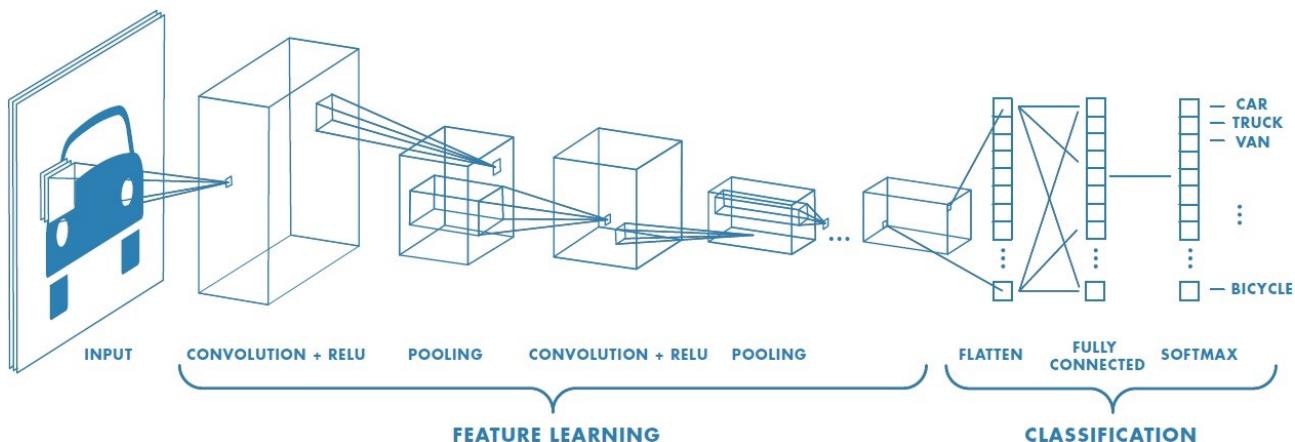
<sup>6</sup>Strogo govoreći, definisana operacija nije zaista konvolucija, već unakrsna korelacija (eng. *cross-correlation*), ali je razlika u slučaju neuronskih mreža nepostojeća i u praksi se implementira na prikazani način.

$$\begin{array}{|c|c|c|c|c|c|c|} \hline
 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline
 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline
 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ \hline
 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline
 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline
 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline
 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline
 \end{array}
 \quad \mathbf{I}$$
  

$$\begin{array}{|c|c|c|} \hline
 1 & 0 & 1 \\ \hline
 0 & 1 & 0 \\ \hline
 1 & 0 & 1 \\ \hline
 \end{array}
 \quad \mathbf{K}$$
  

$$\begin{array}{|c|c|c|c|c|} \hline
 1 & 4 & 3 & 4 & 1 \\ \hline
 1 & 2 & 4 & 3 & 3 \\ \hline
 1 & 2 & 3 & 4 & 1 \\ \hline
 1 & 3 & 3 & 1 & 1 \\ \hline
 3 & 3 & 1 & 1 & 0 \\ \hline
 \end{array}
 \quad \mathbf{I} * \mathbf{K}$$

Slika 11.15: Ilustracija konvolucije. Broj rezultujuće matrice u zelenom polju se dobija kao skalarni proizvod filtera  $\mathbf{K}$  i crveno obojenog dela ulazne slike  $\mathbf{I}$ . Ostatak rezultujuće matrice dobija se analogno.



Slika 11.16: Shema konvolutivne neuronske mreže.

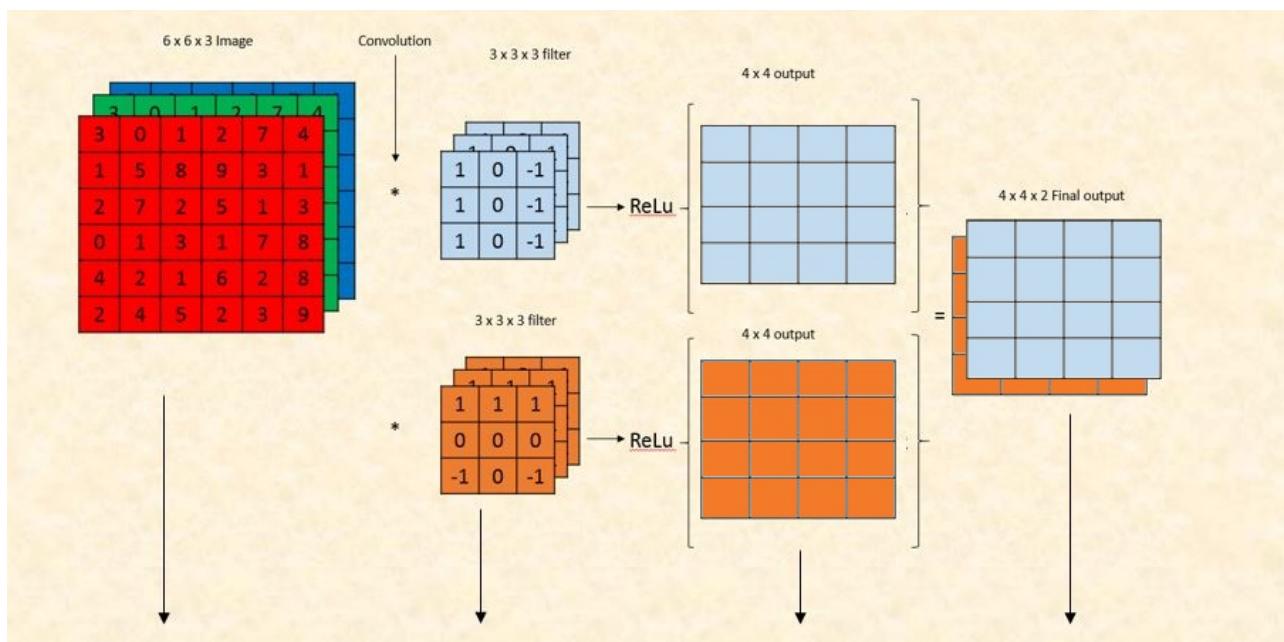
skalarni proizvod! Ako se svakom od skalarnih proizvoda koji se izračunavaju u konvoluciji pridruži jedan neuron možemo te neurone smatrati jednim slojem neuronske mreže. Ono što je za njih specifično je da svi imaju iste parametre – one koji čine filter. To deljenje parametara među neuronima čini prepoznatljivo svojstvo konvolutivnih mreža.

U mnogim situacijama, kao u obradi slika, potrebno je imati više ulaznih kanala, tako da ulaz ne predstavlja nužno matricu već takozvani *tenzor*, odnosno višedimenzioni niz. Shodno tome, i operacija konvolucije se uopštava na proizvoljan broj dimenzija, po potrebi.

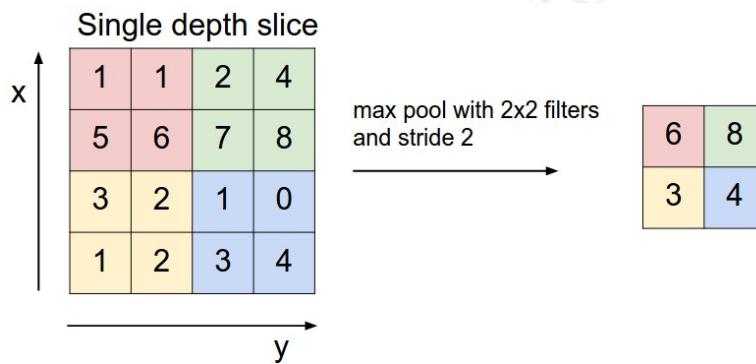
Iako je principijelno jasno da konvolutivni filteri mogu biti naučeni, i dalje je potrebno definisati arhitekturu mreže koja to radi. Shema jedne arhitekture konvolutivne mreže prikazana je na slici 11.16. Na njoj se vidi da se ulaz transformiše nizom konvolucija i agregacija koji se međusobno prepliću, a da se na kraju nalazi potpuno povezana mreža koja, u ovom primeru, vrši finalnu klasifikaciju ulaza.

Neuronska mreža u svakom sloju može učiti veći broj filtera, što je u praksi uvek i slučaj. Jedan filter može naučiti da detektuje vertikalne ivice, drugi horizontalne, treći kose i tako dalje. Izlaz svakog filtera, koji nazivamo *kanalom*, predstavlja matricu koja ima visoke vrednosti na mestima gde filter detektuje ono što traži, a niske tamo gde detekcija nije uspešna. Svaki kanal se transformiše nelinearnom aktivacionom funkcijom, kao i kod potpuno povezanih neuronskih mreža. Skup filtera koji deluju nad istim ulazom nazivamo *konvolutivnim slojem* i on na svom izlazu daje tenzor sastavljen slaganjem kanala koje daju različiti filteri u istom sloju. Nad njime je moguće izgraditi sledeći konvolutivni sloj i tako dalje. Filteri viših konvolutivnih slojeva zapravo konstruišu nova svojstva nad svojstvima koje su konstruisali niži slojevi. Na taj način oni traže složenije obrasce u slici od filtera nižeg nivoa, a koji su obrasci bitni za dati problem, uči se minimizacijom greške koju mreža pravi na podacima.

**Primer 11.9.** Slika 11.17 prikazuje ilustraciju dejstva konvolutivnog sloja. Sloj kao ulaz uzima tenzor koji se u ovom primeru sastoji od tri kanala dimenzija  $6 \times 6$ . Sloj se sastoji od dva filtera. Primetimo da filter takođe ima treću dimenziju koja se poklapa sa brojem kanala ulaznog tenzora. Svaki od ovih filtera proizvodi jedan izlazni kanal. Kako konvolucija smanjuje dimenzije ulaznih kanala, izlazni su dimenzija  $4 \times 4$ . Na



Slika 11.17: Ilustracija konvolutivnog sloja.



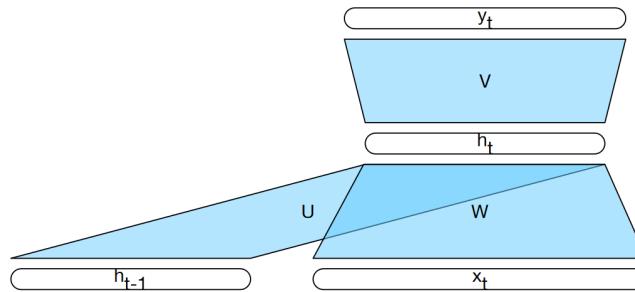
Slika 11.18: Ilustracija agregacije sloja.

*njih se primjenjuje aktivaciona funkcija. Izlazni tenzor se formira slaganjem dobijenih kanala.*

Pored konvolutivnih slojeva, često se koriste i *slojevi agregacije* (eng. *pooling layer*) koji služe agregaciji informacije u cilju smanjenja količine izračunavanja i broja parametara u mreži. Obično se realizuju tako što se svaki kanal ulaznog tenzora podeli na delove određenih dimenzija koji se zamene jednom vrednošću poput maksimuma vrednosti u tom tenzoru. Konkretni izbor maksimuma je čest i intuitivan. Naime, visoke vrednosti u tenzoru znače da je filter koji je proizveo tu vrednost uspeo na nekom mestu da detektuje obrazac u svom ulazu koji traži. Uprosečavanjem bi se visoke vrednosti izgubile zbog prisustva niskih, dok maksimum čuva informaciju o tome da je traženi obrazac pronađen. Ipak, agregacijom se gubi informacija o tačnoj lokaciji. Značaj tog gubitka zavisi od konkretne primene.

**Primer 11.10.** Slika 11.18 prikazuje ilustraciju agregacije maksimumom dimenzija  $2 \times 2$ . Primetimo da se za razliku od konvolucije, maksimum ne računa nad preklapajućim delovima ulaznog kanala, već se on deli na susedne i disjunktnе delove koji odgovaraju dimenzijama agregacije. Ukoliko ulazni sloj ima više kanala, svaki se obrađuje nezavisno.

Na kraju, neretko se na krajnje izlaze takve mreže nadovezuje potpuno povezana mreža koja kao svoje ulaze uzima sve vrednosti krajnjeg tenzora dobijenog iz konvolutivnog dela. Upotreba potpuno povezane mreže ipak nije neophodna i postoje arhitekture koje je ne uključuju, tipično u slučaju kada su ulazne slike različitih dimenzija.



Slika 11.19: Shema zavisnosti stanja, ulaza i izlaza kod rekurentne neuronske mreže.

Trening konvolutivnih mreža u svrhe klasifikacije i regresije se ne razlikuje od potpuno povezanih mreža. Specifičnosti vezane za neke druge vrste zadataka biće diskutovane kasnije kada bude reči o praktičnim primenama.

#### 11.5.4 Rekurentne neuronske mreže

Mnogi zadaci mašinskog učenja vezani su za sekvenčne podatke, tj. podatke koji se mogu predstaviti u vidu niski jednostavnih elemenata. Primeri takvih zadataka su klasifikacija tekstova prema njihovoj temi ili sentimentu (osećanju, raspoloženju, afektivnom tonu), kreiranje sažetaka tekstova, prevodenje rečenica sa jednog prirodnog jezika na drugi, predviđanje kretanja cena akcija na berzi, predviđanje budućeg stanja pacijenta na osnovu zdravstvene istorije i tako dalje. Sekvenčni podaci najčešće su podaci promenljive dužine. Potpuno povezane mreže očekuju kao ulaze vektorske reprezentacije fiksne dimenzije i stoga su neprimerene obradi ovakvih podataka. Dodatno, redosled reči u rečenici može se menjati bez promene smisla rečenice. S druge strane, redosred svojstava u vektorskoj reprezentaciji podataka ne može se slobodno menjati jer mreža uči različite vrednosti parametara za različita svojstva. Već je komentarisano da se konvolutivne mreže mogu primeniti na slike različitih dimenzija, pa bi se moglo primeniti i na sekvence različitih dužina. Ipak, ako i izlaz treba da bude promenljive dužine, koja se ne poklapa sa dužinom ulaza (kao u slučaju automatskog prevodenja sa jednog na drugi prirodni jezik), potrebno je drugačije rešenje.

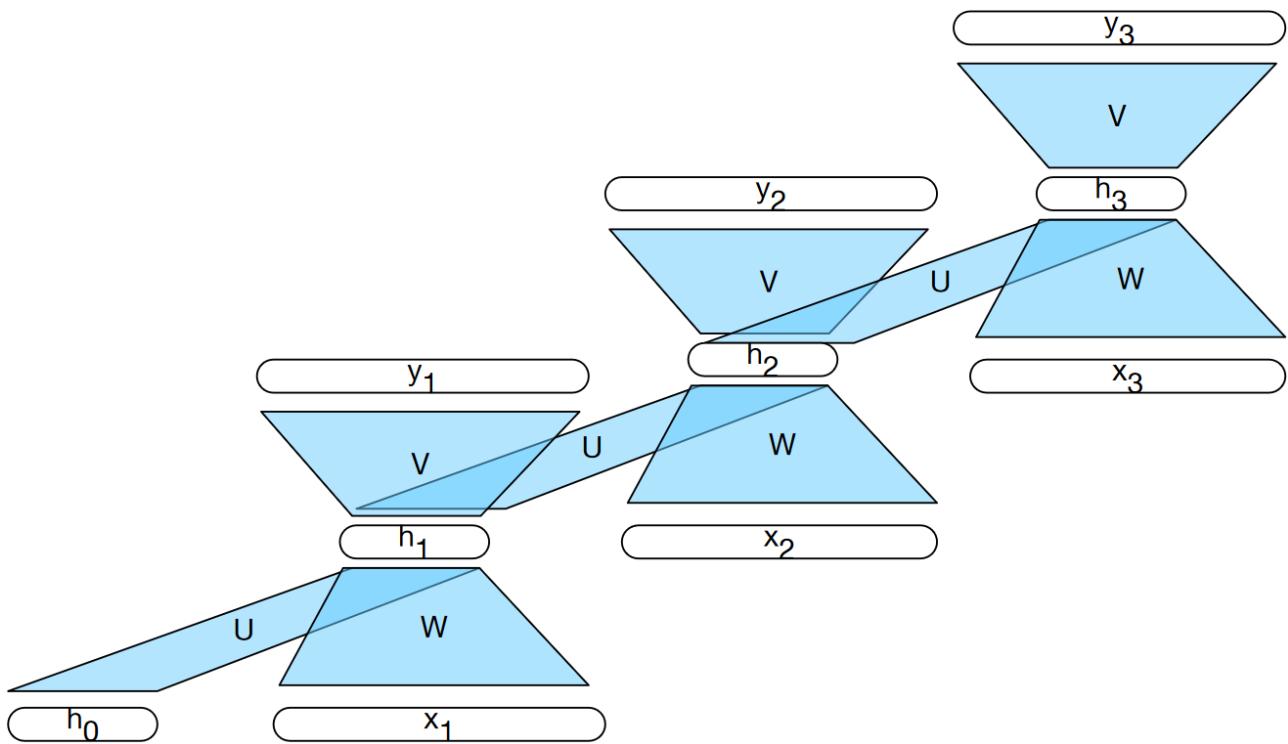
Osnovna ideja rekurentnih neuronskih mreža je da se sekvenca čita element po element, pri čemu se održava stanje procesa obrade u vidu vektora koji se menja iz koraka u korak. Da bi ovaj proces funkcionišao, potrebno je definisati kako naredno stanje zavisi od prethodnog stanja i od tekućeg ulaza. Takođe, kako mreža treba da daje i izlaze, potrebno je definisati i na koji način izlaz u svakom koraku zavisi od tekućeg stanja. Ove zavisnosti se, kao i u prethodnim slučajevima neuronskih mreža, izražavaju linearnim transformacijama sa nelinearном aktivacionom funkcijom, pri čemu se matrice koje definišu linearne transformacije uče. Ova shema zavisnosti prikazana je na slici 11.19.

Osnovni model neuronske mreže može se opisati sledećim formulama:

$$\begin{aligned}\mathbf{h}_0 &= 0 \\ \mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b}_h), \quad t = 1, \dots, T \\ \mathbf{o}_t &= \tilde{g}(\mathbf{V}\mathbf{h}_t + \mathbf{b}_o), \quad t = 1, \dots, T\end{aligned}$$

gde  $\mathbf{h}_t$  predstavlja vektor skrivenog stanja u trenutku  $t$ , a matrica  $\mathbf{U}$  definiše zavisnost novog od prethodnog skrivenog stanja, matrica  $\mathbf{W}$  zavisnost novog skrivenog stanja od novog ulaza, a matrica  $\mathbf{V}$  definiše zavisnost izlaza od tekućeg skrivenog stanja. Vektori  $\mathbf{b}_h$  i  $\mathbf{b}_o$  su vektori slobodnih članova. Pomenute matrice i slobodni članovi skupa čine parametre mreže  $\mathbf{w}$ . Bitno je primetiti da ovi parametri ne zavise od koraka  $t$ . Vrednost modela  $f_{\mathbf{w}}(\mathbf{x}_1, \dots, \mathbf{x}_T)$  može biti bilo koji podskup proizvedenih izlaznih vrednosti, zavisno od namene modela. Potom je nad datim izlazom moguće izračunati grešku u odnosu na prave izlaze i formirati funkciju greške koju je moguće optimizovati gradijentnim metodama.

Kako bi bilo jasnije kako trening funkcioniše u slučaju rekurentnih mreža korisno je primetiti da se one mogu predstaviti na način sličan ranije prikazanim modelima takozvanim mehanizmom razmotavanja koji je ilustrovan na slici 11.20. Naime, za svaku sekvencu u trening skupu, moguće je formirati mrežu poput mreže na slici, pri čemu će za različite sekvence te mreže biti različite tužine. Ipak, to ne menja činjenicu da se sve mogu smatrati složenim funkcijama i da mehanizmi izračunavanja gradijenata po formuli izvoda složene funkcije mogu biti primenjeni na svaku od tako dobijenih mreža i potom sumirani kako bi se dobio ukupni gradijent na svim podacima.



Slika 11.20: Razmotavanje rekurentne neuronske mreže.

Moderne rekurentne mreže uključuju mnoštvo unapređenja u odnosu na ovaj osnovni princip, ali njihovo razmatranje izlazi van okvira ovog teksta.

## 11.6 Modeli zasnovani na instancama: metoda $k$ najbližih suseda

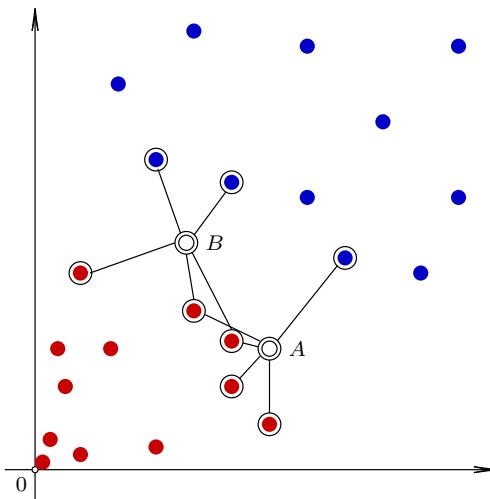
Osnovna karakteristika metoda učenja zasnovanih na instancama je da ne grade eksplicitan model podataka u vidu neke funkcije kao što to radi većina metoda mašinskog učenja. Stoga se predviđanje ne vrši na osnovu već formulisanog modela, nego na osnovu skupa instanci za trening. Umesto izgradnje modela, instance predviđene za treniranje se čuvaju i bivaju upotrebljene tek kad je potrebno izvršiti predviđanje za novu, nepoznatu instancu. Time se većina izračunavanja premešta iz faze učenja u fazu primene. Najpoznatija metoda ove vrste je metoda  $k$  najbližih suseda (eng. *k nearest neighbours*). Ove metode često ne uključuju sva četiri pomenuta elementa dizajna algoritama nadgledanog učenja. Na primer, algoritam  $k$  najbližih suseda nema eksplicitnu formu modela, funkciju greške, niti koristi optimizaciju.

Metoda  $k$  najbližih suseda zasniva se na vrlo jednostavnoj ideji — pronaći  $k$  instanci najsličnijih nepoznatoj instanci, takozvanih suseda i predvideti vrednost njene ciljne promenljive na osnovu vrednosti koje odgovaraju susedima. U slučaju klasifikacije, predviđanje odgovara najčešćoj klasi među klasama suseda, a u slučaju regresije, predviđanje može biti prosečna vrednost ciljne promenljive suseda. Tekst u nastavku se odnosi na problem klasifikacije, ali analogni zaključci mogu se izvesti i u slučaju regresije. Pojam sličnosti instanci najjednostavnije se formalizuje preko funkcija rastojanja.

**Definicija 11.1** (Rastojanje). Neka je  $X$  skup instanci. Funkcija  $d : X \times X \rightarrow \mathbb{R}$  predstavlja rastojanje na skupu  $X$  ukoliko zadovoljava sledeće uslove:

1.  $d(\mathbf{x}, \mathbf{y}) \geq 0$ , pritom  $d(\mathbf{x}, \mathbf{y}) = 0 \Leftrightarrow \mathbf{x} = \mathbf{y}$  (pozitivna definitnost)
2.  $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$  (simetričnost)
3.  $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$  (nejednakost trougla)

**Primer 11.11.** Neki primeri rastojanja su:

Slika 11.21: Stabilnost klasifikacije pomoću algoritma  $n$  najbližih suseda.

- $d(\mathbf{x}, \mathbf{y}) = \sqrt[n]{\sum_i (x_i - y_i)^n}$

- $d(\mathbf{x}, \mathbf{y}) = \begin{cases} 0, & \mathbf{x} = \mathbf{y} \\ 1, & \mathbf{x} \neq \mathbf{y} \end{cases}$

Intuitivno, što je rastojanje između dva objekta veće, to je sličnost između njih manja i obratno.<sup>7</sup> Drugim rečima, rastojanje između objekata predstavlja meru njihove različitosti. Moguće je izabrati raznovrsne funkcije rastojanja a naravno poželjno je izabrati funkciju koja stvarno oslikava različitost između dva objekta, u smislu relevantnom za razmatrani domen.

Kada je funkcija rastojanja izabrana, najjednostavniji način klasifikacije je klasifikovanje nepoznate instance u klasu kojoj pripada instanca trening skupa najbliža nepoznatoj instanci. Ovo je primer metode  $k$  najbližih suseda za  $k = 1$ . U opštem slučaju, metoda  $k$  najbližih suseda sastoji se u pronalaženju  $k$  instanci iz trening skupa koje su najbliže zadatoj instanci i njenom klasifikovanju u klasu čiji se elementi najčešće javljaju među pronađenih  $k$  najbližih suseda. U slučaju izjednačenog ishoda između više klasa nije moguće doneti odluku, ali se opisani osnovni algoritam može modifikovati kako bi se i ovakvi slučajevi razrešili.

Analizirajmo detaljnije metodu  $k$  najbližih suseda. Razmotrimo nepoznate instance  $A$  i  $B$  prikazane na slici 11.21. Metodom  $k$  najbližih suseda uz korišćenje euklidskog rastojanja, instanca  $A$  biva klasifikovana u crvenu klasu za sve vrednosti  $k$  od 1 do 5. Klasifikacija instance  $A$  je postojana zato što se ona nalazi blizu crvenih instanci, a udaljena je od plavih instanci. S druge strane, klasa instance  $B$  može da varira u zavisnosti od broja  $k$ . Za  $k = 1$ , instanca  $B$  klasificuje se u crvenu klasu. Za  $k = 2$ , ne može se odlučiti. Za  $k = 3$ , instanca  $B$  klasificuje se u plavu klasu. Za  $k = 4$ , ponovo nije moguće odlučiti, a za  $k = 5$ , instanca  $B$  ponovo se klasificuje u crvenu klasu. Klasifikacija instance  $B$  nije postojana jer se ona nalazi blizu instanci iz obe klase. Dakle, metoda  $k$  najbližih suseda postojana je u unutrašnjosti oblasti koju zauzimaju instance jedne klase, ali je nepostojana na obodu te oblasti. Nepostojanost klasifikacije može se demonstrirati menjanjem parametra  $k$ . Ali i za fiksiranu vrednost parametra  $k$ , može se uočiti nepostojanost pri variranju vrednosti svojstava instance, jer instanca iz oblasti jedne klase može takvim variranjem preći u oblast druge klase. Ovakvo ponašanje bi se moglo uočiti i kod drugih metoda klasifikacije.

Kao što se može videti u slučaju metode najbližih suseda, bitno svojstvo metoda zasnovanih na instancama je njihova lokalnost. Nepoznata instanca klasificuje se isključivo ili uglavnom na osnovu poznatih instanci koje se nalaze u njenoj blizini. Ovo svojstvo doprinosi fleksibilnosti modela koje ove metode (implicitno) grade. Samim tim, za manje vrednosti parametra  $k$  dobijaju se fleksibilniji modeli, koji su stoga skloniji preprilagođavanju, dok se za veće vrednosti parametra  $k$  dobijaju manje fleksibilni modeli manje skloni preprilagođavanju. Naravno, premala fleksibilnost vodi modelima koji se ne mogu dovoljno prilagoditi podacima i stoga loše uče, pa ni premala ni prevelika vrednost parametra  $k$  nije dobra. Očigledno, parametar  $k$  ima ulogu sličnu ulozi regularizacionog hiperparametra  $\lambda$ . Određivanje njihovih vrednosti biće zajednički diskutovano kasnije.

<sup>7</sup>Kao mere sličnosti, ponekad se koriste i funkcije koje nisu zasnovane na rastojanjima kao, na primer, funkcija  $\cos(\angle(\mathbf{x}, \mathbf{y})) = \frac{\mathbf{x} \cdot \mathbf{y}}{\sqrt{\mathbf{x} \cdot \mathbf{x}} \sqrt{\mathbf{y} \cdot \mathbf{y}}}$ .

Kako ovaj metod ne uključuje trening, već se instance koriste u vreme klasifikacije, trening skup se uvek može ažurirati, recimo dodavanjem novih instanci, bez utroška vremena na ponovni trening kao u slučaju drugih algoritama.

Razmotrimo primer primene algoritma  $k$  najблиžih suseda u problemu regresije.

**Primer 11.12.** Vratimo se na primer procene zagađenja vazduha. Ponavljamo standardizovane podatke (bez kolone jedinica).

$$\bar{\mathbf{X}} = \begin{bmatrix} 1.41 & 1.81 & -0.71 \\ 1.41 & -0.44 & -0.71 \\ 1.41 & -0.75 & -0.71 \\ 1.41 & -0.44 & -0.71 \\ -0.63 & 0.38 & 1.41 \\ -0.63 & -1.26 & 1.41 \\ -0.63 & -1.26 & 1.41 \\ -0.63 & -0.44 & 1.41 \\ -0.78 & 2.01 & -0.71 \\ -0.78 & 0.17 & -0.71 \\ -0.78 & 0.38 & -0.71 \\ -0.78 & -0.14 & -0.71 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 70 \\ 168 \\ 181 \\ 141 \\ 121 \\ 245 \\ 247 \\ 112 \\ 63 \\ 80 \\ 67 \\ 61 \end{bmatrix}$$

Neka je potrebno predvideti zagađenje za novu instancu  $(1000000, 15, 0)$  koja predstavlja grad od milion stanovnika, van kotline, na dan kada vetar duva brzinom od 15 kilometara na sat. Prvo je potrebno izvršiti standardizaciju ove instance kako bi promenljive koje je opisuju bile izražene na istoj skali kao promenljive u matrici podataka, čime se dobija instance  $(0.17, -0.24, -0.71)$ . Vektor rastojanja do datih instanci je:

$$\mathbf{d} = \begin{bmatrix} 2.39 \\ 1.25 \\ 1.34 \\ 1.25 \\ 1.01 \\ 2.49 \\ 2.49 \\ 2.28 \\ 3.24 \\ 1.04 \\ 1.13 \\ 0.96 \end{bmatrix}$$

Regresija može biti izvedena na primer pomoću tri najблиža suseda, uprosečavanjem njihovih vrednosti. To su instance sa rastojanjima 0.96, 1.01 i 1.04, a prosek njihovih  $y$  vrednosti je 87.33, što je konačno predviđanje.

## 11.7 Stabla odlučivanja

Razmotrimo „igru 20 pitanja“. Jedan igrač zamišlja neki predmet, a drugi treba da pogodi o kom je predmetu reč. Kako bi pogodio o kom predmetu se radi, igrač koji pogađa ima pravo da postavi 20 pitanja na koje odgovor može biti „da“ ili „ne“. Kada smatra da je postavio dovoljno pitanja, igrač može dati sud o kom predmetu se radi i igra se završava. Očito, proces ispitivanja može se predstaviti u vidu stabla koje u svakom čvoru ima po jedno pitanje, osim u listovima u kojima se nalazi sud igrača o nepoznatom predmetu. Svaki čvor osim listova ima dve grane označene sa „da“ ili „ne“, koje vode u podstablo koje odgovara nastavku ispitivanja posle razmatranog pitanja. Ovo je primer stabla odlučivanja. Ovakva stabla mogu se uopštiti odbacivanjem ograničenja na 20 pitanja i tako što bi se dozvolilo da odgovori ne moraju biti samo „da“ ili „ne“, već da mogu pripadati nekom unapred definisanom skupu za dato pitanje.

Stabla odlučivanja mogu se automatski naučiti iz skupa primera koji za svaku instancu uključuju vrednosti njenih bitnih svojstava i vrednost ciljne promenljive. Učenje stabala odlučivanja je metod klasifikacije ili regresije u kojem se model predstavlja u vidu stabla. Slično „igri 20 pitanja“, svakom čvoru takvog stabla odgovara test nekog svojstva instance, a grane koje izlaze iz čvora različitim vrednostima tog svojstva. Listovima odgovaraju predviđene vrednosti ciljne promenljive. Instance su opisane vrednostima svojih svojstava. Predviđanje se vrši

polazeći od korena, spuštajući se niz granu koja odgovara vrednosti testiranog svojstva instance za koju se vrši predviđanje. Predviđanje se dodeljuje instanci kad se dođe do lista.

Učenje stabala odlučivanja uspešno se primenjuje u različitim problemima. Još sredinom devedesetih godina prošlog veka, stabla odlučivanja primenjena su u klasifikaciji tumora i prognozi njihovog ponašanja. Svaka trening instance opisivana je pomoću 31 svojstva, a klasifikacije su date nezavisno od strane više stručnjaka. U astronomiji su stabla odlučivanja primenjena u cilju razlikovanja zvezda i tragova kosmičkih zraka na snimcima teleskopa Habl. Na osnovu 20 numeričkih svojstava, sa stablima dubine do 9 čvorova, postignuta je preciznost klasifikacije od 95%. Postoje primene stabala odlučivanja i u ekonomiji i drugim oblastima. Određivanje poze igrača kakvo vrši Majkrosoftov proizvod Kinekt biće detaljnije diskutovano kasnije.

Korišćenje stabala odlučivanja nije podjednako pogodno za sve probleme učenja. Poželjno je da skup vrednosti svojstava bude diskretan i mali, mada postoje varijante koje rade sa neprekidnim svojstvima. Stabla odlučivanja su uspešno primenljiva posebno u slučajevima kada je potrebno predstavljanje disjunkcija uslova. Ukoliko stablo odlučivanja instanci dodeljuje neko predviđanje, to znači da instance ispunjava sve uslove koji su definisani putanjom od korena do odgovarajućeg lista kroz stablo i oblika su  $svojstvo = vrednost$ . Stoga putanje kroz stablo predstavljaju konjunkcije ovakvih uslova. U slučaju klasifikacije, za svaku klasu moguće je uočiti putanje koje se završavaju listovima koji odgovaraju toj klasi. Disjunkcija svih takvih konjunkcija definiše instance koje pripadaju datoj klasi prema datom stablu.

Jedan od najpoznatijih algoritama za učenje stabala odlučivanja je ID3 na koji ćemo se skoncentrisati. Ovaj algoritam konstruiše stablo od korena naniže, pitajući se u svakom čvoru koje je najbolje svojstvo koje se u tom čvoru može testirati. To se određuje statističkim kriterijumom koji meri koliko dobro neko svojstvo samostalno klasificiše podatke. Za sve vrednosti odabranog svojstva kreiraju se grane do čvorova naslednika, a podaci za treniranje se dele između ovih čvorova tako da svaki od njih nasleđuje primere koji imaju odgovarajuću vrednost prethodno testiranog svojstva. Za svaki od čvorova naslednika ovaj postupak se primeni rekurzivno sve dok nije ispunjen bar jedan od sledeća dva uslova: (1) u putanji od korena do trenutnog čvora iskorišćena su sva svojstva, (2) sve instance za trening koje su pridružene trenutnom čvoru imaju istu vrednost ciljne promenljive. Svakom listu se pridružuje najčešća oznaka instanci za trening koje su mu pridružene. Algoritam je preciznije opisan na slici 11.22.

#### Algoritam: ID3(*Primeri, Svojstva*)

**Ulaz:** *Primeri* je skup instanci za trening, a *Svojstva* je lista svojstava koja se mogu testirati u čvorovima stabla

**Izlaz:** Stablo odlučivanja koje odgovara datim instancama

- 1: napravi koren čvor stabla  $R$ ;
- 2: **ako** sve instance iz *Primeri* pripadaju istoj klasi **onda**
- 3:     vrati čvor  $R$  sa oznakom te klase;
- 4: **ako** je *Svojstva* prazna lista **onda**
- 5:     vrati čvor  $R$  označen oznakom najčešće klase koja se javlja u *Primeri*;
- 6: **inace**
- 7:     neka je  $S \in Svojstva$  najbolje svojstvo (prema nekom statističkom kriterijumu) za testiranje u odnosu na *Primeri*;
- 8:     označi čvor  $R$  svojstvom  $S$ ;
- 9:     **za** svaku moguću vrednost  $v_i$  svojstva  $S$  **radi**
- 10:         dodaj granu iz  $R$  koja odgovara testu  $S = v_i$ ;
- 11:         neka je  $Primeri_{v_i}$  podskup od *Primeri* takav da svi njegovi elementi imaju vrednost  $v_i$  svojstva  $S$ ;
- 12:         **ako** je skup  $Primeri_{v_i}$  prazan **onda**
- 13:             na dodatu granu iz  $R$  dodaj list sa oznakom najčešće klase u *Primeri*;
- 14:         **inace**
- 15:             na dodatu granu nadoveži podstablo ID3( $Primeri_{v_i}, Svojstva \setminus \{S\}$ );
- 16: vrati  $R$ .

Slika 11.22: Algoritam ID3.

Veoma je važno pitanje statističkog kriterijuma koji se koristi za izbor najboljeg svojstva za testiranje u nekom čvoru. Algoritam ID3 obično bira svojstvo koje maksimizuje *dobitak informacije* na skupu instanci

koje su pridružene razmatranom čvoru. Dobitak informacije predstavlja razliku entropije u odnosu na ciljnu promenljivu skupa instanci za trening  $D$  pre deljenja i prosečne entropije posle deljenja prema nekom svojstvu  $S$ . Entropija predstavlja meru neuređenosti nekog sistema. Ako sa  $p_i$  označimo verovatnoću da instance pripada  $i$ -toj klasi, onda se entropija može definisati sledećim izrazom (pri čemu se smatramo da važi  $0 \cdot \log_2 0 = 0$ ):

$$\text{Entropija}(D) = - \sum_{i=1}^c p_i \log_2 p_i$$

Notacija  $\text{Entropija}(D_a)$  označava entropiju skupa instanci iz  $D$  koje imaju  $a$  kao vrednost nekog svojstva. Pritom, iz konteksta će biti jasno o kom svojstvu je reč. Dobitak informacije stoga predstavlja smanjenje u potrebnom broju bitova za kodiranje klase proizvoljne instance, kada je poznata vrednost koju na njoj ima svojstvo  $S$ . Dobitak informacije se formalno definiše na sledeći način:

$$\text{Dobitak}(D, S) = \text{Entropija}(D) - \sum_{v \in Vred(S)} \frac{|D_v|}{|D|} \text{Entropija}(D_v) \quad (11.5)$$

gde je  $c$  broj klasa, tj. broj mogućih vrednosti ciljne promenljive,  $p_i$  ideo instance iz skupa  $D$  koje pripadaju klasi  $i$  u celom skupu  $D$ ,  $Vred(S)$  predstavlja skup svih mogućih vrednosti svojstva  $S$ , a  $D_v = \{\mathbf{x} \in D | S(\mathbf{x}) = v\}$ , gde  $S(\mathbf{x})$  označava vrednost svojstva  $S$  za instance  $\mathbf{x}$ .

Pored entropije, postoje i druge mere koje se mogu koristiti za merenje neuređenosti nekog skupa. Jedna jednostavna mera je greška klasifikacije. Ona predstavlja grešku koja se čini ukoliko se sve instance nekog skupa klasifikuju u najbrojniju klasu u tom skupu. Stoga, ako je  $p_i$  verovatnoća da instance pripada  $i$ -toj klasi, greška klasifikacije se definiše izrazom

$$\text{Err}(S) = 1 - \max_i p_i$$

Za ovu meru može se definisati dobitak analogan opisanom dobitku informacije ukoliko se u izrazu 11.5 entropija zameni greškom klasifikacije.

Životinja	Veličina	Ishrana	Otrovnost	Noge	Evropa	Opasna
Lav	Velika	Meso	Ne	4	Ne	Da
Mačka	Mala	Meso	Ne	4	Da	Ne
Koala	Mala	Biljke	Ne	4	Ne	Ne
Zec	Mala	Biljke	Ne	4	Da	Ne
Komodo zmaj	Velika	Meso	Da	4	Ne	Da

Tabela 11.1: Skup primera za trening.

Životinja	Veličina	Ishrana	Otrovnost	Noge	Evropa	Opasna
Zmija	Mala	Meso	Da	0	Da	Da
Pčela ubica	Mala	Biljke	Da	6	Ne	Da
Morska krava	Velika	Biljke	Ne	0	Ne	Ne

Tabela 11.2: Dodatni primeri za trening.

**Primer 11.13.** U tabeli 11.1 su date instance koje opisuju različite životinje, sa datom klasifikacijom koja označava da li je životinja opasna po čoveka. Izdvojeno je nekoliko karakteristika koje bi mogle biti relevantne u određivanju vrednosti ciljne promenljive, ali su namerno dodata dva svojstva koji nisu relevantna – broj nogu i da li životinja živi u Evropi.

Da bi se izgradilo stablo odlučivanja, za svako od svojstava je potrebno izračunati dobitak informacije pri deljenju skupa podataka prema tom svojstvu. U prvom koraku, važi:

$$\text{Entropija}(D) = -\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} = 0.971$$

Ukoliko se izvrši podela instance po vrednosti prvog svojstva, dobijamo

$$\text{Entropija}(D_{Mala}) = -1 \cdot \log_2 1 - 0 \cdot \log_2 0 = 0$$

$$\text{Entropija}(D_{\text{Velika}}) = -0 \cdot \log_2 0 - 1 \cdot \log_2 1 = 0$$

$$\text{Dobitak}(D, \text{Velicina}) = 0.971 - \frac{2}{5} \cdot 0 - \frac{3}{5} \cdot 0 = 0.971$$

Slično se dobija:

$$\text{Dobitak}(D, \text{Ishrana}) = 0.42$$

$$\text{Dobitak}(D, \text{Otrovnost}) = 0.322$$

$$\text{Dobitak}(D, \text{Noge}) = 0$$

$$\text{Dobitak}(D, \text{Evropa}) = 0.42$$

Odavde se vidi da je najbolje svojstvo za testiranje u prvom čvoru svojstvo Veličina. Stablo koje se u ovom slučaju dobija primenom algoritma ID3 je dano na slici 11.23. U slučaju datih primera za učenje, dobijeno stablo je bilo očigledan izbor i bez primene bilo kakve metodologije.

Nešto komplikovanije stablo odlučivanja može se dobiti dodavanjem primera iz tabele 11.2.

Jedno ručno konstruisano stablo koje je saglasno sa podacima za trening dato je na slici 11.24. Izbor lošeg svojstva za testiranje u korenom čvoru je namerno učinjen. To dovodi do potrebe za ponavljanjem istih testova u levom i desnom podstablu, pošto informacija dobijena testiranjem u korenu nije relevantna za određivanje klase instance. Takođe, u slučaju životinja koje ne žive u Evropi, prisutan je nepotreban test vezan za njenu ishranu. U oba slučaja klasa je ista, pa se čvor sa tim testom može zameniti listom sa klasom DA. Upotreba algoritma ID3 daje bolje stablo odlučivanja.

Vrednosti entropije i dobitka informacije se sada razlikuju:

$$\text{Entropija}(D) = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1$$

Ukoliko se izvrši podela instanci po vrednosti prvog svojstva, dobijamo

$$\text{Entropija}(D_{\text{Mala}}) = -\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5} = 0.971$$

$$\text{Entropija}(D_{\text{Velika}}) = -\frac{1}{3} \log_2 \frac{1}{3} - \frac{2}{3} \log_2 \frac{2}{3} = 0.918$$

$$\text{Dobitak}(D, \text{Velicina}) = 1 - \frac{5}{8} \cdot 0.971 - \frac{3}{8} \cdot 0.918 = 0.049$$

Slično se dobija:

$$\text{Dobitak}(D, \text{Ishrana}) = 0.189$$

$$\text{Dobitak}(D, \text{Otrovnost}) = 0.549$$

$$\text{Dobitak}(D, \text{Noge}) = 0$$

$$\text{Dobitak}(D, \text{Evropska}) = 0.02$$

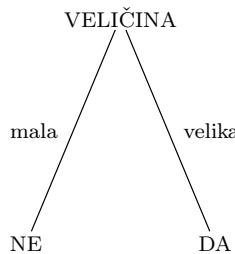
Posle dodavanja novih instanci, najbolje svojstvo za testiranje su Otrovnost i Ishrana. Rekurzivnom primenom ovog postupka dobija se stablo prikazano na slici 11.25. Ono je očigledno manje od ručno konstruisanog i ima relevantnija svojstva pri vrhu stabla, dok se dva nebitna svojstva uopšte ne testiraju.

Stabla odlučivanja mogu se koristiti i za probleme regresije. U tom slučaju, za merenje neuređenosti skupa instanci prirodan izbor je disperzija vrednosti ciljne promenljive za te instance:

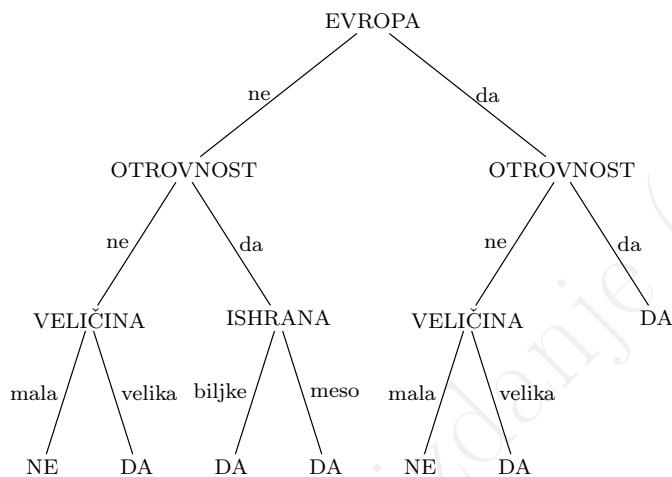
$$\frac{1}{N-1} \sum_{i=1}^N (y_i - \bar{y})^2$$

gde važi  $\bar{y} = \frac{1}{N} \sum_{i=1}^N y_i$ . U slučaju regresije potrebno je modifikovati i način na koji se listovima pridružuju predviđanja koja daju. Ona se mogu definisati kao prosek instanci koje su prilikom treninga pridružene listu.

Kao i drugi metodi učenja, ID3 se može shvatiti kao pretraga skupa dopustivih modela za onim koji najviše odgovara podacima za trening. Prostor pretrage je potpun prostor svih stabala odlučivanja. Svaka diskretna funkcija može se predstaviti nekim stablom odlučivanja, tako da se učenjem stabala odlučivanja može postići



Slika 11.23: Jednostavno stablo odlučivanja.



Slika 11.24: Ručno konstruisano stablo odlučivanja.

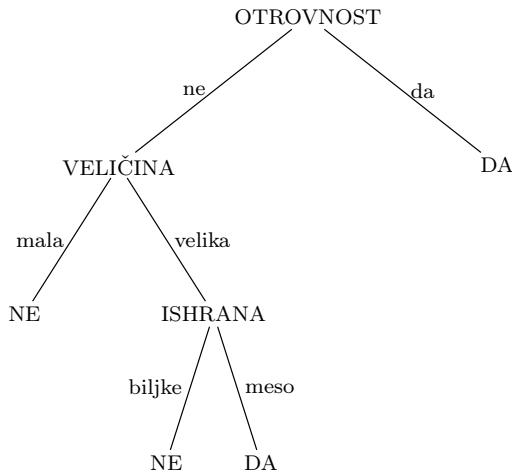
greška 0 na trening podacima ukoliko podaci nisu protivrečni. ID3 u svakom trenutku čuva samo jedan tekući model, pa je nemoguće znati koliko ima drugih modela koji su saglasni sa podacima za trening. Pošto nema vraćanja u pretrazi (*backtracking*), već se stablo gradi od jednostavnijeg ka složenijem, postoji opasnost od dostizanja lokalnih optimuma koji nisu globalni.

Algoritam ID3 preferira stabla sa manjom dubinom, kao i stabla u kojima se svojstva koja nose veći dobitak informacije nalaze bliže korenu. Ovo je posledica toga što izgradnja stabla počinje od praznog stabla pri čemu se dodaje nivo po nivo i posledica načina na koji se biraju svojstva koja se pridružuju čvorovima. Afinitet prema kraćim stablima je zanimljivo svojstvo jer je u skladu i sa odavno poznatim filozofskim principom kojim se često vode i naučnici — Okamovom oštricom: entitete ne treba umnožavati preko potrebe, tj. najjednostavnije objašnjenje je verovatno i najbolje. Svakoj putanji od korena do nekog od listova odgovara po jedno pravilo oblika

$$\text{if } S_1 = v_1 \wedge S_2 = v_2 \wedge \dots \wedge S_n = v_n \text{ then} \\ \text{Klase} = \text{klaša koja odgovara listu}$$

gde su  $S_i$   $0 \leq i \leq n$  svojstva koja se testiraju na putanji od korena do odgovarajućeg čvora, a  $v_i$  njihove vrednosti za datu instancu. Kako stabla odlučivanja sa manjom dubinom imaju manji broj listova, predstavljaju manje skupove ovakvih pravila, te ih možemo smatrati jednostavnijim.

Osvrnićemo se i na problem preprilagođavanja. Sa povećanjem dozvoljene dubine stabla, povećava se moć učenja, odnosno verovatnoća da će u skupu dopustivih modela biti nađen onaj koji dobro opisuje podatke. Zato se sa povećanjem dozvoljene dubine stabala, smanjuje greška na trening podacima. Međutim, ako nema ograničenja na dubinu stabla, takav skup modela je očigledno vrlo bogat i stoga postoji opasnost od preprilagođavanja. Jedan pristup rešavanju ovog problema je ograničavanje maksimalne dubine stabla nakon koje algoritam učenja neće dalje razgranavati stablo. O načinu na koji se vrši izbor dubine biće reči kasnije, pošto se bira na sličan način kao vrednost regularizacionog hiperparametra  $\lambda$ . Zapravo, kako se ovom tehnikom smanjuje fleksibilnost modela u vreme učenja, ona se može smatrati upravo vidom regularizacije.



Slika 11.25: Stablo odlučivanja konstruisano pomoću ID3 algoritma.

## 11.8 Evaluacija modela i konfigurisanje algoritama učenja

Za primenu modela u praksi, od presudnog je značaja znati koliko dobro model generalizuje, odnosno koliko su mu dobra predviđanja. Za to je s jedne strane potrebno imati nekakve mere kvaliteta, s druge procedure koje obezbeđuju da se tako izračunatim merama može dovoljno verovati.

Kao što se moglo primetiti, algoritmi mašinskog učenja su konfigurabilni. Naime, moguće je podešavati regularizacioni hiperparametar kod linearnih modela,  $k$  kod algoritma  $k$  najbližih suseda, maksimalnu dubinu stabla, arhitekturu mreže, itd. Izbor različite konfiguracije vodi različitom optimizacionom problemu i stoga različitom rešenju. Poželjno je izabrati konfiguraciju koja vodi dobrom rešenju. Ovo ne mora biti trivijalan posao.

### 11.8.1 Mere kvaliteta regresije

Osnovna mera kvaliteta regresije je srednjekvadratna greška, koja meri odstupanje predviđenih vrednosti na nekom test skupu od stvarnih:

$$E(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N (y_i - f_{\mathbf{w}}(\mathbf{x}))^2$$

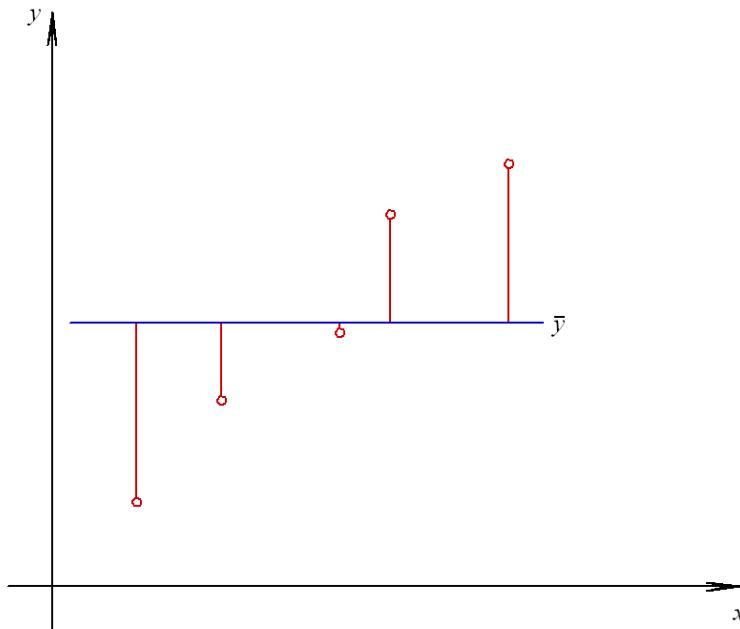
Obično je bolje posmatrati njen koren, pošto se izražava na istoj skali kao i veličina  $y$ , pa je taj broj razumljiviji u kontekstu domena. Poželjno je da srednjekvadratna greška bude što manja, međutim ukoliko nemamo konkretan zahtev za postizanjem određene srednjekvadratne greške, teško je reći da li je učenje uspešno ili ne. Zbog toga se često koriste i druge mere. Česta je upotreba *koeficijenta determinisanosti*. Koeficijent determinisanosti  $R^2$  između predviđenih i stvarnih vrednosti na test skupu, računa se na osnovu formule:

$$R^2(\mathbf{w}) = 1 - \frac{\sum_{i=1}^N (y_i - f_{\mathbf{w}}(\mathbf{x}))^2}{\sum_{i=1}^N (y_i - \bar{y})^2}$$

gde nadvučena linija označava prosek uzorka. Vrednost 1 koeficijenta determinisanosti označava potpuno podudaranje stvarnih i predviđenih vrednosti. Što je vrednost koeficijenta manja, to je poklapanje lošije. Često se kaže da  $R^2(\mathbf{w})$  predstavlja ideo varijanse vrednosti ciljne promenljive  $y$  koji model objašnjava. Smisao ovog tvrđenja je sledeći. Ukoliko se odrekнемo korišćenja bilo kakvih metoda predikcije pri predviđanju vrednosti  $y$ , najmanju srednjekvadratnu grešku očekujemo ukoliko uvek predviđamo vrednost  $\bar{y}$ , tj. ako kao prediktivni model za vrednosti  $y$  koristimo prosek opaženih vrednosti  $y_i$ . Pri tome je srednjekvadratna greška

$$E(\bar{y}) = \frac{1}{N} \sum_{i=1}^N (y_i - \bar{y})^2$$

odnosno uzoračka varijansa za  $y$ . Rastojanja koja ulaze u ovu grešku prikazana su na slici 11.26.



Slika 11.26: Grafik reziduala u zavisnosti od predviđenih vrednosti telesne težine.

U slučaju korišćenja linearog regresionog modela, srednjekvadratna greška ili varijansa vrednosti  $y$  u odnosu na model jednaka je

$$E(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N (y_i - \mathbf{w} \cdot \mathbf{x})^2$$

Rastojanja koja ulaze u ovu grešku prikazana su na slici 11.27.

Ova, preostala, varijansa se ne može objasniti zavisnošću od korišćenih svojstava, odnosno tu preostalu varijansu možemo smatrati neobjašnjrenom. Njen količnik sa  $E(\bar{y})$  može se onda smatrati udelom neobjašnjene varijanse, a koefficijent determinisanosti udelom objašnjene varijanse.

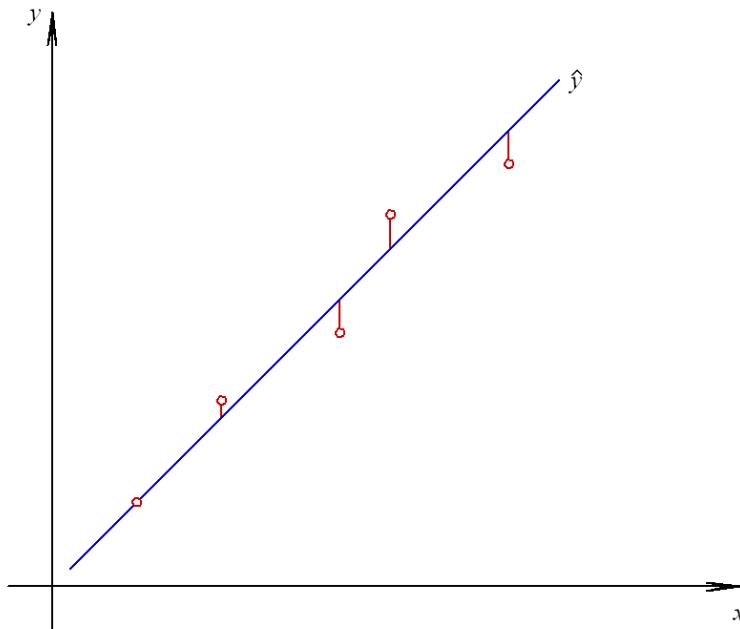
**Primer 11.14.** U primeru predviđanja telesne težine na osnovu visine, koefficijent determinisanosti iznosi 0.51 što znači da promenljiva  $x$  objašnjava oko pola varijanse promenljive  $y$  i da postoji prostor da se predikcija dalje popravi dodavanjem novih svojstava.

### 11.8.2 Mere kvaliteta klasifikacije

Kvalitet klasifikacije zavisi od broja i vrste grešaka koje klasifikator pravi. Polazna tačka u definisanju mera kvaliteta klasifikacije je prepoznavanje mogućih ishoda rada klasifikatora. U nastavku razmatramo slučaj binarne klasifikacije, ali je to razmatranje moguće generalizovati i na veći broj klasa. Pod *stvarno pozitivnim*instancama podrazumevamo pozitivne instance koje su prepoznate kao pozitivne. Pod *stvarno negativnim*, instance koje su negativne i prepoznate kao negativne. Pod *lažno pozitivnim* podrazumevamo instance koje su negativne, ali su greškom u klasifikaciji proglašene pozitivnim i pod *lažno negativnim* podrazumevamo instance koje su pozitivne, ali su greškom klasifikovane kao negativne. Brojeve ovih instanci označavamo redom  $SP$ ,  $SN$ ,  $LP$  i  $LN$ . Ove brojeve obično zapisujemo u takozvanoj *matrici konfuzije* koja po vrstama prikazuje kako su instance klasifikovane, a po kolonama prikazuje koje su stvarne klase instanci. Ova matrica je prikazana u vidu tabele 11.3.

U slučaju problema klasifikacije, zbog svoje intuitivnosti, najčešće korišćena mera kvaliteta je *tačnost* (eng. *accuracy*), odnosno broj tačno klasifikovanih instanci podeljen ukupnim brojem instanci, odnosno:

$$\text{tačnost} = \frac{SP + SN}{SP + LP + SN + LN}$$



Slika 11.27: Grafik reziduala u zavisnosti od predviđenih vrednosti telesne težine.

	P	N
KP	SP	LP
KN	LN	SN

Tabela 11.3: Vrste matrice konfuzije u zbiru daju broj instanci koje su klasifikovane pozitivno (KP) i broj instanci koje su klasifikovane negativno (KN). Kolone u zbiru daju broj instanci koje su zaista pozitivne (P) ili negativne (N).

U primeru sa prepoznavanjem računarskih članaka, koristili smo upravo tačnost kao meru kvaliteta.

U nekim slučajevima tačnost nije adekvatna mera. Ukoliko postoje dve klase i jedna je značajno manja od druge, moguće je dobiti visoku tačnost tako što će se sve instance klasifikovati u veću grupu. Takav je slučaj sa testovima koji ustanovljavaju da li je pacijent oboleo od neke bolesti. Ako bolest ima samo 1% ljudi u populaciji, test koji bi uvek prijavljivao da pacijent nema bolest imao bi tačnost od 99%, ali je neupotrebljiv. Otud se koriste i druge mere. Dve često korišćene mere su preciznost (neg. *precision*) i odziv (eng. *recall*). Preciznost se definiše kao udeo (zaista) pozitivnih instanci među onima koje su klasifikovane kao pozitivne, odnosno

$$prec = \frac{SP}{SP + LP}$$

dok se odziv definiše kao udeo instanci koje su klasifikovane kao pozitivne među onima koje zaista jesu pozitivne, odnosno

$$odziv = \frac{SP}{SP + LN}$$

Intuitivno, prva mera nam kaže koliko često smo u pravu kada tvrdimo da je neka instanca od interesa, dok druga govori koliki udeo instanci od interesa uspevamo da detektujemo. Poželjno je da ove vrednosti budu što veće. Svaka od ove dve mere lako se maksimizuje odvojeno, te je uvek potrebno razmatrati ih zajedno. Ako nikad ne tvrdimo da je neka instanca pozitivna, nikad ne grešimo u pozitivnoj klasifikaciji i preciznost je jednaka 1 (implicitno definišemo da je u ovom kontekstu 0/0 jednako 1), ali će tada odziv biti 0. S druge strane, ako sve instance proglašimo za pozitivne odziv je jednak 1, ali će preciznost biti mala (jednaka udelu pozitivne klase). Kako je ponekad teško razmatrati dve mere paralelno, posebno kada se želi poređenje dva ili više različitih modela, često se razmatra njihova harmonijska sredina, takozvana mera  $F_1$ . Harmonijska sredina bliža je nižoj od dve vrednosti koje se usrednjavaju, tako da se na taj način dobija stroža ocena od

aritmetičke sredine. Primer konteksta u kojem se ove mera koristi je pronalaženje dokumenata iz neke kolekcije koji odgovaraju nekom upitu, na primer prilikom pretrage na vebu. Ove mera korisna je i jer ne daju, poput tačnosti, nerealističnu sliku u slučaju neizbalansiranih klasa.

**Primer 11.15.** Neka je testirano 400 ispitanika. Od toga 5 imaju bolest zbog koje se testiraju, a ostali ne. Neka je test dao pozitivnu klasifikaciju u slučaju 3 osobe koje sve tri imaju bolest. Matrica konfuzije data je tabelom 11.4

Odavde se mogu izračunati mere kvaliteta:

$$\text{tačnost} = \frac{398}{400} = 0.995$$

$$\text{prec} = \frac{3}{3} = 1$$

$$\text{odziv} = \frac{3}{5} = 0.6$$

$$F_1 = 2 \frac{1 \cdot 0.6}{1 + 0.6} = 0.75$$

Tačnost je izuzetno visoka i sugerije odlično ponašanje klasifikatora. Preciznost sugerije isto, ali poznavanje udela stvarno pozitivnih otkriva da je ovaj utisak lažan zato što je identifikovan mali broj elemenata pozitivne klase. Kako  $F_1$  mera uzima u obzir i preciznost i odziv i po njoj se vidi da klasifikator ipak nije dobar kao što deluje na osnovu tačnosti.

	P	N
KP	3	0
KN	2	395

Tabela 11.4: Matrica konfuzije vezana za klasifikaciju obolelih.

	P	N
KP	3	5996
KN	1	94000

Tabela 11.5: Matrica konfuzije vezana za klasifikaciju aerodromskih putnika.

**Primer 11.16.** Pretpostavimo da je potrebno razviti sistem koji prepoznae potencijalne teroriste među putnicima na nekom aerodromu. Kroz aerodrom prođe dnevno oko 100000 putnika, a očekivani broj potencijalnih terorista je, na primer, 4. Sistem procenu pravi na osnovu više informacija, uključujući starosnu dob putnika, državljanstvo, zemlje iz koje putuje i zemlje u koju putuje, arhivu ekstremističkih pokreta i slično. Svaki putnik označen kao potencijalni terorista mora da prođe kroz detaljnju dodatnu proveru.

Ako bi sistem trivijalno za svakog putnika davao odgovor „ne“ (tj. procenu da nije potencijalni terorista), onda bi sistem imao odličnu tačnost ( $99996/100000 = 99.996\%$ ), ali je jasno da je takav sistem potpuno nekoristan. Upotrebljivost ovakvog sistema mora se proveriti kroz druge mere kvaliteta. U ovom slučaju, preciznost je savršena (jednaka 1) jer sistem nikoga nije pogrešno označio kao teroristu, ali odziv je jednak 0, što jasno ukazuje da je sistem beskorisan.  $F_1$  mera je takođe jednak 0.

Upotrebljivost ima različite aspekte i ponekad suprotstavljene zahteve. U ovom konkretnom problemu, sa aspekta bezbednosti, potrebno je da odziv bude jednak 1 ili što bliži toj vrednosti. Da bi se to obezbeđilo, potrebno je sprovoditi dodatne provere nad što većim brojem putnika. Međutim, s druge strane, sa aspekta aerodromske logistike i ugođaja putnika, potrebno je da i preciznost bude što veća, jer nije moguće organizovati dodatni pregled za veliki broj putnika, niti je poželjno izlagati putnike neprijatnosti i gubitku vremena.

Pretpostavimo da je ponašanje sistema opisano matricom konfuzije koja je data u tabeli 11.5. Odatle se mogu izračunati mere kvaliteta:

$$tačnost = \frac{94003}{100000} = 0.94003$$

$$prec = \frac{3}{5999} \approx 0.0005$$

$$odziv = \frac{3}{4} = 0.75$$

$$F_1 = 2 \frac{0.0005 \cdot 0.75}{0.7505} \approx 0.001$$

Tačnost je lošija nego kod prethodnog, trivijalnog sistema, ali i dalje deluje visoka. Ipak, poređenjem  $F_1$  mera moglo bi se zaključiti da ovaj sistem nije mnogo bolji od prethodnog sistema, ali se razlozi bolje vide analizom ostalih mera. Preciznost je očito i dalje vrlo niska, a ni odziv verovatno nije dovoljno visok za praktičnu upotrebljivost ovog sistema.

### 11.8.3 Tehnike evaluacije i konfigurisanja algoritama učenja

Pored izbora mera kvaliteta, bitno je izabrati i način na koji se ta mera ocenjuje. Česta praksa je da se model trenira na jednom skupu podataka, a da se evaluira na odvojenom skupu podataka za testiranje. Pritom se podela raspoloživih podataka na podatke za trening i podatke za testiranje vrši slučajnim izborom podataka za testiranje. Međutim, ovakav način evaluacije može dovesti do značajnih oscilacija u vrednostima mera kvaliteta u zavisnosti od toga koji je podskup izabran. Pouzdaniji način evaluacije naučenog znanja je takozvana *unakrsna validacija* (eng. *cross-validation*). Ceo skup podataka kojim se raspolaze se deli na  $n$  približno jednakih podskupova. Jeden podskup se izdvaja i trening se vrši na ostalih  $n - 1$  podskupova. Posle treninga, kvalitet naučenog znanja se ocenjuje 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. Pored toga, prednost ovog metoda je da se u svakom od  $n$  koraka unakrsne validacije koristi velika količina podataka pri treniranju, a da sve raspoložive instance u jednom trenutku budu iskorišćene za testiranje.

Iako nije očigledno da je u vezi sa tehnikama evaluacije, u nastavku će biti razmotreno dugo odlagano pitanje izbora vrednosti regularizacionog hiperparametra  $\lambda$ , kao i broja suseda  $k$  i maksimalne dubine stabla za koje smo najavili da ćemo ih razmatrati skupa. Izbor vrednosti ovih parametara predstavlja samo primer opštijeg problema konfigurisanja algoritama učenja. U opštem slučaju, algoritmi učenja se mogu podešavati na različite načine, pri čemu različite konfiguracije daju različite modele za iste ulazne podatke. U nastavku će biti reči samo o izboru parametra  $\lambda$ , ali se diskusija odnosi i na probleme konfigurisanja algoritama učenja u opštijem smislu.

Za dati skup podataka, svakoj vrednosti parametra  $\lambda$  odgovara neka vrednost optimalnih koeficijenata  $\mathbf{w}_\lambda$  i samim tim neki model  $f_{\mathbf{w}_\lambda}(\mathbf{x})$ . Postavlja se pitanje koji od ovih modela je najbolji. Osim ako nije raspoloživa velika količina podataka, male vrednosti parametra  $\lambda$  uzrokuju loše rezultate zbog preprilagođavanja, a velike vrednosti uzrokuju loše rezultate zbog premale fleksibilnosti modela. Poželjne vrednosti parametra se obično nalaze negde između dva ekstrema. Stoga je prvi korak u pronalaženju pogodne vrednosti određivanje granica intervala u kojem će se vrednost tražiti, što se može uraditi eksperimentalno. Recimo interval  $[10^{-10}, 10^5]$  je verovatno dovoljno širok u većini slučajeva, ali to je ipak potrebno detaljnije razmotriti u konkretnom slučaju. Potom se formira niz vrednosti parametra koje se ispituju. Na primer, često se koristi geometrijska progresija  $\lambda_1 = 10^{-10}, \lambda_2 = 10^{-9}, \dots, \lambda_{11} = 10^5$ . Potom se, grubo rečeno, model za svaku od tih vrednosti evaluira i bira se najbolji. Ipak, postavlja se pitanje kako se vrši evaluacija.

Prva ideja bi bila da se za svaku od izabranih vrednosti parametra izvrši treniranje na trening skupu i da se dobijeni model evaluira na test skupu nekom merom kvaliteta i da se izabere najbolji od njih. Pažljivijim razmatranjem se uviđa da je ovaj postupak pogrešan. Naime, na ovaj način se podaci iz test skupa koriste pri izboru modela, što je sve deo treninga i samim tim i oni predstavljaju deo trening skupa. Međutim, disjunktnost trening i test skupa je osnovno pravilo evaluacije modela u mašinskom učenju. Korektan postupak bi bio da se umesto podele ukupnog skupa podataka na trening i test skup izvrši njegova podela na trening skup, validacioni skup i test skup. Tada se na trening skupu vrši treniranje svakog od modela (dobijenih za različite vrednosti  $\lambda_i$ ), na validacionom skupu se vrši evaluacija na osnovu koje se bira najbolji model i potom se taj model evaluira na test skupu i njegov kvalitet predstavlja finalnu meru kvaliteta učenja.

Prethodni postupak je korektan, ali ponovo se konstatuje da usled slučajnog deljenja podataka na trening i test skup, može doći do različitih ishoda evaluacije. Prethodno je ovaj problem rešen pomoću unakrsne validacije i varijanta te tehnikе se može primeniti i u ovom slučaju, ali kako je taj pristup osetno komplikovaniji, o njemu

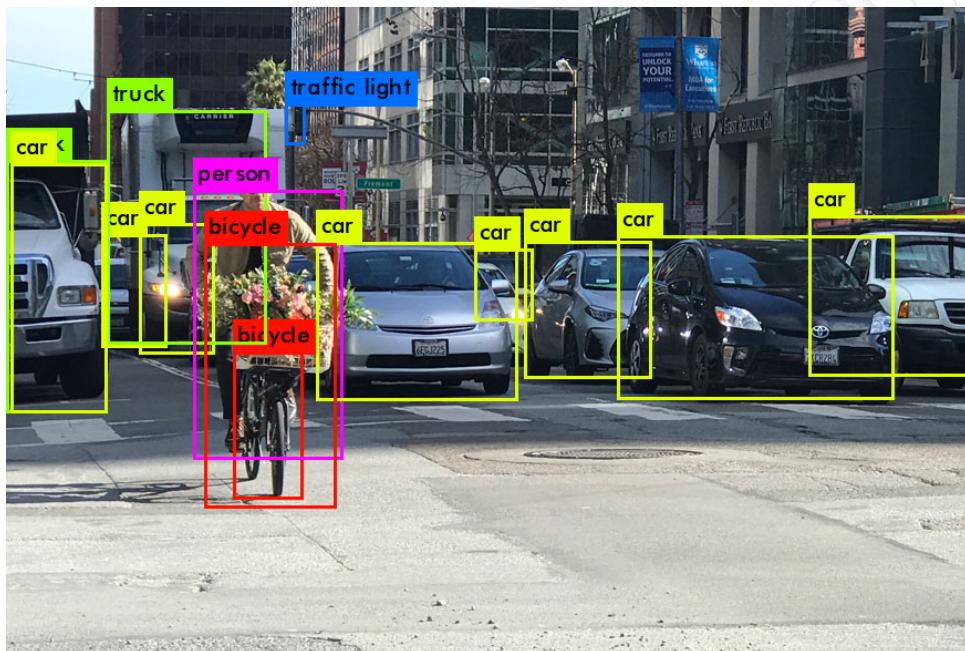
neće biti reči.

## 11.9 Primeri primena nadgledanog učenja

Primeri praktičnih primena nadgledanog učenja su u poslednjoj deceniji sve brojniji. Najuzbudljivije i praktično najvažnije primene odnose se na neuronske mreže, pa će i akcenat u dalje tekstu biti na njima, pre svega na obradi slika i teksta. Ipak, biće diskutovane i primene drugih metoda.

### 11.9.1 Detekcija objekata na slikama

Prepoznavanje objekata na slikama je problem od velikog praktičnog značaja. U autonomnoj vožnji bitno je prepoznati druge učesnike u saobraćaju, saobraćajne znake i slično. Prepoznavanje lica može služiti identifikaciji ljudi u različite svrhe (koje se mogu smatrati legitimnim ili ne). Pristupi rešavanju ovih problema su raznovrsni. Ovde će biti diskutovan jedan pristup, poznatiji kao YOLO (*you only look once*), poznat po svojoj brzini i preciznosti.<sup>8</sup> Primer detekcije objekata dat je na slici 11.28.



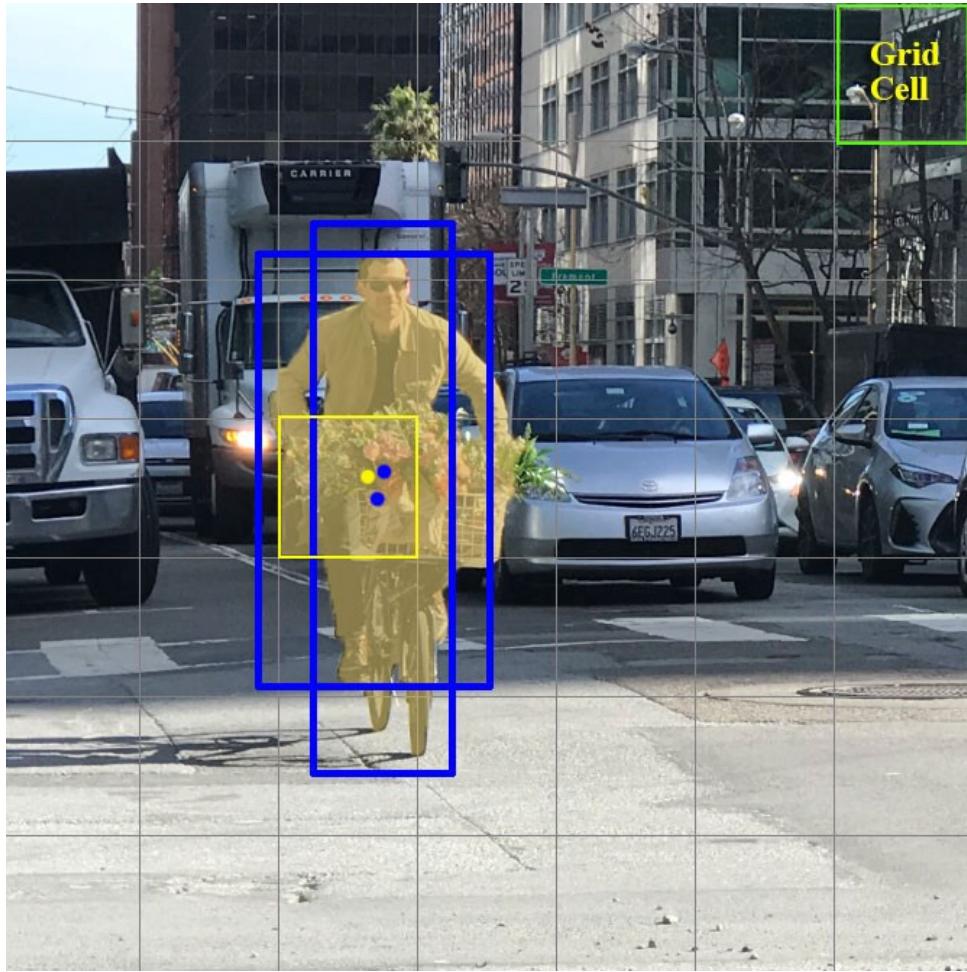
Slika 11.28: Detekcija objekata na slici pomoću YOLO mreže.

YOLO prepostavlja da se slika deli na  $S \times S$  pravougaonih ćelija. Pojednostavljeno, svaka ćelija zadužena je za detekciju jednog objekta čiji je centar u njoj (mada se ova veza ne nameće eksplicitnim uslovima). Za ćeliju je vezan konvolutivni klasifikator kojim se objekat prepoznat u toj ćeliji klasificuje u jednu od kategorija, pritom dajući informaciju o verovatnoći te klasifikacije. Pored toga, za svaku ćeliju vezano je  $B$  regresionih konvolutivnih modela čija je svrha da predvide pouzdanost pomenute detekcije objekta, kao i pravougaonik<sup>9</sup> koji sadrži ceo objekat od značaja (za razliku od pravougaonika koji definiše ćeliju i koji treba da sadrži centar objekta). Pravougaonik se predviđa predviđanjem  $x$  i  $y$  koordinata centra pravougaonika i njegove širine i visine. Pouzdanost predviđanja definije se kao proizvod dve relevantne veličine. Prva je verovatnoće koju daje klasifikator i koja odražava sigurnost klasifikatora u predloženu klasu objekta. Druga je količnik površina preseka i unije predviđenog pravougaonika i stvarnog pravougaonika koji obuhvata objekat na slici – IoU (eng. *intersection over union*). Ona odražava preciznost detekcije. Na ovaj način se pojedinačne pouzdanosti klasifikacije i preciznosti detekcije objedinjuju u jedan skor koji odražava ukupan kvalitet detekcije, koji mreža takođe predviđa. Iz prethodnog opisa je jasno da broj obojkovata koje YOLO može da detektuje zavisi od broja ćelija  $S \times S$ , pa je taj broj značajan hiperparametar. Primer pravougaonika vezanih za ćeliju i gruba shema modela vezanog za ćeliju mogu se videti na slikama 11.29 i 11.30.

Prilikom primene modela, lako može doći do značajnog preklapanja velikog broja pravougaonika koji mogu prepoznavati isti objekat, pa i različite, ali sa različitim pouzdanostima. Heuristika koja se koristi kako bi se

<sup>8</sup>Ovaj sistem ima više verzija. Akcenat nije na njegovim najmodernijim detaljima, već na osnovnom principu.

<sup>9</sup>Podrazumeva se da su strane pravougaonika paralelne koordinatnim osama.

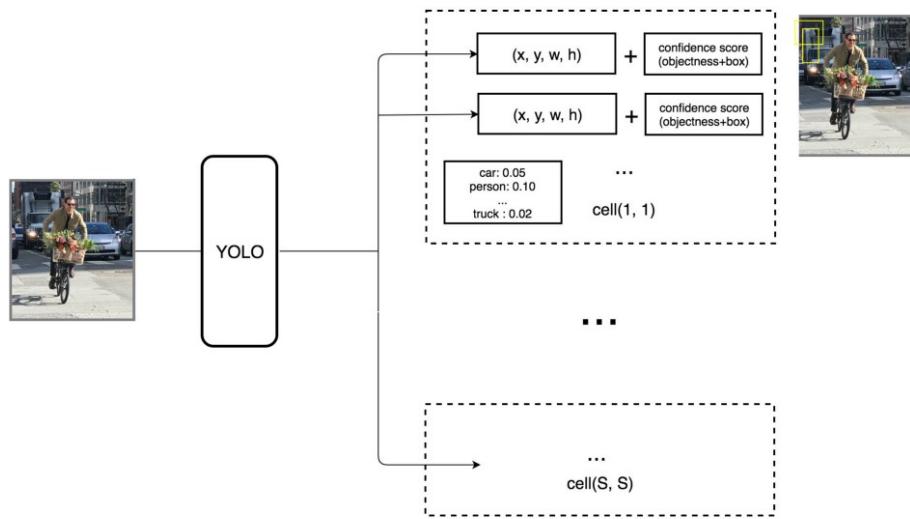


Slika 11.29: Ilustracija prvougaonika vezanih za jednu ćeliju.

otklonio ovaj problem podrazumeava sortiranje predloženih pravougaonika po pouzdanosti i njihovu selekciju na sledeći način. Bira se pravougaonik sa najvećom pouzdanošću koji se zadržava, dok se odbacuju svi pravougaonici čiji je IoU skor sa tim pravougaonikom veći od nekog unapred odabranog praga. Potom se postupak nastavlja nad preostalim predloženim pravougaonicima dok se svi ne obrade. Izabrani pravougaonici čine predviđanje modela.

YOLO objedinjuje veći broj klasifikacionih i regresionih modela. Za svaku ćeliju postoji više regresionih modela i jedan klasifikacioni. Regresioni modeli u jednoj ćeliji obično se ne slažu i ne bi ni imalo smisla da se slažu jer u bi u tom slučaju bio dovoljan jedan. Postavlja se pitanje kako ovakav model treba trenirati. Pre svega, podaci treba da budu obeleženi tako da su dati svi pravougaonici od značaja i tačne klase objekata u tim pravougaonicima. Za svaki od datih pravougaonika postoji regresioni model koji daje pravougaonik koji se najviše preklapa sa datim pravougaonikom. Taj model je odgovoran za dati pravougaonik. Informacija o grešci u odnosu na dati pravougaonik uticaće samo na model koji je odgovoran za njega. To dovodi do toga će se različiti modeli u vezani za istu ćeliju specijalizovati za objekte različitih dimenzija (neki za male, neki za velike, neki za horizontalno izdužene, neki za vertikalno izdužene i slično).

Definišimo funkciju greške. Oznaka  $\mathbb{1}_{ij}$  označava vrednost 1 ako je za objekat u ćeliji  $i$  (ćelija može detektovati najviše jedan objekat) odgovoran  $j$ -ti regresioni model, a 0 u suprotnom. Oznaka  $\mathbb{1}_i$  slično označava da se u ćeliji  $i$  nalazi objekat. Vrednosti  $x_i$  i  $y_i$  su stvarne koordinate centara pravougaonika koji obuhvataju objekte, a  $\hat{x}_i$  i  $\hat{y}_i$  predviđene i analogno za visinu i širinu  $h_i$  i  $w_i$ . Za klasu  $j$ ,  $p(j|i)$  označava predviđenu verovatnoću klase  $j$  za objekat u  $i$ -toj ćeliji (može imati proizvoljnu vrednost ukoliko se u toj ćeliji ne nalazi objekat).  $N_C$  je ukupan broj klasa.  $c_{ij}$  je 1 ukoliko objekat u ćeliji  $i$  stvarno pripada klasi  $j$ , a 0 u suprotnom (može imati proizvoljnu vrednost ukoliko se u toj ćeliji ne nalazi objekat).  $C_i$  označava stvarnu pouzdanost za predviđeni pravougaonik (IoU skor stvarnog i predviđenog pravougaonika pomnožen verovatnoćom najverovatnije klase),



Slika 11.30: Shema modela pridruženog ćeliji.

a  $\hat{C}_i$  predviđenu pouzdanost. Greška modela data je izrazom:

$$\begin{aligned}
 & \lambda_{bb} \sum_{i=1}^{S^2} \sum_{j=1}^B \mathbb{1}_{ij} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
 & + \lambda_{bb} \sum_{i=1}^{S^2} \sum_{j=1}^B \mathbb{1}_{ij} \left[ (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
 & + \lambda_{conf} \sum_{i=1}^{S^2} \sum_{j=1}^B \mathbb{1}_{ij} (C_i - \hat{C}_i)^2 \\
 & + \lambda_{nconf} \sum_{i=1}^{S^2} \sum_{j=1}^B (1 - \mathbb{1}_{ij})(C_i - \hat{C}_i)^2 \\
 & + \lambda_{class} \sum_{i=1}^{S^2} \mathbb{1}_i \sum_{j=1}^{N_C} c_{ij} \log p(j|i)
 \end{aligned} \tag{11.6}$$

Poslednji sabirak predstavlja uopštenje unakrsne entropije na proizvoljan broj klasa. Hiperparametri  $\lambda$  omogućavaju podešavanje težina različitih delova greške. Primetimo da su težine greške pouzdanosti različite za regresione modele koji su odgovorni za neke objekte i za one koji to nisu. Kako će većina ćelija obično biti prazna, a među onima koje sadrže objekat, samo će jedan model biti odgovoran za njega, ovo omogućava da se izbalansiraju redovi veličine ovih izraza. I na kraju, dimenzije pravougaonika su korenovane kako bi se povećala preciznost u detekciji malih objekata. Naime, ukoliko koren ne bi bio primjenjen, greške u predviđanju dimenzija velikih objekata bi drastično dominirale u odnosu na greške u predviđanju dimenzija malih objekata, što bi značajno narušilo preciznost njihove detekcije.

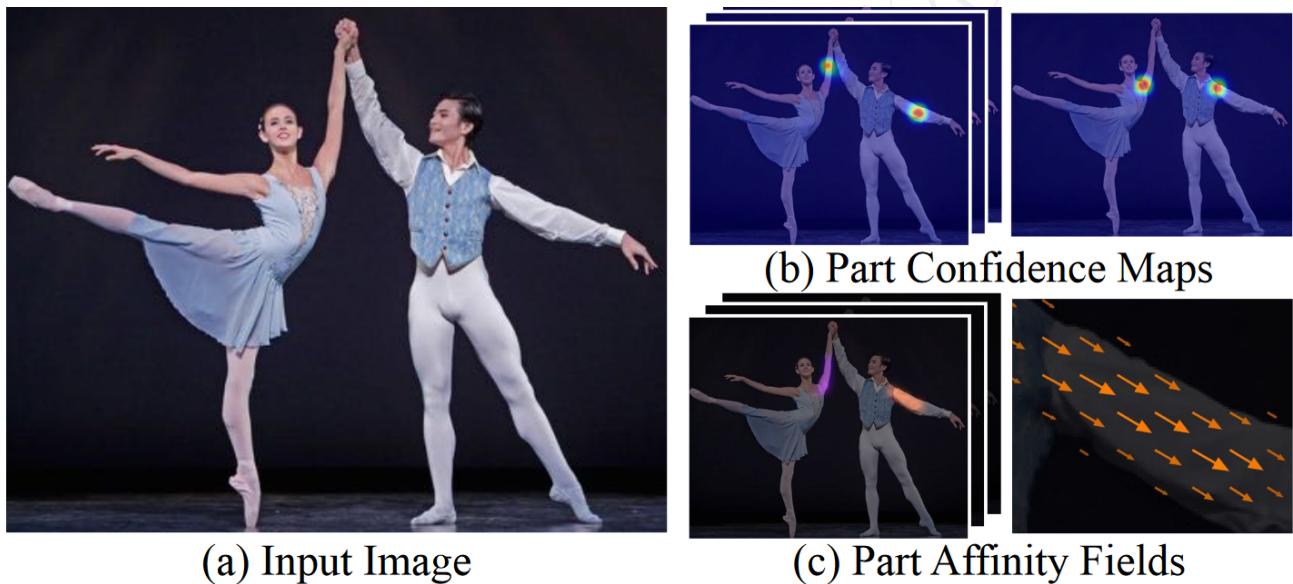
Da sumiramo, svaka ćelija detektuje jedan objekat za koji predviđa klasu, pravougaonik u kojem se nalazi i skor pouzdanosti. Dizajnirana je posebna funkcija gubitka koja kombinuje greške koje se prave u klasifikaciji, lokalizaciji objekta i u proceni pouzdanosti detekcije. U ovom opisu nismo diskutovali konkretnu arhitekturu mreže pošto detalji arhitekture mogu varirati od primene do primene.

### 11.9.2 Prepoznavanje poze sa slike konvolutivnim mrežama

Jedna od primena konvolutivnih mreža je i prepoznavanje poza u kojima se osobe nalaze na slikama. Primer je dat na slici 11.31. Kao što se može i prepostaviti, jedan od ključnih elemenata u rešavanju ovog problema je određivanje pozicija zglobova. Ipak, pozicije zglobova nisu dovoljne za određivanje poze. Naime, pitanje je koji zglob treba povezati sa kojim. Čak i u slučaju da se na slici nalazi samo jedna osoba i da je poznata je



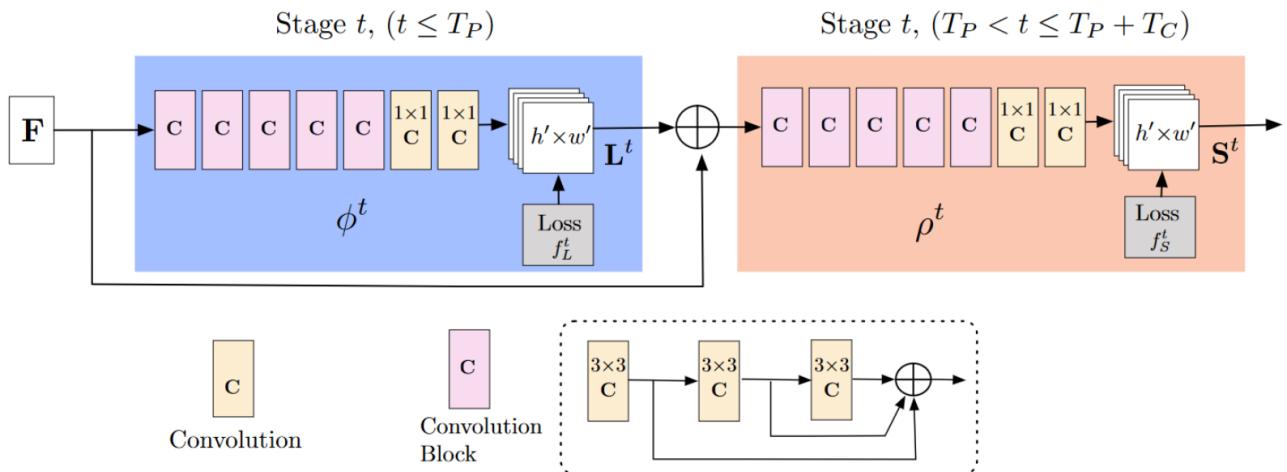
Slika 11.31: Primer detekcije poze na slici.



Slika 11.32: Primer mapa pouzdanosti za dve vrste zgoba i vektorskog polja koje definije orientaciju dela tela koji ih povezuje.

lokacija svih njenih zglobova, iako je jasno da treba povezati njeno koleno sa člankom, u nekim pozama članak jedne noge može biti bliži kolenu druge, pa može doći do greške. U slučaju da slika sadrži veći broj osoba, ovaj problem još je izraženiji. Zato je, pored detekcije zglobova, potrebno izvršiti i predviđanje orijentacije delova tela (podlaktice, potkolenice, itd).

Metod OpenPose vrši prepoznavanje poze tela u dve faze. U prvoj fazi određuje pozicije zglobova i orientacije delova tela, a u drugoj fazi povezuje zglobove i tako vrši prepoznavanje poze. Na mašinskom učenju zasnovana je ova prva faza i ona će biti grubo opisana u nastavku. Koristi se model koji se trenira na mnoštvu slika i ručno napravljenih izlaza — slika na kojima su pogodno označene pozicije zglobova i orientacije delova tela. Za svaku novu ulaznu sliku, model daje dve vrste izlaza: *mapa pouzdanosti i vektorska polja*. Pozicije zglobova određuju se mapama pouzdanosti, a orientacije delova tela vektorskim poljima. Postoji po jedna mapa pouzdanosti za svaku vrstu zgloba i to je slika istih dimenzija kao sto je ulazna. Konkretno, mapa koja odgovara levom laktu detekuje leve laktove svih ljudi na zadatoj ulaznoj slici, a mapa koja odgovara levom ramenu detektuje sva leva ramena na zadatoj ulaznoj slici. Mapa pouzdanosti ima visoku vrednost тамо где је највероватније да се на slici nalazi konkretni zglob, а ниску тамо где то nije вероватно. Postoji po jedno vektorsko polje за svaku deo tela (на primer, desna podlaktica) и оно svakom pikselu ulazne slike pridružuje dvodimenzionalni vektor koji predstavljaju koordinate vektora koji određuje kako je orijentisan deo tela kojem taj piksel pripada. Ovo je ilustrovano na slici 11.32.



Slika 11.33: Arhitektura OpenPose mreže.

Arhitektura mreže koja vrši prepoznavanje poze ilustrovana je slikom 11.33. Kasnije će biti dat njen formalniji opis. Modul  $\mathbf{F}$  predstavlja konvolutivnu mrežu opšte namene koja je unapred trenirana za klasifikaciju slika nevezanu za ovaj problem (postoje različite takve mreže). Ipak, ona je u stanju da prepozna neka svojstva slika koja su se u problemu klasifikacije pokazali kao relevantna i očekuje se da će ostatak treninga biti kraći ako se za polaznu tačku uzme takva mreža. Poslednji klasifikacioni sloj mreže je uklonjen i poslednji izlaz se povezuje sa ostatkom sistema i trenira zajedno sa njim.

Sledeći deo arhitekture, koji se nadovezuje na mrežu koja izdvaja svojstva, služi za predviđanje vektorskih polja koja definišu orientaciju delova tela. Ovo predviđanje vrši se pomoću nekoliko nadovezanih modula iste strukture (ali koji imaju različite vrednosti parametara) koji na svojim izlazima za svaki piksel daju predviđanje dve koordinate vektora pravca u kojem se pruža deo tela na kojem se taj piksel nalazi. Zanimljivo je da svaki od ovih modula na kraju ima izračunavanje funkcije greške, koje se sve zajedno minimizuju. Ovo nije intuitivno, pošto obično očekujemo da model daje takva predviđanja u poslednjem sloju. Ipak, poznato je da duboke mreže imaju problem nestajućih gradijenata (gradijenti bivaju bliski nuli), što vodi izrazito sporom treningu. Otud se pribegava izračunavanju grešaka i na nižim nivoima, čime se obezbeđuje da ako je doprinos gradijentu sa viših nivoa nizak, ipak postoji doprinos sa nižih nivoa. Time se mreža tera da već na nižim nivoima uči aproksimacije vektorskih polja, koje možda nisu dovoljno dobre, ali mogu poslužiti kao gruba aproksimacija koju će naredni slojevi profiniti.

Slično je organizovan i ostatak mreže koji ponovo pomoću više nadovezanih modula određuje mape pouzdanosti lociranja zglobova, tako što na izlazima svakom pikselu pridružuje jedan broj koji je utoliko veći što je pouzdanost lociranja zgloba na tom mestu veća. Primetimo da svi moduli mreže pored izlaza prethodnog modula kao ulaze dobijaju i svojstva  $\mathbf{F}$  koja nose informacije koje su možda izgubljene pri transformacijama koje vrše prethodni moduli.

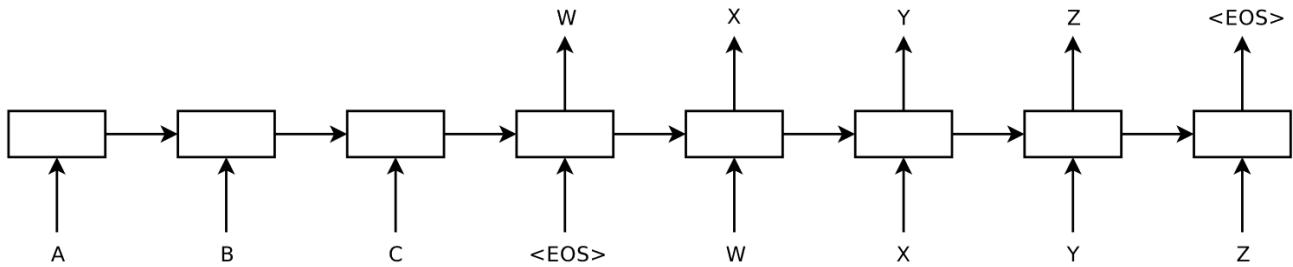
Kao funkcija greške za obe vrste predviđanja koristi se kvadratna greška između predviđene i stvarne vrednosti kako za mapu pouzdanosti, tako i za vektorska polja koja određuju orientaciju delova tela.

U drugoj fazi, metod OpenPose na osnovu dobijenih mapa pouzdanosti i vektorskih polja određuje šta se sa čime povezuje. To nije trivijalno uraditi čak ni kad su raspoložive savršene mape pouzdanosti i vektorska polja. Ovaj problem može se modelovati kao NP-težak problem kombinatorne optimizacije (gde je parametar problema ukupan broj zglobova svih osoba na slici). Za njega postoji efikasno aproksimativno rešenje čiji je kvalitet dovoljan za praktične potrebe, ali neće biti ovde opisan.

### 11.9.3 Mašinsko prevodenje i srodnici problemi

Rekurentne neuronske mreže najčešće nalaze svoju primenu u obradi prirodnog jezika. Jedna od najzanimljivijih takvih primena je mašinsko prevodenje teksta sa jednog jezika na drugi. Prvo će biti opisan način na koji se vrši mašinsko prevodenje pomoću rekurentnih mreža, a onda će biti dat osvrt na to kako se na sličnim principima mogu rešiti problemi sumiranja teksta i opisivanja slika tekstom.

Osnova mašinskog prevodenja pomoću neuronskih mreža je enkoder-dekoder arhitektura. Sastoji se od dve LSTM mreže. Prva se naziva enkoderom i uloga joj je da pročita ulaznu rečenicu (uključujući znak za kraj rečenice) i da u poslednjem koraku pruži njenu reprezentaciju u vidu vektora fiksne dimenzije – svog skrivenog stanja. Druga se naziva dekoder i uloga joj je da polazeći od tog vektora kao svog inicijalnog skrivenog



Slika 11.34: Ilustracija enkoder-dekoder arhitekture.

stanja generiše prevod polazne rečenice na drugi jezik (uključujući znak za kraj rečenice). Ova arhitektura (u razmotranom obliku) ilustrovana je na slici 11.34.

Ako je  $\mathbf{x}_1, \dots, \mathbf{x}_T$  polazna rečenica,  $\mathbf{c}$  reprezentacija generisana od strane enkodera, a  $\mathbf{y}_1, \dots, \mathbf{y}_{T'}$  tačan prevod, učenje se vrši maksimizovanjem verodostojnosti parametara  $\mathbf{w}$ :

$$P_{\mathbf{w}}(\mathbf{y}_1, \dots, \mathbf{y}_{T'} | \mathbf{x}_1, \dots, \mathbf{x}_T) = \prod_{i=1}^{T'} P_{\mathbf{w}}(\mathbf{y}_t | \mathbf{c}, \mathbf{y}_1, \dots, \mathbf{y}_{t-1})$$

Pritom, raspodela  $P_{\mathbf{w}}(\mathbf{y}_t | \mathbf{v}, \mathbf{y}_1, \dots, \mathbf{y}_{t-1})$  se definiše pomoću funkcije *softmax* (definisane formulom 11.3) nad izlazima mreže. Naravno, kao i ranije, ne maksimizuje se verodostojnost, već se minimizuje negativna vrednost logaritma verodostojnosti u odnosu na neki skup parova rečenica na dva različita jezika.

U ranim eksperimentima sa ovim modelom, ustanovljeno je da lakše uči kada se obrne redosled reči ulazne rečenice, pošto je za generisanje prvih reči prevoda potrebno imati informacije iz prvih reči ulazne rečenice, koje su pouzdano sačuvane u skrivenom stanju enkodera ukoliko se te reči pojave na kraju. Naravno, ovo znači da će generisanje poslednjih reči prevoda biti teže. Treba imati u vidu da je ovaj problem svojstven i čoveku. Naime, kada pročitamo rečenicu na jednom jeziku, ako je ona duga i komplikovana, nije nam lako da je celu prevedemo odjednom na drugi jezik. Obično ćemo u toku prevođenja s vremenom na vreme obraćati pažnju na različite delove ulazne rečenice dok ne kompletiramo prevod. Ovo je inspirisalo mehanizam pažnje (eng. *attention*) koji predstavlja nezaobilazan deo neuronskih modela za obradu prirodnog jezika. Objasnićemo ga detaljnije. Neka su  $\mathbf{h}_1, \dots, \mathbf{h}_T$  skrivena stanja enkodera. Prethodni model koristi samo poslednje stanje dekodera i koristi ga da inicijalizuje prvo stanje enkodera. Umesto toga, mehanizam pažnje prepostavlja da se izlaz generiše na osnovu stanja dekodera  $\mathbf{s}_i$  i vektora konteksta  $\mathbf{c}_i$  koji se računa u svakom koraku na sledeći način:

$$\mathbf{c}_i = \sum_{j=1}^T \frac{\exp(a_{\mathbf{w}}(\mathbf{s}_i, \mathbf{h}_j))}{\sum_{k=1}^T \exp(a_{\mathbf{w}}(\mathbf{s}_i, \mathbf{h}_k))} \mathbf{h}_j$$

Primetimo da su vrednosti

$$\frac{\exp(a_{\mathbf{w}}(\mathbf{s}_i, \mathbf{h}_j))}{\sum_{k=1}^T \exp(a_{\mathbf{w}}(\mathbf{s}_i, \mathbf{h}_k))}$$

nenegativne i da u zbiru daju 1. Otud vektor  $\mathbf{c}_i$  predstavlja težinski prosek stanja enkodera. Kojim stanjima će biti data veća težina, odnosno na koje delove ulazne rečenice će biti obraćena pažnja, zavisi od vrednosti  $a_{\mathbf{w}}(\mathbf{s}_i, \mathbf{h}_j)$ . Funkcija  $a_{\mathbf{w}}$  predstavlja potpuno povezanu neuronsku mrežu čija je uloga da prepozna relevantna stanja enkodera  $\mathbf{h}_j$  za dato stanje dekodera  $\mathbf{s}_i$  i da tada da veću vrednost. Kako sada dekoder ima pristup svim skrivenim stanjima enkodera i u svakom koraku mehanizam njihovog izbora, posao enkodera je značajno olakšan – više nije neophodno da sve relevantne informacije, uključujući one sa početka rečenice, budu sačuvane u jednom vektoru.

Za treniranje ovog modela korišćen je dvojezični paralelni korpus koji svaku rečenicu uparuje sa njenim prevodom. Ukupna veličina korpusa bila je 348 miliona reči. Korupsi pomoću kojih se treniraju najnoviji modeli za prevođenje i razne druge poslove obrade prirodnog jezika su značajno veći.

Vrlo slični modeli predstavljaju osnovu i za problem sumiranja teksta – generisanje kratkih sažetaka teksta koji sadrže najvažnije informacije iz njega. Taj problem može se smatrati problemom mašinskog prevođenja na isti jezik sa gubitkom. Otud i sličnost osnovnog mehanizma.

Još jedan srođan problem je opisivanje slika. Naime, o tom problemu može se razmišljati kao o problemu prevođenja sa „jezika slike“ na, recimo, srpski jezik. Pritom se dekoder implementira pomoću konvolutivne mreže koja predstavlja prirodan izbor za obradu slika. Poslednji sloj mreže predstavlja skriveno stanje koje se predaje dekoderu.

Ovi primeri ukazuju na jedan zanimljiv opštiji zaključak – skriveno stanje predstavlja značenje ulaznog sadržaja. Enkoder ekstrahuje to značenje iz ulazne reprezentacije, a dekoder ga transformiše u izlaznu reprezentaciju. Postoji još argumenata koji opravdavaju razmišljanje o vektorskim reprezenzacijama koje daju skrivena stanja enkodera kao o značenju i javljaju se i u mnogim drugim kontekstima. Na primer, uočeno je da je vektorska aritmetika nad tim reprezentacijama često saglasna sa značenjima ulaznih objekata. Konkretno, ako su **a**, **b**, **c**, **d** vektorske reprezentacije reči (dobijene iz enkodera) Pariz, Francuska, Rim i Italija, ispostavlja se da važi  $\mathbf{a} - \mathbf{b} + \mathbf{d} \approx \mathbf{c}$ , iako enkoder uopšte nije treniran da prepozna relaciju glavni grad, vec je treniran na velikom neobeleženom skupu podataka samo pokušavajući da za svaku reč što bolje pogodi reči u njenoj okolini pri svakom njenom pojavljivanu u tekstu. Štaviše, isto zapažanje važi i za mnoge druge relacije i čak je na ovaj način moguće odgovarati na pitanja o analogijama. Ovo i druga slična zapažanja stvorila su novu podoblast obrade prirodnog jezika – razumevanje prirodnog jezika, ali dalja diskusija na tu temu izlazi iz okvira ove knjige.

#### 11.9.4 Klasifikacija teksta algoritmom k najbližih suseda

Mnoge metode mašinskog učenja, pa i algoritam  $k$  najbližih suseda, formulisane su tako da se jednostavno primenjuju na numeričke podatke, ali teško na tekst. Rekurentne neuronske mreže prirodno rade sa tekstualnim podacima, ali nose mnoštvo tehničkih izazova poput hardverske zahtevnosti i trikova koji olakšavaju trening i, pored toga, traže velike količine podataka. Otud ne moraju u svim primenama biti prvi izbor. Kako bi se mogli primeniti stariji i jednostavniji algoritmi, potrebni su načini da se tekstualni podaci predstave u numeričkom obliku. U obradi tekstova, proteinskih sekvenci i sličnih podataka, često se za predstavljanje podataka u numeričkom obliku koriste  $n$ -gramske profili.

Ako je data niska  $S = s_1 s_2 \dots s_N$  nad azbukom  $\Sigma$ , gde je  $N$  pozitivan ceo broj,  $n$ -gram niske  $S$ , za  $n \geq N$ , je bilo koja podniska susednih simbola dužine  $n$ . Na primer, za nisku **sad\_ili\_nikad**, 1-grami su: **s**, **a**, **d**, **\_**, **i**, **l**, **i**, **\_**, **n**, **i**, **k**, **a**, **d**. 2-grami su: **sa**, **ad**, **d\_**, **\_i**, **il**, **li**, **i\_**, **\_n**, **ni**, **ik**, **ka**, **ad**. 3-grami su: **sad**, **ad\_**, **d\_i**, **\_il**, **ili**, **li\_**, **i\_n**, **\_ni**, **nik**, **ika**, **kad**, itd.

$N$ -gramske profile niske je lista uređenih parova (*n-gram, frekvencija*), gde je frekvencija izračunata u odnosu na sve  $n$ -grame niske.

Osnovne prednosti korišćenja  $n$ -grama su robustnost (na primer, nisu mnogo osetljivi na greške u kucanju ili na pojavljivanje reči u različitim gramatičkim oblicima), nezavisnost od domena koji se analizira, efikasnost (dovoljan je jedan prolaz kroz tekst) i jednostavnost. Mana je eksponencijalna zavisnost broja mogućih  $n$ -grama u odnosu na dužinu  $n$ -grama.

$N$ -gramske profile uspešno se koriste u različitim primenama koje uključuju prepoznavanje autorstva tekstova, prepoznavanje jezika kojim je tekst pisan, prepoznavanje govora i određene probleme iz oblasti bioinformaticke.  $n$ -gramske profili često se koriste u sadejstvu sa metodom  $k$  najbližih suseda.

Razmotrićemo na konkretnom primeru klasifikaciju tekstova na osnovu jezika. Srpski i engleski jezik biće predstavljeni po jednim kraćim tekstom označenim sa S1 i E1. Pošto se izračunaju frekvencije  $n$ -grama za ta dva teksta, njihovi  $n$ -gramske profili čine trening skup. Test skup će biti dobijen na osnovu četiri kratka teksta od kojih su dva na srpskom označena sa S2 i S3, a dva na engleskom jeziku označena sa E2 i E3. Klasifikacija će biti izvršena pomoću algoritma jednog najbližeg suseda. U tekstovima na srpskom jeziku nisu korišćena srpska slova kako bi se izbegla laka identifikacija na osnovu pisma.

**S1:** U prethodnom delu prikazani su teorijski okviri i algoritmi pomocu kojih je moguce sprovoditi logicko zaključivanje. Iako zaključci moraju nuzno slediti iz zadatih prepostavki, proces njihovog dokazivanja nije pravolinijski vec uključuje odredene odluke o pravcu u kome ce se postupak sprovoditi. Drugim recima, uocljivo je traganje za dokazom nekog tvrdenja. Primera radi, u primeni procedure DPLL moguce je uociti i korake zaključivanja i korake pretrage. Kada se uoci jedinicna klauza u nekoj formuli, njeno zadovoljenje je nuzno i predstavlja korak zaključivanja. S druge strane kada je nemoguce direktno zaključivanje, potrebno je prepostaviti vrednost iskazne promenljive. U daljem toku dokazivanja, ta akcija ce se pokazati kao opravdana ili neopravdana. U slučaju da se pokaze kao neopravdana, preduzima se alternativna akcija. Znaci, situacija u kojoj nije moguce izvrsiti direktno zaključivanje zahteva primenu pretrage.

Manje apstraktan primer je upravljanje robotskom rukom. Prepostavimo da robotska ruka ima nekoliko mehanickih zglobova cije se kretanje kontrolise električnim impulsima. Pritom, neki zglobovi omogućavaju rotacije samo oko jedne ose (kao ljudski lakat ili zglobovi na prstima), a drugi rotaciju oko veceg broja osa (kao ljudsko rame ili zglobovi u korenu prstiju). Pokret hvatanja case ovakvom robotskom rukom je netrivijalan zadatak, ali se može razbiti na sekvencu atomicnih koraka — pokreta pojedinacnih zglobova oko razlicitih osa za odreden ugao. Mozemo zamisliti da se ovi koraci izvrsavaju strogo jedan po jedan u kom bi slučaju kretanje ruke bilo znacajno razlicito od ljudskog i sporo, ali bi problem bio laksi jer ne bi bila potrebna sinhronizacija razlicitih zglobova i svaki bi se pojedinačno dovodio u željeni položaj. Druga mogućnost je da se kretanja zglobova izvode simultano, kao kod čoveka, pri tom povećavajući broj mogućih kombinacija u svakom trenutku.

Dati primeri motivisu razmisljjanje o pretrazi kao o nalazenu niza akcija kojima se ostvaruje cilj kada to ne moze biti ostvareno pojedinačnim akcijama. Iako u opstem slučaju ovakva definicija ne mora delovati adekvatno, u kontekstu vestacke inteligencije u kome obично prepostavljamo postojanje nekog entiteta koji deluje preduzimanjem nekih akcija (agenta), ona je prirodna.

**E1:** There are two paths to achieving an AGI, says Peter Voss, a software developer and founder of the firm Adaptive A.I. Inc. One way, he says, is to continue developing narrow AI, and the systems will become generally competent. It will become obvious how to do that. When that will happen or how it will come about, whether through simbots or some DARPA challenge or something, I dont know. It would be a combination of those kinds of things. The other approach is to specifically engineer a system that can learn and think. Thats the approach that [my firm] is taking. Absolutely I think thats possible, and I think its closer than most people think five to 10 years, tops. The two approaches outlined by Vosseither tinkering with mundane programs to make them more capable and effective or designing a single comprehensive AGI system speak to the long-standing philosophical feud that lies at the heart of AI research: the war between the neats and the scruffies. J. Storrs Hall, author of Beyond AI: Creating the Conscience of the Machine (Prometheus Books, 2007), reduces this dichotomy to a scientific approach vs. an engineering mind-set. The neats are after a single, elegant solution to the answer of human intelligence, Hall says. Theyre trying to explain the human mind by turning it into a math problem. The scruffies just want to build something, write narrow AI codes, make little machines, little advancements, use whatever is available, and hammer away until something happens. The neat approach descends from computer science in its purest form, particularly the war game studies of Von Neumann and his colleagues in the 1930s and 1940s. The 1997 defeat of world chess champion Garry Kasparov by IBMs Deep Blue computer is considered by many the seminal neat success. Up until that moment, the mainstream scientific community generally accepted the premise that AIs could be written to perform specific tasks reasonably well, but largely resisted the notion of superhuman computing ability. Deep Blue proved that an AI entity could outperform a human at a supposedly human task, perceiving a chess board (Deep Blue could see 200 million board positions per second) and plotting a strategy (74 moves ahead as opposed to 10, the human record).

**S2:** Precizni postupci za resavanje matematičkih problema postojali su u vreme starogrčkih matematičara (npr. Euklidov algoritam za određivanje najvećeg zajednickog delioca dva broja), a i pre toga. Ipak, sve do početka dvadesetog veka nije se uvidala potreba za preciznim definisanjem pojma algoritma. Tada je, u jeku reforme i novog utemeljivanja matematike, postavljeno pitanje da li postoji algoritam kojim se (pojednostavljeni rečeno) mogu dokazati sve matematičke teoreme. Da bi se ovaj problem uopste razmatrao, bilo je neophodno najpre definisati (matematički precizno) sta je to precizan postupak, odnosno sta je to algoritam.

**S3:** Dositej Obradović (svetovno ime Dimitrije) (Cakovo, 1744 — Beograd, 1811) je bio srpski prosvetitelj i reformator revolucionarnog perioda nacionalnog budjenja i preporoda. Rodjen je u rumunskom delu Banata tadasne Austrije. Skolovao se za kaludjera, ali je napustio taj poziv i krenuo na putovanja po celoj Evropi, gde je primio ideje evropskog prosvetiteljstva i racionalizma. Ponesen takvim idejama radio je na prosvecivanju svog naroda, prevodio je razna dela među kojima su najpoznatije Ezopove basne, a potom je i sam pisao dela, prvenstveno programskog tipa, među kojima je najpoznatije „Zivot i prikljucenja“. Dositej je bio prvi popecitelj (ministar) prosvete u Sovjetu i tvorac svecane pesme „Vostani Serbie“. Njegovi ostaci pocivaju u Beogradu, na ulazu u Sabornu crkvu.

**E2:** The planet Mars, I scarcely need remind the reader, revolves about the sun at a mean distance of 140,000,000 miles, and the light and heat it receives from the sun is barely half of that received by this world. It must be, if the nebular hypothesis has any truth, older than our world; and long before this earth ceased to be molten, life upon its surface must have begun its course. The fact that it is scarcely one seventh of the volume of the earth must have accelerated its cooling to the temperature at which life could begin. It has air and water and all that is necessary for the support of animated existence.

**E3:** Principia Mathematica, the landmark work in formal logic written by Alfred North Whitehead and Bertrand Russell, was first published in three volumes in 1910, 1912 and 1913. Written as a defense of logicism (the view that mathematics is in some significant sense reducible to logic) the book was instrumental in developing and popularizing modern mathematical logic. It also served as a major impetus for research in the foundations of mathematics throughout the twentieth century. Along with the Organon written by Aristotle and the Grundgesetze der Arithmetik written by Gottlob Frege, it remains one of the most influential books on logic ever written.

Koriste se  $n$ -grami dužine 3, tj. koristi se vrednost  $n = 3$ . Iz tekstova S1 i E1 izdvojeno je po 10 najfrekventnijih  $n$ -grama. Ovi  $n$ -gramske profili činiće svojstva instanci. Svojstva trening i test skupa dati su u tabeli 11.6.

$n$ -gram	Trening skup		Test skup			
	S1	E1	S2	S3	E2	E3
JE_	0.0129	0	0.0131	0.0201	0	0
_PR	0.0125	0.0023	0.0098	0.0148	0	0
ANJ	0.0076	0	0.0082	0.0027	0	0
_KO	0.0076	0	0.0016	0.0027	0	0
JA_	0.0076	0	0.0033	0.0040	0	0
_JE	0.0067	0	0.0082	0.0121	0	0
_PO	0.0067	0.0009	0.0147	0.0080	0	0.0016
_SE	0.0062	0.0018	0.0049	0.0027	0.0016	0.0032
NJE	0.0058	0	0.0065	0.0027	0	0
_U_	0.0058	0	0.0033	0.0067	0	0
_TH	0	0.0212	0	0	0.0270	0.0175
THE	0	0.0148	0	0	0.0202	0.0191
HE_	0	0.0120	0	0	0.0185	0.0127
ING	0	0.0088	0	0	0.0017	0.0032
NG_	0	0.0078	0	0	0.0034	0.0048
_CO	0.0004	0.0074	0	0	0.0051	0
ER_	0.0009	0.0069	0	0	0.0051	0.0032
ND_	0	0.0065	0	0	0.0101	0.0079
_TO	0.0018	0.0065	0.0049	0	0.0034	0.0016
TO_	0.0009	0.0065	0.0033	0	0.0034	0.0016

Tabela 11.6: Trening i test skup za klasifikaciju tekstova prema jeziku. Za svaki 3-gram prikazana je njegova frekvencija u tekstu S1 i E1.

Prilikom klasifikacije biće korišćeno Euklidsko rastojanje

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^m (x_i - y_i)^2}$$

Potrebno je ispitati rastojanja od instanci test skupa do instanci trening skupa.

$$\begin{aligned} d(S2, S1) &= 0.0124 \\ d(S2, E1) &= 0.0417 \end{aligned}$$

$$\begin{aligned} d(S3, S1) &= 0.0133 \\ d(S3, E1) &= 0.0450 \end{aligned}$$

$$\begin{aligned} d(E2, S1) &= 0.0482 \\ d(E2, E1) &= 0.0149 \end{aligned}$$

$$\begin{aligned} d(E3, S1) &= 0.0397 \\ d(E3, E1) &= 0.0141 \end{aligned}$$

Pošto je rastojanje od instance S2 do S1 manje nego od S2 do E1, zaključuje se da je S1 najbliži sused instance S2. Zbog toga se instanca S2 prepoznaje kao tekst na sprskom jeziku. Slično se ispravno zaključuje i da je S3 tekst na srpskom, E2 tekst na engleskom i E3, takođe, tekst na engleskom jeziku. Posebno je zanimljivo da tekstovi S3 i E2 po svom sadržaju nemaju dodira sa instanicama za trening S1 i E1, a to ne ometa uspešnu klasifikaciju.

### 11.9.5 Prepoznavanje poze i delova tela iz dubinskih slika stablima odlučivanja

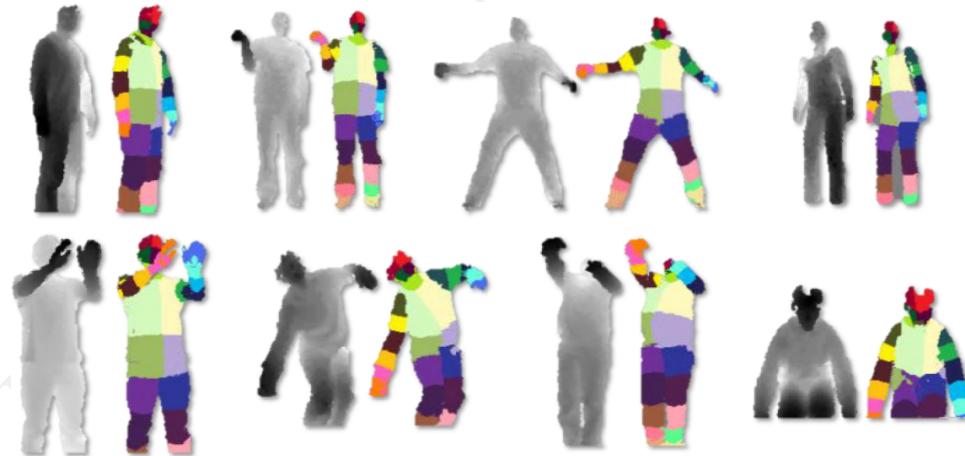
Majkrosoftov sistem Kinekt (eng. *Microsoft Kinect*) sa svojom mogućnošću prepoznavanja poze tela igrača predstavlja jedan od zanimljivijih primera primene mašinskog učenja koja je doprla do velikog broja ljudi.

Sistem prepoznaće poze i pokrete igrača i preduzima akcije u igri u skladu sa njima. Kinekt je prvi sistem koji je omogućio prepoznavanje punog spektra pokreta i veliku robusnost u odnosu na oblik tela, odeću i slično, a pri brzini od 200 slika u sekundi. Ovakva brzina posledica je dve odluke u dizajnu sistema. Prvo, svojstva na osnovu kojih se uči definisana su tako da se izrazito brzo izračunavaju. Drugo, koriste se stabla odlučivanja, a kod njih je postupak predviđanja vrlo brz, značajno brži nego kod modernih neuronskih mreža: u slučaju izbalansiranih stabala, vreme je logaritamsko u odnosu na veličinu stabla. Sistem se u potpunosti oslanja na slike dubinske kamere koje kao vrednost svakog piksela daju *dubinu* — rastojanje od kamere do lokacije na objektu na kojoj se taj piksel nalazi.

Sistem je treniran na osnovu više stotina hiljada dubinskih slika (pravih i simuliranih), pri čemu je za svaku takvu sliku dostupna i druga, njoj odgovaraajuća slika sa datim informacijama o delovima tela kojima pikseli pripadaju (ilustracija na slici 11.35). Prepoznavanje dela tela vrši se za svaki piksel pomoću skupa stabala odlučivanja, takozvane *slučajne šume*. Kao i svaki model, i ovaj zahteva svojstva na osnovu kojih će se davati predviđanja. Svaki piksel opisan je skupom svojstava uniformne strukture. Za dati piksel, svako svojstvo predstavlja razliku između dubina neka dva, pogodno odabrana, piksela bliska polaznom. Neka je  $d_I(\mathbf{x})$  dubina piksela na poziciji  $\mathbf{x}$  na slici  $\mathbf{I}$  (u slučaju pozicija koje su van slike ili na pozadini, za dubinu se uzima velika pozitivna vrednost). Onda, za par vektora  $\theta = (\mathbf{u}, \mathbf{v})$  u prostoru koordinata slike, odgovarajuće svojstvo piksela  $\mathbf{x}$  definisano je na sledeći način:

$$f_\theta(\mathbf{I}, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

Razmotrimo smisao ovog svojstva. Za dati piksel  $\mathbf{x}$ ,  $\mathbf{x} + \mathbf{u}/d_I(\mathbf{x})$  predstavlja piksel u relativnom susedstvu u smeru vektora  $\mathbf{u}$ . Deljenje dubinom piksela  $\mathbf{x}$  služi da umanji pomeraj na slici u slučaju daljih piksela kako bi relativni odnos pomeraja za piksele na različitim dubinama ostao isiti. Onda svojstvo definisano za dati par  $\theta$  predstavlja razliku dubina odgovarajućih suseda datog piksela  $\mathbf{x}$ . Mnoštvo ovakvih svojstava za različite parove  $\theta$  opisuje oblik okoline piksela  $\mathbf{x}$ , što može dati dovoljno informacija za prepoznavanje dela tela. Jedan od ključnih motiva za izbor ovakvih svojstava je njihovo izrazito brzo izračunavanje. Konkretno, za izračunavanje jednog svojstva potrebno je izvršiti samo tri čitanja iz memorije kako bi se odredile dubine i pet aritmetičkih operacija kako bi se izračunala njegova vrednost.



Slika 11.35: Dubinske slike i odgovarajuće slike sa obeleženim delovima tela.

Algoritam učenja je po duhu sličan algoritmu ID3 koji je predstavljen u poglavljiju 11.7, ali kako su vrednosti svojstava neprekidne, ID3 se ne može direktno primeniti. Dodatno, jedno stablo manje je moćno od većeg broja relativno nezavisnih stabala, pa se umesto jednog koristi šuma stabala. Najpre ćemo opisati kako se trenira jedno stablo, a potom kako se vrši zajedničko odlučivanje pomoću većeg broja stabala.

Algoritam učenja primenjuje se nad skupom piksela  $D$  koji je dobijen nasumičnim izborom po 2000 piksela sa svake slike. Algoritam je određen sledećom rekurzivnom procedurom:

1. Nasumice generisati testove kao parove  $\phi = (\theta, \tau)$ , gde je  $\tau$  skalar.
2. Za svaki test  $\phi$ , izračunati dobitak informacije kada se podaci podele na skup  $D_l(\phi)$  u kojem su pikseli za koje je vrednost svojstva  $f_\theta$  manja od praga  $\tau$  i na skup  $D_r(\phi)$  koji sadrži ostale piksele.
3. Izabratи za koren stabla test  $\phi^*$  za koji je dobitak informacije maksimalan.

4. Rekurzivno konstruisati levo podstablo nad skupom piksela  $D_l(\phi^*)$  i desno nad skupom  $D_r(\phi^*)$ .

Svako stablo  $t$  u svakom listu definiše raspodelu po klasama  $P_t(c|\mathbf{I}, \mathbf{x})$  gde je verovatnoća svake klase  $c$  proporcionalna broju piksela iz trening skupa koji su pridruženi tom listu, a pripadaju klasi  $c$ . Naravno, nije za svaki list potrebno čuvati sve instance koje su mu u treningu pridružene, već samo (diskretnu) raspodelu verovatnoće po klasama. Kada je na ovaj način generisano  $T$  stabala, raspodela verovatnoće po klasama se definiše kao

$$P(c|\mathbf{I}, \mathbf{x}) = \frac{1}{T} \sum_{i=1}^T P_t(c|\mathbf{I}, \mathbf{x})$$

a za datu sliku  $I$  i piksel  $\mathbf{x}$ , klasa se prirodno bira kao najverovatnija klasa u odnosu na ovu raspodelu.

Primetimo da pomenuta relativna nezavisnost stabala dolazi iz nasumičnog izbora podataka i testova pri njihovoj izgradnji. Osnovna ideja algoritma je da će ovako generisana stabla grešiti nezavisno jedna od drugih, pa ako je svako od njih bolje od slučajnog pogađanja, većina stabala će biti u pravu i moći će da nadglaša manjinu koja greši na dатој instanci.

Na ovaj način, moguće je označiti sve piksele najverovatnijim klasama koje se odnose na delove tela. Ipak, za potrebe praćenja pokreta, to je previše sirova informacija. Umesto da se koriste pozicije svih piksela koji čine neki deo tela, poželjnije je koristiti svega nekoliko reprezentativnih tačaka. Ovo se postiže klasterovanjem odnosno grupisanjem svih tačaka koje čine neki deo tela i uzimanjem centralnih tačaka tih klastera koje nadalje predstavljaju taj deo tela. Na ovaj način je moguće oceniti pozicije različitih delova tela, a u procesu klasterovanja se eliminiše i šum koji nastaje usled pogrešne klasifikacije malog broja piksela.



## Glava 12

# Nenadgledano mašinsko učenje

Nenadgledano učenje je vid mašinskog učenja kod kojeg nisu date vrednosti ciljne promenljive. Naravno, mašinsko učenje ne može dati korisnu informaciju ni iz čega. Dok su kod nadgledanog učenja algoritmi takvi da često mogu učiti bilo kakve zakonitosti u datim podacima, a da se to što se uči definiše vrednostima ciljne promenljive, kako tih, ciljnih vrednosti nema u slučaju nenadgledanog učenja, ono što se uči mora biti definisano samim algoritmom. Dakle, algoritmi nenadgledanog učenja su algoritmi specifične namene.

Većina problema koji odgovaraju ovoj grupi potпадa pod probleme klasterovanja, učenja reprezentacije i detekcije anomalija:

**Klasterovanje** predstavlja uočavanje grupa u podacima, što može da nešto govori o strukturi podataka i može biti korisno u različite svrhe, o kojima će biti reči u nastavku.

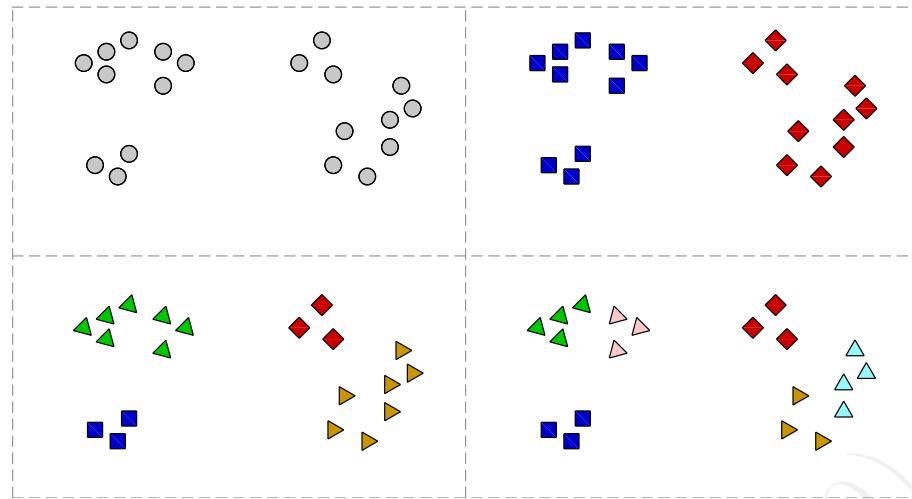
**Učenje reprezentacije** predstavlja sve značajniju vrstu zadataka u mašinskom učenju. Neretko podaci nisu u obliku u kojem ih algoritam učenja ili čovek mogu lako iskoristiti. Na primer, algoritmi mašinskog učenja obično zahtevaju više parametara ukoliko podaci imaju više dimenzija i zahtevaju više računskih operacija, a podaci su često visokodimenzionalni – recimo,  $1.000.000$  dimenzija u slučaju slika kod kojih svaki piksel predstavlja jednu dimenziju ili  $100.000$  dimenzija u slučaju obrade teksta gde se za svojstva koriste frekvencije pojedinih reči. Međutim, ovako visoka dimenzionalnost najčešće je posledica izabrane reprezentacije podataka. Na primer, slike lica, čak i u rezoluciji  $1.000 \times 1.000$ , ne popunjavaju ravnomerno  $1.000.000$ -dimenzionalni prostor, već samo njegov delić. Ostatak odgovara drugim slikama, od kojih mnoge ne predstavljaju ništa prepoznatljivo čoveku. To sugerira da postoji reprezentacija manje dimenzionalnosti koja takođe opisuje sva lica. Nalaženje takvih reprezentacija i učenje nad njima značajno povećava uspešnost algoritama učenja. Nekada ovakvi algoritmi služe i za smanjenje dimenzionalnosti podataka na dve ili tri dimenzije koje najbolje oslikavaju njihovu varijabilnost, što omogućava čoveku da u nekim slučajevima posmatranjem uoči neke važne aspekte podataka.

**Detekcija anomalija** se tiče uočavanja podataka koji štete u odnosu na ostale, bilo kako bi se izbacili iz skupa podataka, čime se često olakšava njihovo modelovanje i analiza, bilo kako bi se dalje analizirali. Primera radi, transakcije kreditnim karticama koje predstavljaju prevaru, lako mogu odudarati od uobičajenog načina na koji korisnik koristi karticu, bilo po vremenu upotrebe, bilo po vrsti usluge ili proizvoda koji se kupuje, bilo po iznosu transakcije. Slično važi i za upade u računarske sisteme – ta vrsta ponašanja često odudara od uobičajenog ponašanja korisnika.

U nastavku ćemo se fokusirati na problem klasterovanja.

### 12.1 Klasterovanje

Klasterovanje predstavlja identifikaciju grupa u datim podacima. Potreba za rešavanjem ovakvog problema može se javiti u različitim praktičnim problemima, poput identifikacije zajednica u društvenim mrežama (na primer, za potrebe oglašavanja), detekcije raznorodnih tkiva na medicinskim snimcima, ustanavljanja zajedničkog porekla jezika, identifikacije grupa živilih bića i, specifično, ljudskih zajednica i slično. Pored primena koje u sebi direktno kriju problem klasterovanja, ova tehnike korisne su i u pretprocesiranju podataka na koje kasnije treba da budu primenjene metode nadgledanog učenja. Na primer, u slučaju obrade ogromnog broja podataka, cele grupe podataka mogu biti zamjenjene svojim reprezentativnim predstavnicima. Ovo nije uvek idealno sa tačke gledišta kvaliteta dobijenog prediktivnog modela, ali sa tačke gledišta računske i memorijске efikasnosti može biti isplativo. Takođe, kako bi se obezbedilo da prilikom unakrsne validacije različiti podskupovi



Slika 12.1: Različita klasterovanja nad istim podacima: jedan klaster (gore levo), dva klastera (gore desno), četiri klastera (dole levo), šest klastera (dole desno).

imaju slično raspodeljene podatke, podaci se mogu prvo klasterovati, a potom se  $n$  podskupova može formirati tako što se svaki od klastera podeli na  $n$  jedakih delova koji se razvrstavaju u različite podskupove. Dakle, klasterovanje može biti korisno i kao tehnika rešavanja problema i kao tehnika preprocesiranja.

Pojam klasterovanja nije jednoznačno definisan. Kao što primer prikazan na slici 12.1 ukazuje, u jednom skupu može se identifikovati više različitih grupisanja, često različite granularnosti. Pritom, takvi slučajevi nisu posledica nedovoljnog promišljanja definicije klasterovanja, već raznovrsnosti konteksta u kojima se grupisanje može vršiti i ciljeva koji se pomoću klasterovanja žele postići. Za očekivati je da je nekada potrebno izvršiti grublje klasterovanje – u manji broj klastera, a nekada finije – u veći broj klastera. Algoritmi klasterovanja obično omogućavaju podešavanje nivoa granularnosti tj. broja klastera koji se u podacima pronalazi.

Pojam klasterovanja nije jednoznačno definisan ne samo u odnosu na broj klastera koji se u podacima mogu naći, već i u odnosu na ideju šta jednu grupu instanci čini klasterom. U odnosu na to, postoji više neformalnih definicija klasterovanja. *Globularni* ili *centrični* klasteri su grupe tačaka koje popunjavaju unutrašnjost lopte ili, opštije, elipsoida. *Dobro razdvojeni* klasteri su grupe tačaka koje su bliže drugim tačkama svoje grupe nego bilo kojoj tački iz neke druge grupe. *Gustinski* klasteri su klasteri čije su tačke razdvojene od tačaka drugih klastera regionima manje gustine. *Hijerarhijski* klasteri su ili pojedinačne tačke ili klasteri čije su tačke takođe organizovane u strukturu hijerarhijskih klastera.

U nastavku je prikazano nekoliko jednostavnih, a često korišćenih algoritama klasterovanja.

### 12.1.1 Algoritam $k$ sredina

Algoritam  $k$  sredina pronalazi  $k$  klastera koje predstavlja pomoću  $k$  težišta tih klastera, od kojih se svako dobija uprosečavanjem elemenata datog klastera. Taj korak čini algoritam primenljivim samo na podatke koji se mogu uprosečavati, poput vektora. Pod određenim uslovima, postoje uopštenja algoritma i na drugačije vrste podataka, ali o njima neće biti reči. Polaznih  $k$  težišta bira se nasumično (mada, ako korisnik zna nešto o strukturi svojih podataka, mogu biti i unapred data), a potom se ponavljaju koraci pregrupisavanja tačaka u nove klastere prema bliskosti sa postojećim težištimi i preračunavanja novih težišta sve dok se ona menjaju. Algoritam je preciznije formulisan na slici 12.2.

Primer klastera koje pronalazi ovaj algoritam, dat je na slici 12.3. Može se pokazati da ovaj algoritam minimizuje veličinu

$$\sum_{i=1}^k \sum_{\mathbf{x} \in C_i} d(\mathbf{x}, \mathbf{c}_i)^2$$

gde je  $d$  euklidsko rastojanje. Na osnovu toga može se nešto zaključiti i o njegovom ponašanju. Zahvaljujući tome što je zasnovan na minimizaciji euklidskog rastojanja, algoritam teži pronalaženju klastera u obliku lopte. Kako je rastojanje kvadrirano, algoritam je osetljiv na podatke koji značajno odudaraju od ostalih. Naime, u slučaju podatka koji značajno odudara od ostalih, veće rastojanje će uticati na ukupnu grešku neproporcionalno u odnosu na ostala rastojanja i takav podatak će neprpororacionalno uticati na lokaciju težišta. Takođe, ako gustina podataka, tj. tačaka prostora ne varira drastično i rastojanja među klasterima nisu velika, algoritam

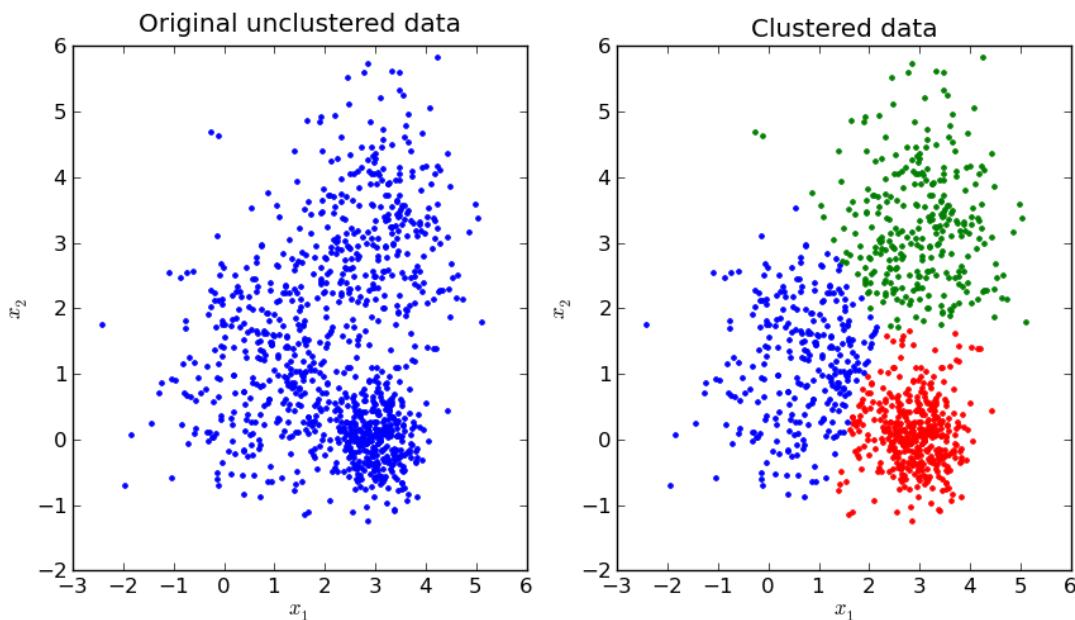
**Algoritam:** Algoritam  $k$  sredina

**Ulaz:** Trening skup  $T \subset \mathbb{R}^n$ , broj klastera  $k$

**Izlaz:** Particionisanje skupa  $T$  na disjunktne neprazne podskupove  $C_1, \dots, C_k$

- 1: Nasumice izaber i težišta  $c_1, \dots, c_k$  iz skupa  $T$
- 2: **ponavljam**
- 3: Postavi skupove  $C_1, \dots, C_k$  na prazne skupove;
- 4: **za svaku** instancu  $x \in T$  **radi**
- 5:     pronađi težište  $c_i$  koje je najbliže instanci  $x$ ;
- 6:     dodaj  $x$  u skup  $C_i$ ;
- 7: Izračunaj proseke  $c_1, \dots, c_k$  instanci iz skupova  $C_1, \dots, C_k$ ;
- 8: **dok nije ispunjen** težišta su ista kao u prethodnoj iteraciji;
- 9: vrati  $C_1, \dots, C_k$  kao rešenje.

Slika 12.2: Algoritam klasterovanja  $k$  sredina.



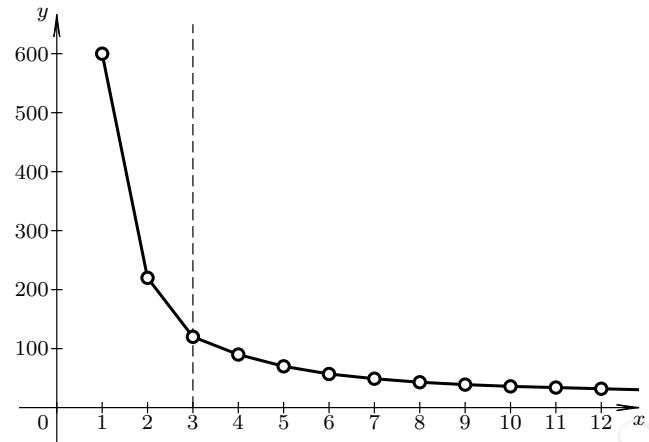
Slika 12.3: Klasteri pronađeni algoritmom  $k$  sredina.

preferira klastere sa sličnim brojem tačaka u njima, pošto bi u suprotnom brojan klaster sadržao i tačke dalje od težišta koje bi značajno povećavale sumu kvadrata rastojanja.

Činjenica da algoritam  $k$  sredina minimizuje navedenu sumu navodi na njenu dalju analizu. Bitno je pitanje da li ona ima jedan globalni minimum, odnosno da li je najbolje klasterovanje u odnosu na datu sumu kvadrata rastojanja jedinstveno. Odgovor na ovo pitanje je negativan. Moguće je da postoji veći broj klasterovanja jednakog kvaliteta. Jedan primer u kojem bi to bilo i intuitivno je kada su tačke uniformno raspoređene unutar kruga i potrebno ih je podeliti na dva klastera. Rotiranje dobijenih težišta u odnosu na centar kruga daje podjednako dobro klasterovanje. Drugim rečima, u slučaju takvog skupa podataka, postoji puno globalnih, i samim tim podjednako dobrih, minimuma. Takva situacija nije zabrinjavajuća. Ipak, ispostavlja se da mogu postojati i lokalni minimumi slabijeg kvaliteta od globalnog i da algoritam može naći takav minimum, što nije dobro. Ovaj problem ublažava se tako što se klasterovanje pokreće veći broj puta sa različitim inicijalnim tačkama i za rezultat se uzima klasterovanje sa najmanjom vrednošću sume kvadrata rastojanja.

Algoritam  $k$  sredina omogućava fleksibilnost pri pronalaženju klastera kroz mogućnost podešavanja broja  $k$ . Ipak, ta fleksibilnost često vodi do nedoumica jer u praksi često nije jasno kako izabrati broj  $k$ . Jedno pravilo heurističko je „pravilo lakta“ koje sugerise da se za različite vrednosti broja  $k$  izvrši klasterovanje, da se nacrti grafik zavisnosti sume kvadrata rastojanja u zavisnosti od  $k$  i da se za izabere klasterovanje koje odgovara broju

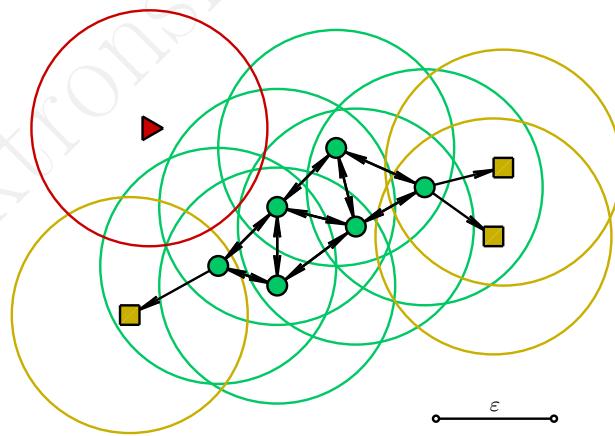
$k$  koji leži na tački nagle promene brzine opadanja grafika ili na njegovom „laktu“. Ovakva situacija prikazana je na grafiku 12.4. Intuitivno obrazloženje je da su nakon „lakta“ klasteri već homogeni i dodavanje novih težišta ne doprinosi značajno smanjenju sume kvadrata rastojanja.



Slika 12.4: Pravilo lakta sugerije da za broj  $k$  treba uzeti vrednost koja odgovara uspravnoj isprekidanoj liniji; na  $x$  osi je broj klastera, a na  $y$  osi suma kvadrata rastojanja.

### 12.1.2 DBSCAN

Algoritam DBSCAN (eng. *density-based spatial clustering of applications with noise*) služi za detekciju gustinskih klastera i nije ograničen njihovim oblikom. Algoritam zahteva dva parametra – rastojanje  $\varepsilon$  i minimalan broj tačaka  $\mu$ . Algoritam razvrstava tačke na *tačke koje čine jezgro* – koje u svojoj  $\varepsilon$  okolini imaju bar  $\mu$  tačaka, *granične tačke* – koje u svojoj okolini imaju neku tačku koja čini jezgro i *tačke koje čine šum* – koje nisu granične niti čine jezgro. Tačke koje čine šum se zanemaruju, dok se ostale grupišu u različite klasterne na osnovu bliskosti. Ilustracija vrsta tačaka data je na slici 12.5. Algoritam je preciznije opisan na slici 12.6.



Slika 12.5: Tri vrste tačaka kojima operiše algoritam DBSCAN (za vrednost  $\mu = 4$ ): (zelenim) kružićima označene su tačke jezgra, (žutim) kvadratićima granične tačke, a (crvenim) trouglicima tačke koje čine šum.

DBSCAN očigledno ne prepostavlja oblik klastera, tako da se može koristiti za detekciju klastera najrazličitijih oblika ako ih je moguće razdvojiti regionima niže gustine. Još jedna dobra strana ovog algoritma je to što može odstraniti odudarajuće podatke kao što je šum (dok je algoritam  $k$  najbližih suseda vrlo osjetljiv na njihovo prisustvo). Potencijalni problem nastaje kada su klasteri, iako razdvojeni, sami vrlo različite gustine. Naime, u okolini vrlo gustog klastera, šum može imati veću gustinu nego ceo drugi klaster (koji ipak može biti prepoznatljiv po tome što će značajno odudarati po gusitni od svoje okoline). Kao i u slučaju algoritma  $k$  sredina, nema jasnih opštih pravila za izbor vrednosti  $\varepsilon$  i  $\mu$ .

**Algoritam:** Algoritam DBSCAN**Ulaz:** Trening skup  $T$ , rastojanje  $\varepsilon$  i broj  $\mu$ **Izlaz:** Particionisanje skupa  $T$  na disjunktne neprazne podskupove  $C_1, \dots, C_k$ 

- 1: Formiraj skup  $\mathcal{C}$  svih tačaka koje u svojoj  $\varepsilon$  okolini imaju bar  $\mu$  tačaka;
- 2: Formiraj skup  $\mathcal{B}$  svih tačaka iz  $T \setminus \mathcal{C}$  koje u svojoj  $\varepsilon$  okolini imaju bar jednu tačku iz  $\mathcal{C}$ ;
- 3: Formiraj graf  $G$  čiji su čvorovi tačke iz  $\mathcal{C} \cup \mathcal{B}$ , a grana postoji između svake dve tačke koje su na rastojanju najviše  $\varepsilon$ ;
- 4: vrati komponente povezanosti  $C_1, \dots, C_k$  grafa  $G$  kao rešenje.

Slika 12.6: Algoritam klasterovanja DBSCAN.

**12.1.3 Hijerarhijsko klasterovanje**

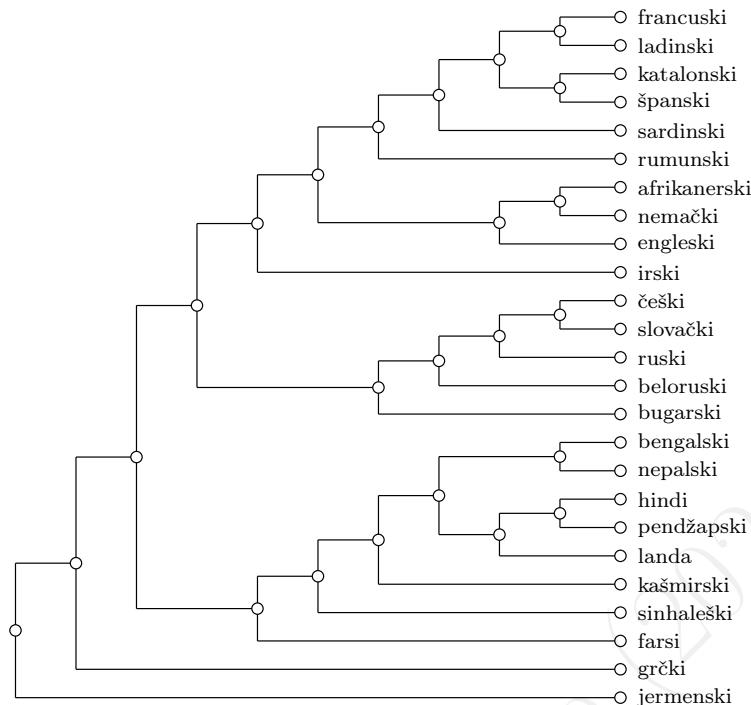
Hijerarhijsko klasterovanje konstruiše stablo u čijim se listovima nalaze instance trening skupa, a unutrašnji čvorovi definišu strukturu klastera. Klaster koji odgovara nekom unutrašnjem čvoru sastoји se iz klastera koji odgovaraju njegovim direktnim potomcima. Problem klasterovanja svodi se na problem konstrukcije ovakvog stabla. Pristupa rešavanju ovog problema ima više. Jedan koji se često razmatra je *hijerarhijsko aglomerativno klasterovanje* pri kojem se skup klastera inicijalizuje pojedinačnim instancama, a potom se u svakom koraku, spajaju dva najsličnija klastera u jedan, čime se konstruiše binarno stablo. Takvo stablo naziva se *dendogram* i ilustrovano je na slici 12.8. Sličnost nad klasterima nije trivijalno definisati i ne postoji jedan izbor. Najčešće se definiše neka mera sličnosti ili rastojanja nad pojedinačniminstancama (poput euklidskog rastojanja), pa se mera sličnosti ili rastojanja klastera definije na osnovu nje. Na primer, rastojanje između dva klastera se može definisati kao minimum, prosek ili maksimum rastojanja njihovih elemenata. Precizniji opis hijerarhijskog aglomerativnog klasterovanja dat je na slici 12.7. U algoritmu se koristi mera rastojanja, a izmene kojima se prelazi na meru sličnosti su trivijalne.

**Algoritam:** Algoritam aglomerativnog hijerarhijskog klasterovanja**Ulaz:** Trening skup  $T = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ , mera rastojanja  $d$ **Izlaz:** Stablo kalasterovanja  $\mathcal{T}$ 

- 1: Inicijalizuj skup  $\mathcal{C}$  na skup listova  $\{\{l_1 = (\mathbf{x}_1, 0)\}, \dots, \{l_N = (\mathbf{x}_N, 0)\}\}$ ;
- 2: **ponavljam**
- 3: Izračunaj rastojanja  $d_{ij} = d(c_i, c_j)$  među svim elementima  $(c_i, d_i), (c_j, d_j) \in \mathcal{C}$ ;
- 4: Pronadi najbliži par elemenata  $(c_i, d_i), (c_j, d_j)$  iz  $\mathcal{C}$ ;
- 5: Konstruiši unutrašnji čvor  $(c_i \cup c_j, d_{ij})$  i dodaj mu čvorove  $(c_i, d_i)$  i  $(c_j, d_j)$  kao direktnе potomke;
- 6:  $\mathcal{C} \leftarrow (\mathcal{C} \setminus \{(c_i, d_i), (c_j, d_j)\}) \cup \{(c_i \cup c_j, d_{ij})\}$ ;
- 7: **dok nije ispunjen** skup  $\mathcal{C}$  je jednočlan
- 8: Vrati stablo čiji je koren jedini čvor u skupu  $\mathcal{C}$ .

Slika 12.7: Algoritam aglomerativnog hijerarhijskog klasterovanja.

Sečenjem stabla na različitim nivoima mogu se dobiti klasterovanja različite granularnosti. Konkretno, za neku vrednost rastojanja  $d$ , na osnovu stabla moguće je identifikovati poslednji skup  $\mathcal{C}$  koji je u toku keriranja stabla nastao spajanjem čvorova čije rastojanje ne prelazi datu vrednost  $d$ . Dobra strana ovakvog klasterovanja je to što pruža više informacija nego algoritmi koji pronalaze samo jednu podelu podataka na klastera. Takođe, različitim izborima mere sličnosti nad klasterima mogu se dobiti različiti klasteri. Loša strana algoritma je to što u toku rada izračunavanje sličnosti svih klastera sa svim klasterima vodi visokoj vremenskoj, a najčešće i prostornoj složenosti zbog čuvanja matrice rastojanja.



Slika 12.8: Primer hijerarhijskog klasterovanja nekih indoevropskih jezika.

## 12.2 Primeri primena klasterovanja

Kao što je već navedeno, klasterovanje se često koristi kao korak preprocesiranja podataka za potrebe nadgledanog učenja. Ipak, zanimljivije je razmotriti primere primene u kojima je rezultat klasterovanja relevantan sam za sebe. U nastavku diskutujemo dve takve primene.

### 12.2.1 Utvrđivanje srodnosti prirodnih jezika

Jedno od ključnih pitanja uporedne lingvistike je ustanovljavanje srodnosti između postojećih prirodnih jezika. Smatra se da se svi jezici mogu hijerarhijski grupisati u skladu sa njihovom srodnosću koja je posledica nastanka više jezika od zajedničkog pretka, takozvanog proto jezika. Ovaj postupak grupisanja može se hijerarhijski ponavljati. Iako ne postoji saglasnost u vezi sa tačnim brojem grupa i njima odgovarajućih proto jezika, obično postoji saglasnost do nekog nivoa grupisanja. Na primer, neke jezičke grupe su grupe indo-evropskih, sino-tibetanskih, turkijskih i uralskih jezika. Neke od podgrupa indo-evropskih jezika su germanska, balto-slovenska, indo-iranska i keltska. Ove grupe se mogu dalje deliti. Ovakvo grupisanje obično se ustanovljava temeljnom višedecenijskom analizom koju sprovode zajednice lingvista koje se bave ovim problemima. Nekada postoje neslaganja u vezi sa pripadnostima nekih jezika određenim podgrupama, posebno u vezi sa starim jezicima na kojima nema mnogo pisanih dokumenata (često su to samo natpisi u kamenu).

Zanimljivo je da se ovakva grupisanja vrše i automatskim metodama – metodama hijerarhijskog klasterovanja koje prirodno formiraju ugnezđene klastere. Naravno, pitanje je kako predstaviti jezike tako da se nad njima može izvršiti klasterovanje i kako meriti sličnost između njih. Jedan način da se ovo uradi sastoji se u uparivanju odgovarajućih reči različitih jezika. Sličnost dva jezika se onda može kvantifikovati zbirom edit rastojanja po svim parovima odgovarajućih reči za ta dva jezika. Koristeći te informacije, moguće je primeniti neku od diskutovanih tehnika hijerarhijskog klasterovanja. Izbor skupa reči kojima će biti predstavljeni jezici može uticati na ishode klasterovanja. Na primer, skorašnja preuzimanja iz stranih jezika verovatno će mnoge jezike učiniti naizgled srodnim sa engleskim jezikom. Otud je u tom izboru potrebno učešće stručnjaka koji bi sugerisali koje grupe reči su bolje za ovakva istraživanja (na primer, reči za male brojeve, floru, faunu, zemljoradnju i slično). Naravno, postupak može biti i složeniji od opisanog ili zasnovan na drugačijoj reprezentaciji. Recimo, jezik bi se mogao predstaviti frekvencijama bigrama izračunatih iz velikog neobeleženog korpusa. Takvi vektori mogu se porebiti euklidskim rastojanjem. Ishod hijerarhijskog klasterovanja nekih predstavnika indoevropskih jezika zasnovanog na bigramskoj reprezentaciji i euklidskom rastojanju dat je na slici 12.8. Jasno se mogu

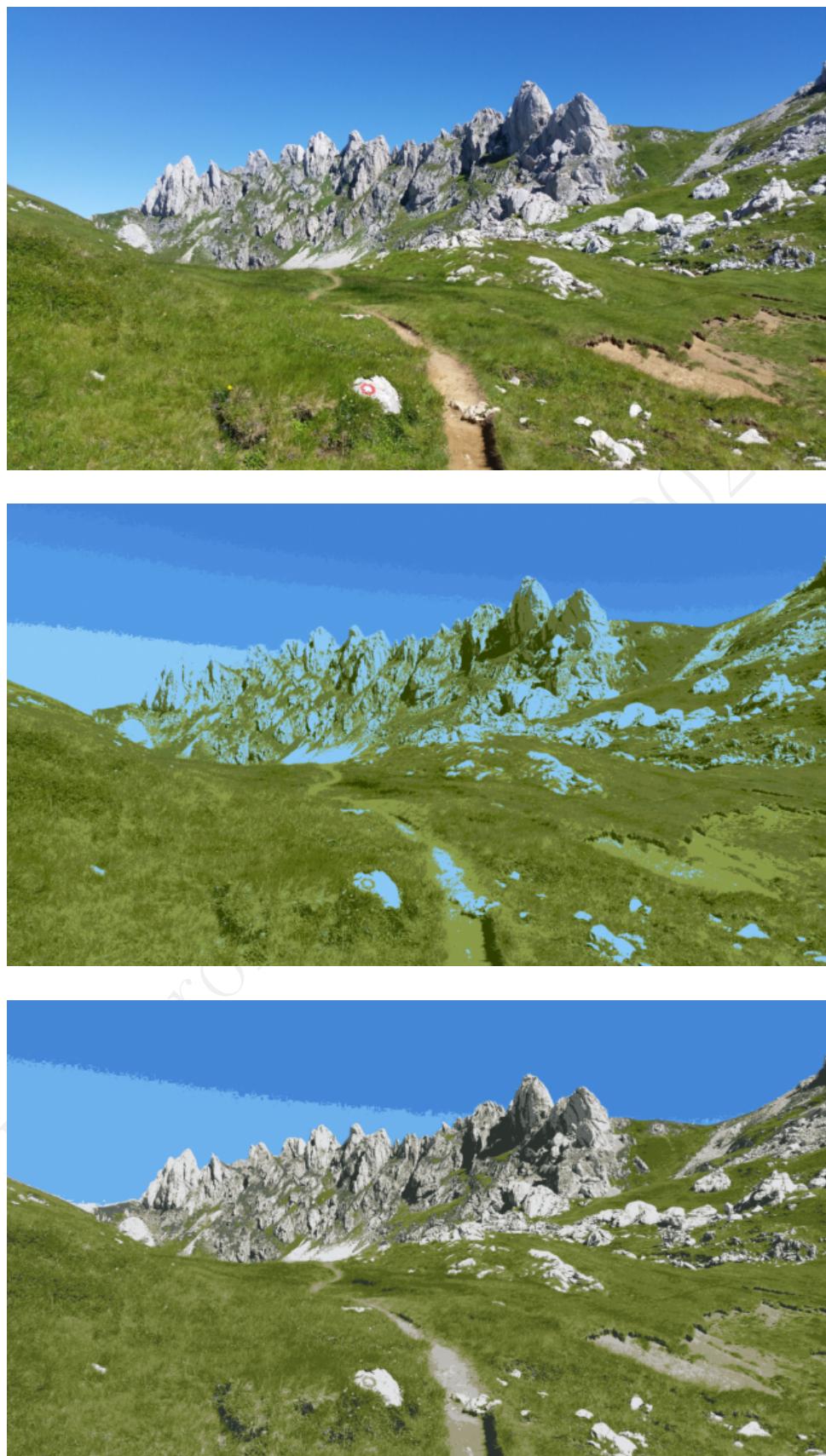
prepoznati karakteristične podgrupe ove familije jezika.

Primetimo da je ovakav postupak mogao biti primenjen (i primenjuje se) i za utvrđivanje srodnosti živih bića, na osnovu njihovog genetskog materijala koji se takođe može predstaviti tekstrom u vidu nizova aminokiselina od kojih svakoj odgovara po jedno slovo.

### 12.2.2 Smanjenje paleta boja slike

Nekada je potrebno smanjiti broj različitih nijansi boja koje su prisutne na slici. Ovo može biti rađeno u svrhe smanjenja zapisa slike ili zbog potreba štampe. Jedan od mogućih pristupa bi bio uočavanje sličnih nijansi i njihova zamena jednom nijansom koja je u nekom smislu najsličnija svima njima. Pretpostavimo da se slika predstavlja pomoću 24-bitnog zapisa u kojem su boje predstavljene pomoću intenziteta crvene, zelene i plave boje (tzv. RGB sistem boja) tako da se intenzitet svake od njih predstavlja pomoću jednog bajta brojevima od 0 do 255. Ovakav zapis dozvoljava preko 16 miliona boja. Za zapis slike rezolucije  $1000 \times 1000$ , potrebno je 3 miliona bajtova. Ukoliko bi se broj različitih boja sveo na 256, bilo bi potrebno, bez ikakve kompresije, milion bajtova za zapis slike u kojem bi svakom pikselu bio pridružen jednobajtni kôd boje i bilo bi potrebno  $256 \times 3$  bajtova za predstavljanje palete – niza od 256 24-bitnih brojeva koji predstavljaju izbor 256 korišćenih nijansi iz polaznog 24-bitnog sistema. Problem se sastoji u identifikaciji izbora podskupa, odnosno palete boja koje će poslužiti za aproksimaciju svih ostalih boja na slici. Pritom, ta paleta može da varira od slike do slike i bira se tako da bude pogodna za konkretnu sliku.

Jedan jednostavan pristup smanjenju palete sastoji se u klasterovanju piksela pomoću algoritma  $k$  sredina u onoliko klastera kolika je željena veličina palete. Kao što je rečeno, ovaj algoritam minimizuje sume kvadrata rastojanja podataka od odgovarajućih težišta. Ako su podaci svi pikseli na slici, predstavljeni svojim bojama, nakon izvršavanja algoritma, za njihovu aproksimaciju se mogu koristiti odgovarajuća težišta. Ona su izabrana tako da budu u proseku najsličnija svim bojama iz klastera. Jedino preostalo pitanje je kako predstaviti piksele tako da se algoritam može primeniti na njih, ali odgovor na to pitanje je jednostavan, pošto RGB sistem već nudi reprezentaciju boje piksela nad kojom se mogu računati proseci i rastojanja. Na slici 12.9 dat je primer smanjenja palete polazne slike na ukupno 10 boja, na dva načina – nasumičnim izborom od 166075 boja sa polazne slike i pomoću klasterovanja algoritmom  $k$  sredina. U oba slučaja, rezultujuće slike se dobijaju tako što se boja svakog piksela zameni najbližom bojom iz paleta. Očigledno, pristup zasnovan na klasterovanju daje bolji rezultat, iako se može primetiti jasna granica između dve nijanse neba, nedostatak crvene boje u markaciji na kamenu i sivkasta boja zemljišta.



Slika 12.9: Slika durmitorskog vrha Zupci, slika koja je od nje dobijena smanjenjem palete na 10 nasumično izabranih od 166075 boja sa polazne slike i slika koja je od polazne dobijena smanjenjem palete algoritmom  $k$  sredina i korišćenjem težišta.

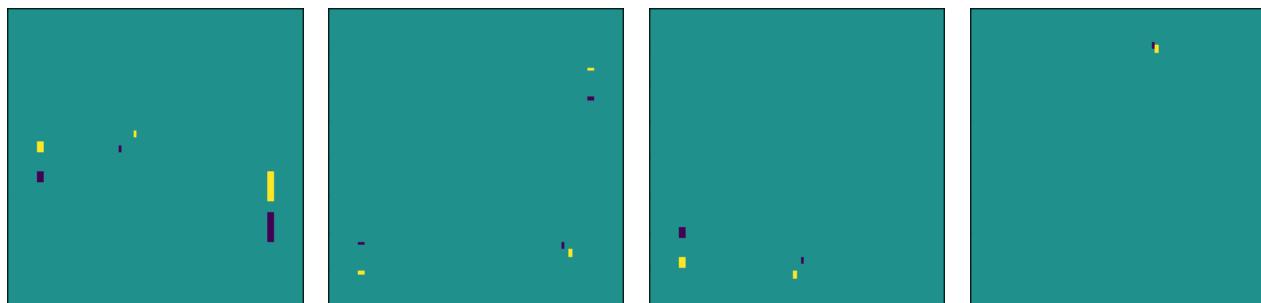
## Učenje potkrepljivanjem

Mnogi problemi prirodno se rešavaju nizom akcija koje svojim zajedničkim efektom dovode do cilja. Očigledan primer je igranje strateških igara poput šaha, ali i mnogi problemi robotike, planiranja, upravljanja i slično. Pritom, u takvima problemima često za neku akciju u nizu nije lako reći da li je bila ispravna ili nije. Na primer, u šahu, gubitak figure često je loš, ali se u nekim situacijama figure i namerno žrtvuju kako bi na duže staze bio postignut neki cilj. U takvima problemima obično je lako reći samo da li je problem uspešno rešen čitavim nizom akcija, a preraspodela zasluga za uspeh ili neuspeh na pojedinačne akcije je netrivijalna. Takođe, ishod neke akcije zavisi od konteksta u kojem je preduzeta. U šahu, uzimanje protivnikove figure može imati različite posledice zavisno od toga da li je ta figura bila čuvana ili nije. Ovo je tipičan scenario u kojem se koristi takozvano *učenje potkrepljivanjem* (eng. *reinforcement learning*), koje predstavlja pristup mašinskom učenju inspirisan načinom učenja pomoću pokušaja i grešaka, često prisutnom kod životinja. Cilj ove vrste učenja je da pomoći iskustava iz mnoštva pokušaja rešavanja problema nauči kako da u različitim situacijama u toku rešavanja problema bira adekvatne akcije.

Inspirisana razmatranjima nalik prethodnom, postavka učenja potkrepljivanjem prepostavlja postojanje *agenta* koji je u stanju da svojim *akcijama* interaguje sa *okolinom*. Pri svakoj interakciji menja se *stanje* u kojem se agent nalazi, a agent dobija odeđenu *nagrodu*, predstavljenu relativenim brojem. Cilj agenta je da nauči *politiku*, preslikavanje iz skupa stanja u skup akcija, koje koristi za odlučivanje, tako da maksimizuje sumu svih nagrada koje dobije u procesu interagovanja. Rešavanje problema pomoći učenja potkrepljivanjem zahteva najpre definisanje pomenutih elemenata. Skupovi akcija i stanja su nekad određeni definicijom problema, ali ih je nekada moguće i pogodno definisati ili aproksimirati u cilju efikasnijeg rešavanja problema. Jedan od najvažnijih izbora je definisanje nagrada. Potrebno je definisati i funkcionalnu formu politike, ali to razmatranje zbog njegove tehničke prirode ostavljamo za kasnije i ne diskutujemo ga u narednim primerima.

**Primer 13.1.** *U slučaju šaha, agent je igrač. Okruženje čine igrač protivnik, tabla i figure (prepostavljamo da se ne koristi sat). Stanja su uređene 64-vorke od kojih svaka može biti jedna šahovska figura ili prazno polje. Akcije koje agent može da preduzme su legalni šahovski potezi u datom rasporedu figura na tabli. Nagrade mogu biti 1 za potez kojim agent dobija partiju, odnosno -1 za potez kojim je gubi, a 0 u svim ostalim potezima. Tipična greška prilikom izbora nagrada bila bi da se nagrade definišu kao pozitivne vrednosti pri svakom potezu kojim agent uzima protivničku figuru. Na ovaj način bi agent mogao da nauči da uzme veliki broj protivničkih figura, čak i po cenu toga da izgubi partiju. Otud je vrlo važno prilikom definisanja nagrada voditi računa da politika koja vodi maksimalnoj nagradi zaista vodi rešenju problema.*

**Primer 13.2.** *U igri Pong dva igrača se dobacuju lopticom sa ciljem da suprotni igrač ne uspe da vrati lopticu, pri čemu se loptica može odbiti reketom koji se kreće vertikalno duž ivice ekrana koja odgovara datom igraču. Agent je ponovo igrač. Okruženje čine loptica, zakoni njenog kretanja i drugi igrač. Stanja su u idealnom slučaju određena lokacijom loptice, njenim vektorom kretanja, brzinom njenog kretanja i istim tim parametrima za rekeete oba igrača. Za razliku od prethodnog primera, ovde se javlja jedan od ključnih problema vezanih za predstavljanje stanja. Igra ne daje neposredno informacije o vektorima pravaca kretanja i brzini objekata, već generiše slike jednu za drugom u skladu sa kretanjem objekata. Stoga se ove veličine ne mogu direktno koristiti prilikom učenja. Ove informacije mogle bi da se aproksimiraju tako da se stanje definiše kao razlika dve slike dobijene u susednim trenucima. Sve pomenute veličine vidljive su u takvoj reprezentaciji, kao što je ilustrovano slikom 13.1. Akcije su pomeranje reketa gore ili dole i mirovanje. Što*



Slika 13.1: Tri stanja igre Pong. Zuta boja označava vrednost 1, a ljubicasta vrednost  $-1$ . U prvom stanju se vidi da se loptica kreće gore desno, a oba reketa nagore, pri čemu desni značajno brže nego levi. Na drugoj se loptica kreće dole desno, levi reket sporo nadole, a desni sporo nagore. Na trećoj se loptica kreće dole levo, levi reket poro nadole, a desni miruje. Na četvrtoj slici se loptica vrlo sporo kreće dole desno, a oba reketa miruju.

*se tiče nagrada, ponovo je dobro dati nagradu 1 kada agent dobije partiju,  $-1$  kada je izgubi i 0 u svakom drugom koraku. Na prvi pogled, možda bi imalo smisla dati agentu nagradu 1 kad god uspešno vrati lopticu. Međutim, u tom slučaju bi optimalna politika bila takva da agent svaki put vrati lopticu ali tako da protivnik takođe može lako da je vrati. Naime, što se igra duže igra, to vodi većem broju prikupljenih nagrada.*

**Primer 13.3.** Neka je potrebno upravljati helikopterom tako da se kreće unapred zadatom putanjom od jedne do druge tačke u praznom prostoru. Poželjno je zadati put preći što brže. Kako su helikopteri skupi, prepostavka je da se učenje primarno obavlja u simulatoru. Agent je vozač helikoptera, a okruženje je definisano fizičkim zakonima koji utiču na letenje - gravitacijom, mehaničkim svojstvima, itd. Stanje bi moglo biti definisano vektorom položaja težišta helikoptera u odnosu na neki fiksirani koordinatni sistem, izvodom tog vektora (koji daje vektor brzine kretanja), vektorom orientacije helikoptera (npr. vektorom težište-nos) i njegovim izvodom. Ovakva definicija stanja ima nekoliko manjkavosti. Prvo, ona možda ne sadrži sve relevantne informacije. Na primer, drugi izvodi pomenutih vektora su takođe relevantni jer predstavljaju ubrzanja. Drugo, za razliku od prethodnih primera, ove veličine su neprekidne, što nosi određene izazove u procesu učenja. Ukoliko se ovakva definicija stanja ispostavi manjkavom u praksi, može se probati sa drugačijom. Akcije se mogu definisati u terminima komandi koje pilot ima na raspolaaganju. I takve akcije su neprekidne jer uključuju pomeranje upravljačke palice u neprekidnom prostoru. Kako i to nosi tehničke izazove za algoritme učenja, ove akcije se mogu diskretizovati, ali postoje i algoritmi učenja koji to ne zahtevaju. U prethodnim primerima, vreme je bilo diskretno. U prvom primeru potpuno prirodno, a u drugom uslovljeno brojem slika koje mogu biti prikazane u sekundi. U ovom primeru vreme je neprekidno. To se može promeniti ili diskretizacijom vremena ili prelaskom na komplikovanije algoritme učenja koji bi umesto sumiranja, vršili integraciju nagrade. Jedna ideja za definisanje nagrade bi mogla biti negativna vrednost vremena potrebnog za sticanje do cilja. Očigledno, ovakva nagrada ne forsira let duž unapred definisane putanje, već favorizuje pravu liniju između tačaka. Stoga bi nagrada koja se daje u svakom koraku mogla biti definisana kompozitno – tako da različiti faktori od značaja daju svoje doprinose ukupnoj nagradi. Jedna komponenta bilo bi negativno rastojanje od definisane putanje, čime se favorizuje let po putanji ili u njenoj bliskoj okolini. Druga komponenta bila bi  $-1$  kako bi bilo favorizovano brže kretanje. Ove komponente mogu imati tazličite težine pri sumiranju, a koje određuju relativni značaj navedenih ciljeva.

U navedenim primerima od prvog ka poslednjem raste sloboda u definisanju relevantnih elemenata problema učenja, pa čak i u definisanju samog problema, kao i komplikovanost rešenja. U praksi je neretko situacija kao u trećem primeru. Ipak, prvi primer sakriva pravu složenost problema koji se razmatra. To što je ponekad, kao u njemu, lako definisati osnovne elemente problema učenja, ne znači da su tako izabrane definicije pogodne za uspešno učenje. U konkrentnom primeru zaista i nisu, što će postati jasnije kroz dalju diskusiju.

## 13.1 Markovljevi procesi odlučivanja i njihovo rešavanje

Markovljev proces odlučivanja je uobičajeni formalni okvir u kojem se diskutuju problemi koji se rešavaju učenjem potkrepljivanjem. Njegovo razumevanje je prvi korak ka konstrukciji algoritama u ovoj oblasti.

**Definicija 13.1.** Markovljev proces odlučivanja je uređena šestorka  $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \tau, p, \gamma)$  gde je  $\mathcal{S}$  skup stanja,  $\mathcal{A}$  je skup akcija,  $\mathcal{R} \subseteq \mathbb{R}$  je skup nagrada,  $\tau$  je trajektorija, odnosno niz slučajnih promenljivih

$$S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2, \dots$$

takvih da važi  $S_t \in \mathcal{S}$ ,  $A_t \in \mathcal{A}$  i  $R_t \in \mathcal{R}$  za svako  $t \geq 0$ ,  $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  je funkcija prelaska takva da za svako  $t \geq 0$  važi

$$p(s', r | s, a) = P(S_{t+1} = s', R_t = r | S_t = s, A_t = a)$$

i  $\gamma$  je realan broj iz intervala  $[0, 1]$  koji se naziva faktorom umanjenja.

**Primer 13.4.** Prethodni primeri već ilustruju koncepte stanja, akcija i nagrada, pa samim tim i trajektorije koja predstavlja niz tih veličina. Funkcija prelaska u slučaju šaha i Ponga je deterministička, odnosno za dato stanje  $s$  i nagradu  $a$ , za tačno jedan par vrednosti  $s'$  i  $r$  važi  $p(s', r | s, a) = 1$ , dok za ostale parove važi  $p(s', r | s, a) = 0$ . Na primer, u slučaju šaha, jednim potezom  $a$  iz datog stanja  $s$  prelazi se u tačno jedno stanje  $s'$  pri čemu se dobija (jednoznačno određena) nagrada  $r$  u skladu sa definicijom nagrada za taj problem. Verovatnoća da se odigravanjem istog poteza u istom polaznom stanju pređe u neko stanje  $s'' \neq s'$  ili da se pri tome dobije neka nagrada  $r'' \neq r'$  jednaka je 0.

U slučaju vožnje helikoptera ne mogu se uzeti u obzir svi zamislivi faktori koji na njegovo kretanje utiču. Na primer, u zavisnosti od ponašanja vetra, helikopter iz jednog stanja pri dатoj akciji može preći u različita, iako bliska, nova stanja. Otud funkcija prelaska ne mora biti deterministička.

Definicija Markovljevog procesa odlučivanja podrazumeva da verovatnoće prelaska iz jednog stanja u drugo, pri preduzimanju neke akcije ne zavise ni od čega osim od polaznog stanja i preduzete akcije:

$$p(s', r' | s, a) = P(S_t = s', R_{t-1} = r | S_t = s, A_t = a)$$

Drugim rečima, da bi neka struktura bila Markovljev proces odlučivanja treba da važi:

$$P(S_t, R_{t-1} | S_0, A_0, R_0, \dots, S_{t-1}, A_{t-1}) = P(S_t, R_{t-1} | S_{t-1}, A_{t-1})$$

Ovo svojstvo se naziva *Markovljevim svojstvom* i govori da ako je poznato trenutno stanje procesa, za zaključivanje o njegovoj budućnosti poznavanje prošlosti nije potrebno. Odnosno, ako znamo trenutno stanje, nije bitno kako se do njega stiglo. Recimo, u slučaju šaha, stanje table i informacija koji je igrač na potezu daju dovoljnu informaciju za odlučivanje o daljem toku igre. Kako se do tog stanja stiglo, za potrebe budućeg odlučivanja potpuno je nebitno.

Primetimo da trajektorije mogu biti konačne, kao što su u slučaju šaha, ali su lako zamislive i beskonačne trajektorije. Njihove realizacije (odnosno nizove realizacija slučajnih promenljivih koje ih čine) nazivamo episodama, pa i epizode mogu biti konačne ili beskonačne. Generisanje epizode je posledica interakcija agenta sa okolinom. Jedan deo generišućeg mehanizma zavisi od funkcije prelaska koja definiše okolinu, dok je drugi definisan ponašanjem agenta. Ponašanje agenta može se formalizovati *politikom*  $\pi(a|s)$ , raspodelom verovatnoće nad akcijama za dato stanje, odnosno smatra se da će agent u stanju  $s$  preuzeti akciju  $a$  sa verovatnoćom  $\pi(a|s)$ . Politika je *deterministička* ukoliko je verovatnoća jedne od akcija jednaka 1, dok su verovatnoće ostalih jednakе 0. Kako je ponašanje agenta opisano isključivo politikom, u kontekstu agentovog odlučivanja, pojmovi agenta i politike mogu se koristiti naizmenično.

U učenju potkrepljivanjem pre svega marimo za ukupan *dubitak* koji agent ostvari u toku jedne epizode. Dobitak od koraka  $t$  se definiše kao red  $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$  uz konvenciju  $0^0 = 1$ . U slučaju beskonačnih epizoda, podrazumeva se da su nepostojeci slučajne promenljive u ovoj definiciji jednakе nuli. Primetimo da, iako nije eksplicitno označeno, dobitak zavisi od politike.

Sada je moguće objasniti i faktor umanjenja. U slučaju da su nagrade ograničene, što je prihvaljiva pretpostavka, ako je  $\gamma < 1$ , dobitak konvergira. Pored ovog čisto tehničkog značaja, faktor umanjenja ima i intuitivni smisao – omogućava nam da vagamo značaj kratkoročnih i dugoročnih nagrada. Ukoliko važi  $\gamma = 0$ , politika koja maksimizuje takav dobitak bila bi pohlepna – u svakom koraku bi birala akciju koja nudi najveću nagradu koja se može dobiti u tom koraku. Za veće vrednosti  $\gamma$  optimalna politika uzima u obzir buduće nagrade sa većom težinom.

Pod rešavanjem Markovljevog procesa odlučivanja podrazumeva se pronalaženje politike koja vodi maksimalnom očekivanom dobitku agenta, pri čemu je očekivanje u odnosu na zajedničku raspodelu svih promenljivih u trajektoriji

$$\tau = S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2, \dots$$

odnosno u odnosu na raspodelu

$$P(\tau) = P(S_0) \prod_{t=0}^{\infty} \pi(A_t|S_t) p(S_{t+1}, R_t | S_t, A_t) \quad (13.1)$$

gde je  $P(S_0)$  raspodela po mogućim polaznim stanjima agenta,  $\pi$  politika prema kojoj agent donosi odluke i  $p$  funkcija prelaska okoline. Raspodela polaznih stanja i funkcija prelaska će se smatrati fiksiranim i često izostavljati kako bi se pojednostavila notacija.

Očekivana nagrada koju agent dobija prateći politiku  $\pi$  iz nekog stanja  $s$

$$v^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

naziva se *funkcija vrednosti stanja*. Primetimo da je zbog Markovljeve pretpostavke nebitno o kom trenutku  $t$  se radi. Pored funkcije vrednosti stanja, često se govori o *funkciji vrednosti akcije u stanju*

$$q^\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

Ove funkcije se mogu lako povezati:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[G_t | S_t = s] \\ &= \sum_a \pi(a|s) \mathbb{E}[G_t | S_t = s, A_t = a] \\ &= \sum_a \pi(a|s) q_\pi(s, a) \end{aligned}$$

Primetimo intuitivnost ove veze. Očekivani dobitak u stanju  $s$  je prosek očekivanih dobitaka kada se u stanju  $s$  preduzme akcija  $a$ , pri čemu se svaki od tih dobitaka pri uprosečavanju otežava verovatnoćom akcije  $a$  čiji je rezultat. Moguće je izraziti i funkciju  $q$  preko funkcije  $v$ :

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}[R_t + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r | s, a) [r + \gamma \mathbb{E}[G_{t+1} | S_{t+1} = s']] \\ &= \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')] \end{aligned}$$

Primetimo da je lakše iz funkcije  $q$  dobiti funkciju  $v$ , nego obrnuto. Naime, prvi smer podrazumeva da je poznata politika  $\pi$  što je u praksi tipično slučaj, pošto se politika koristi za upravljanje agentom. Drugi smer podrazumeva da je poznata funkcija prelaska, odnosno da je poznato funkcionisanje okruženja, što često nije tačno jer je okruženje osim u retkim slučajevima, poput igara, teško precizno opisati. Na primer, u slučaju vožnje helikoptera teško je tačno opisati kretanja vazduha koja bi mogla uticati na ishode akcija pilota u određenim stanjima.

Funkcije vrednosti omogućavaju merenje kvaliteta politika. Naime, ukoliko za sva stanja  $s \in \mathcal{S}$  važi

$$v_{\pi_1}(s) \geq v_{\pi_2}(s)$$

jasno je da je prva politika bolja od druge ili joj je bar jednaka. Time je definsan parcijalni poredak nad politikama  $\pi_1 \geq \pi_2$ . Ono što je zanimljivo je da za svaki Markovljev proces odlučivanja postoji bar jedna *optimalna politika*  $\pi_*$  – politika koja je bolja ili jednaka od svih ostalih politika. Sve optimalne politike dele jednu funkciju vrednosti stanja i jednu funkciju vrednosti akcije u stanju:

$$v_*(s) = \max_\pi v_\pi(s)$$

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

za sve  $s \in \mathcal{S}$  i  $a \in \mathcal{A}$ . Primetimo i neke prirodne veze između ovih funkcija. Na primer, važi

$$v_*(s) = \max_a q_*(s, a)$$

pošto se najveći očekivani dobitak u nekom stanju dobija preduzimanjem najbolje akcije u tom stanju. Naravno, ovo se može i formalno dokazati.

Primetimo da poznavanje funkcije  $q_*$  omogućava izračunavanje bar jedne optimalne politike. Naime, kako je u svakom stanju najpoželjnija akcija ona koja daje najveću očekivanu nagradu u datom stanju, politka

$$\pi_*(a|s) = \begin{cases} 1 & \text{ako važi } a = \arg \max_a q_*(s, a) \\ 0 & \text{inače} \end{cases}$$

je optimalna. Otud je poželjno imati način izračunavanja funkcije  $q_*$ . Jedan način da se to postigne pružaju takozvane *Belmanove jednakosti* koje izražavaju rekurentne veze funkcija vrednosti. Konkretno, za funkciju  $v_*$  važi:

$$\begin{aligned} v_*(s) &= \max_a q_*(s, a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_t + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma \mathbb{E}[G_{t+1} | S_{t+1} = s']] \\ &= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')] \end{aligned}$$

Belmanova jednakost za funkciju  $q$  je:

$$q_*(s, a) = \sum_{s', r} p(s', r | s, a) \left[ r + \gamma \max_{a'} q_*(s', a') \right]$$

Nije teško dokučiti intuitivni smisao poslednje jednakosti. Naime, dobitak pri preduzimanju akcije  $a$  u stanju  $s$  zavisi od neposredno dobijene nagrade i najvećeg dobitka koji se može dobiti iz stanja u koje se dospe nakon preduzimanja akcije. Pritom, kako su različiti prelasci u naredna stanja različito verovatni, potrebno je neposredno dobijene nagrade i buduće dobitke otežati verovatnoćama tih prelaza.

Ove jednačine omogućavaju izračunavanje funkcija vrednosti jednostavnim iterativnim postupcima

$$v_{i+1}(s) \leftarrow \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_i(s')]$$

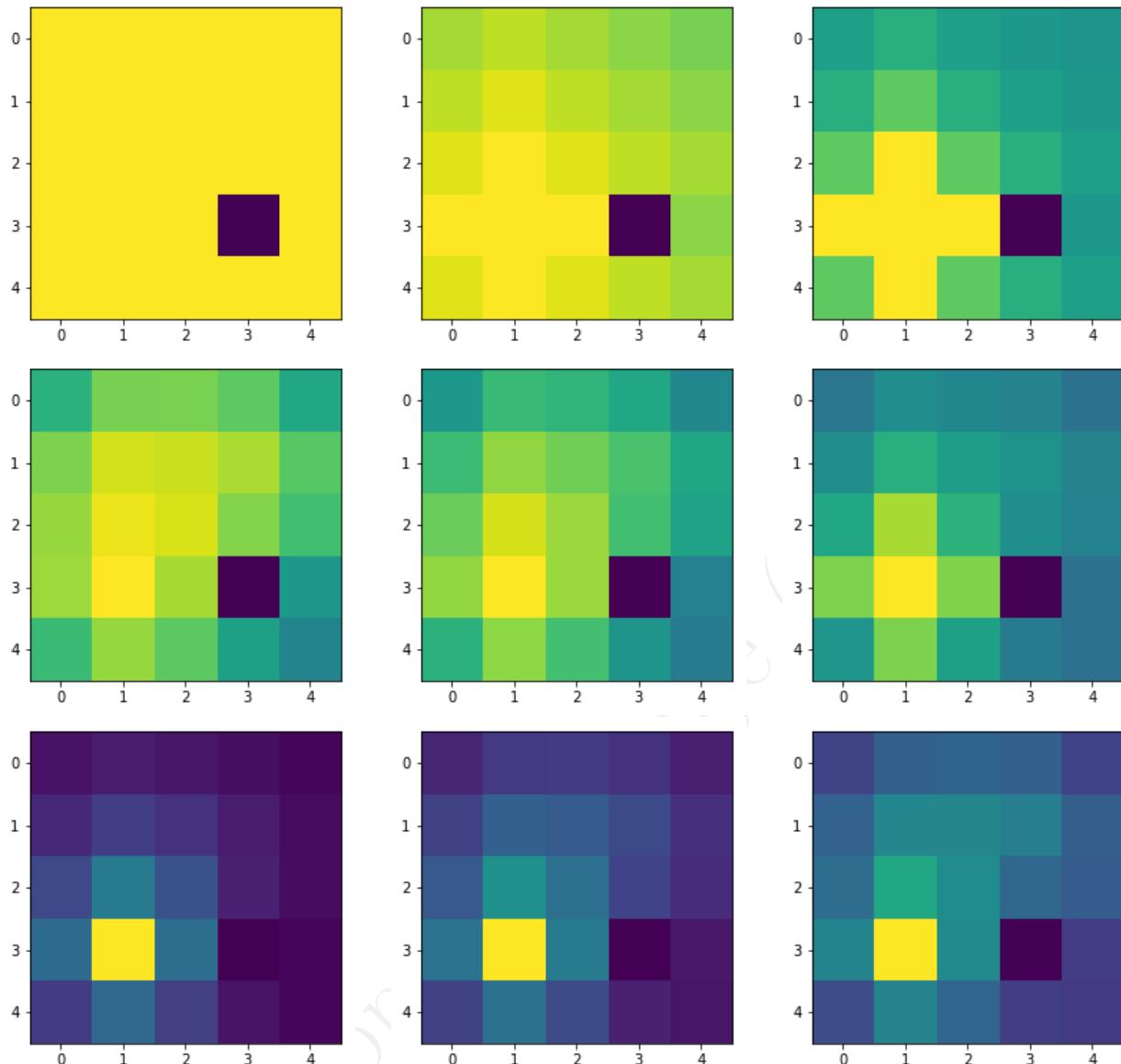
za svako  $s$  i

$$q_{i+1}(s, a) \leftarrow \sum_{s', r} p(s', r | s, a) \left[ r + \gamma \max_{a'} q_i(s', a') \right]$$

za svako  $s$  i  $a$ . Ovi postupci konvergiraju za proizvoljne inicijalne vrednosti  $v_0$  i  $q_0$  u slučaju konačnih skupova  $\mathcal{S}$  i  $\mathcal{A}$ . Međutim, ako su ovi skupovi velike kardinalnosti, ova konvergencija će biti previše spora za praktičnu upotrebu. Ako su beskonačni, onda ovaj algoritam nije ni primenljiv.

**Primer 13.5.** Zamislimo jednostavan svet – tablu dimenzija  $5 \times 5$  polja, koja ima rupu umesto polja (3, 3) i cilj na polju (3, 1) (indeksiranje počinje od 0). Agent može da se kreće po tabli po jedno polje levo, desno, gore i dole. Ipak, preduzimanje jedne takve akcije ne mora uvek da rezultuje željenim ishodom, već se to dešava sa verovatnoćom  $p$ , a sa verovatnoćom  $1 - p$  se nasumice bira jedna od preostale tri mogućnosti. Pad sa table rezultuje nagradom  $-1$  i krajem epizode, dostizanje cilja nagradom  $1$  i krajem epizode, a ostali ishodi nagradom  $0$ .

Pitanje je koja je funkcija vrednosti stanja  $v$  za neke konkretne vrednosti parametara  $p$  i  $\gamma$ . Za različite vrednosti tih parametara funkcija  $v$  data je na slici 13.2. Opšti je trend da su vrednosti više ako je polje bliže cilju, a dalje od rupe i ivica, a da su manje ako je polje dalje od cilja, a bliže rupi i ivicama. U slučaju  $p = 1$  i  $\gamma = 1$ , svako polje osim rupe ima vrednost 1. To je tako jer je sa svakog drugog polja moguće bez greške doći do cilja. Za  $p = 1$ , smanjivanjem faktora umanjenja, očekivano, polja bliža cilju postaju vrednija. Preciznije, ako je  $p = 1$ , vrednost svakog polja osim cilja i rupe je  $\gamma^{k-1}$  gde je  $k$  minimalan broj poteza potrebnih za dolazak do cilja. Ipak, to nije nužno tako i za druge vrednosti parametra  $p$ . Za  $p = 0.25$ , odnosno u slučaju nasumičnog kretanja, recimo vidi se da polja iznad ciljnog postaju vrednija sa smanjenjem parametra  $\gamma$ . Naime, polje neposredno iznad postaje vrednije zbog neposredne nagrade kojoj doprinosi i mogućnost poteza naniže sa vrednošću 1, a koja se ceni više nego dugoročne negativne nagrade koje dominiraju tablom. Povećanje vrednosti tog polja dalje pogoduje povećanju vrednosti njemu okolnih polja.



Slika 13.2: Funkcije vrednosti stanja zaokružena na dve decimale, za različite vrednosti parametara  $\gamma$  i  $p$ . Žuta boja označava vrednost 1, a ljubičasta vrednost  $-1$ . Po redovima, vrednosti parametra  $p$  su 1, 0.7 i 0.25. Po kolonama, vrednosti parametra  $\gamma$  su 1, 0.9 i 0.5.

*Pri smanjivanju vrednosti parametra  $p$ , ponašanje agenta postaje sve nasumičnije i šanse da nađe na polje sa nagradom se smanjuju, što se vidi kroz vrednosti polja.*

## 13.2 Učenje u nepoznatom okruženju

Prethodni algoritam za određivanje optimalne politike primenljiv je samo ako je okruženje poznato, odnosno ako je poznata njegova funkcija prelaska. U realnim primenama retko je slučaj da je ovo ispunjeno. Iako nam funkcionisanje različitih okruženja može biti razumljivo na nekom nivou intuicije, to nije dovoljno za primenu prethodnog algoritma. U takvim situacijama potrebno je paralelno sa određivanjem optimalne politike upoznavati i okruženje u kojem agent dela, što sugerira da agent mora istraživati svoje okruženje preduzimajući akcije koje će ga odvesti u različita stanja. Agent bi to mogao postići preduzimanjem nasumičnih akcija u različitim stanjima, ali bi takav algoritam bio previše spor jer ne bi koristio nikakvu informaciju od okruženja kao smernicu za efikasnije učenje. Postoji više principa na kojima se konstruišu ovakvi algoritmi, ali ćemo se fokusirati samo na jedan popularan princip zasnovan na Belmanovim jednakostima. Pritom, razmotrićemo prvo

učenje konačnim prostorima stanja i akcija, koje se direktno nadovezuje na prethodna razmatranja, a potom i učenje u beskonačnim prostorima koje nosi nove izazove.

### 13.2.1 Učenje u konačnim prostorima stanja i akcija

Primetimo da u Belmanovim jednakostima figuriše očekivanje zbiru tekuće nagrade i vrednosti budućeg stanja:

$$q_*(s, a) = \sum_{s', r} p(s', r | s, a) \left[ r + \gamma \max_{a'} q_*(s', a') \right]$$

Očekivanja se često ocenjuju uzoračkim prosekom, pri čemu se uzorak može dobiti tako što će agent interagovati sa okolinom. U ekstremnom slučaju, taj uzorak može sadržati samo jedno opažanje koje se dobija preuzimanjem neke akcije  $a$  u nekom stanju  $s$ , pri čemu se prelazi u stanje  $s'$  i dobija nagrada  $r$ :

$$r + \gamma \max_{a'} q_*(s', a')$$

Očigledan problem u ovom principu je da funkcija  $q_*$  koja figuriše u ovom izrazu nije poznata – upravo ona se ocenjuje. Ključni korak u izvođenju novog algoritma je korišćenje tekuće ocene ove funkcije u gornjem izrazu umesto njene stvarne vrednosti. Kako se ta funkcija menja u toku procesa njenog ocenjivanja, ovaj postupak je potrebno sprovoditi iterativno. Iterativni postupak omogućava i sakupljanje većeg broja opažanja. Ona se usrednjavaju pokretnim prosekom, odnosno linearnom interpolacijom između tekuće ocene funkcije vrednosti i one koja se dobija u tekucem koraku. Konkretno, algoritam ocene funkcije vrednosti akcije u stanju zasniva se na sledećem pravilu ažuriranja koje se sprovodi pri prelazu iz stanja  $s$  u stanje  $s'$  preuzimanjem akcije  $a$  uz nagradu  $r$ :

$$q(s, a) \leftarrow (1 - \alpha_t)q(s, a) + \alpha_t(r + \gamma \max_{a'} q(s', a'))$$

gde  $\alpha_t \in [0, 1]$  predstavlja brzinu učenja, odnosno parametar koji vaga značaj tekuće i novodobijene ocene u koraku  $t$ . Kao i u slučaju gradijentnog spusta, za konvergenciju je dovoljno da niz  $\alpha_t$  zadovoljava uslove

$$\sum_{t=0}^{\infty} \alpha_t = \infty \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

Iako je definisano pravilo ažuriranja ocene pri jednom prelazu, algoritam nije do kraja definisan. Naime, pitanje je koju akciju agent preuzima u nekom stanju kako bi generisao prelaz. Zanimljivo je da je dovoljno da akcije preuzimati tako da se svi parovi stanja i akcija posećuju beskonačno puta. Način na koji se ovo realizuje u praksi je korišćenje  $\varepsilon$ -pohlepne politike.  $\varepsilon$ -pohlepna politika u odnosu na datu funkciju  $q$  je politika koja sa verovatnoćom  $\varepsilon$  bira akciju nasumično sa jednakom verovatnoćom po svim akcijama, a u suprotnom pohlepno u odnosu na funkciju  $q$ , odnosno:

$$\pi_{q, \varepsilon}(a | s) = \begin{cases} 1 - \varepsilon & \text{ako } a = \arg \max_{a'} q(s, a') \\ \frac{\varepsilon}{|\mathcal{A}| - 1} & \text{inače} \end{cases}$$

Parametar  $\varepsilon$  treba da bude strogo veći od 0, ali se tipično prilikom treninga smanjuje. Jedna varijanta ovakvog postupka precizirana je algoritmom 13.3 i naziva se  $q$ -učenje (eng. *q-learning*). U datom algoritmu važi  $\alpha_t = 1/t$  i  $\varepsilon_t = 1/t$ , ali mogući su i drugi izbori.

Ovaj algoritam rešava problem nepoznavanja okruženja, ali ne i problem velikih ili beskonačnih skupova stanja i akcija, kakav je slučaj u diskutovanom primeru igranja šaha. Naime, u takvom slučaju nije jasno kako čuvati informacije o aproksimaciji  $q(s, a)$  u računaru i kako dovoljno puta posetiti sve parove stanja i akcija da aproksimacija bude dovoljno dobra za praktične potrebe. Dodatno, ovako treniran agent se ne može snaći ni u jednom stanju koje ranije nije video, čak i ako je to stanje slično već viđenim. Odnosno, algoritam ne generalizuje na nepoznate situacije i otud ga je teško smatrati algoritmom učenja. U slučaju šaha, nezamislivo je da će sva moguća stanja biti viđena prilikom treninga.

### 13.2.2 Učenje u beskonačnim prostorima stanja i akcija

Rešenje navedenih problema diskutovaćemo prvo zadržavši pretpostavku o konačnosti skupa akcija, što je dovoljno dobro za mnoštvo primena. Ključno zapažanje je da se funkcija  $q_*$  može učiti kao i u drugim pristupima mašinskog učenja – uvođenjem parametrizovanog modela  $q_w$  nad nekom prezentacijom stanja i akcija. Kako su akcije konačne, nije neophodno tražiti posebnu reprezentaciju za njih – mogu se tretirati kao bilo koje kategoričke promenljive ili čak neprekidne u slučaju da su definisane diskretizacijom neprekidne veličine. Što se

**Algoritam:** Algoritam  $q$ -učenja.

**Ulaz:** Broj iteracija  $N$

**Izlaz:** Aproksimacija funkcije  $q_*$

- 1: nasumično inicijalizuj  $q$  za sve parove  $s$  i  $a$ ;
- 2: inicijalizuj stanje  $s$  na polazno
- 3:  $t \leftarrow 1$
- 4: **ponavljam**
- 5:     preduzmi akciju  $a \sim \pi_{q,1/t}(a|s)$  i opazi nagradu  $r$  i novo stanje  $s'$
- 6:      $\alpha_t = 1/t$
- 7:      $q(s, a) \leftarrow (1 - \alpha_t)q(s, a) + \alpha_t(r + \gamma \max_{a'} q(s', a'))$
- 8:     **ako** je stanje  $s'$  završno **onda**
- 9:         inicijalizuj stanje  $s$  na polazno
- 10:      **inace**
- 11:         postavi stanje  $s$  na  $s'$
- 12:      $t \leftarrow t + 1$
- 13: **dok nije ispunjen** uslov  $t = N$
- 14: vrati  $q$  kao rešenje

Slika 13.3: Algoritam  $q$ -učenja.

tiče stanja, ili je potrebno predstaviti ih unapred definisanim skupom svojstava ili je potrebno upotrebiti model koji je u stanju da sam konstruiše takva svojstva, kao što je slučaj sa dubokim neuronskim mrežama. Pod tim prepostavkama, problem učenja potkrepljivanjem se svodi na sledeći optimizacioni problem:

$$\min_{\mathbf{w}} \mathbb{E}[(q_{\mathbf{w}}(s, a) - q_*(s, a))^2]$$

gde je očekivanje u odnosu na raspodelu nad stanjima i akcijama pri optimalnoj politici. Naravno, očekivanje se u praksi zamenjuje uzoračkim prosekom, čime se dobija problem:

$$\min_{\mathbf{w}} \sum_{s,a} (q_{\mathbf{w}}(s, a) - q_*(s, a))^2$$

gde je suma po parovima viđenim u skupu konkretnih epizoda. Ako bi ovaj problem bio rešavan gradijentnim spustom, formula za ažuriranje parametara bi bila:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_t \sum_{s,a} (q_{\mathbf{w}}(s, a) - q_*(s, a)) \nabla_{\mathbf{w}} q_{\mathbf{w}}(s, a)$$

a u ekstremnoj varijanti jednočlanog uzorka, koja predstavlja osnovu mnoštva metoda učenja potkrepljivanjem, dobija se:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_t (q_{\mathbf{w}}(s, a) - q_*(s, a)) \nabla_{\mathbf{w}} q_{\mathbf{w}}(s, a)$$

U ovom pristupu postoji očigledan problem što funkcija  $q_*$  nije poznata, ali je moguće pribegći ranije korišćenom triku aproksimacije na osnovu nagrade u tekućem koraku i tekuće aproksimacije, odnosno

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_t (q_{\mathbf{w}}(s, a) - (r + \max_{a'} q_{\mathbf{w}}(s', a'))) \nabla_{\mathbf{w}} q_{\mathbf{w}}(s, a)$$

Ključni problem ovakvog algoritma je taj što ne mora konvergirati osim u slučaju linearnih modela, koji u praksi nisu dovoljni. Postoji niz poboljšanja koja vode ka pouzdanijoj i bržoj konvergenciji, ali po pravilu algoritmi učenja potkrepljivanjem su osetno teži za primenu od algoritama nadgledanog učenja zbog nestabilnosti optimizacije.

U slučaju beskonačnih prostora akcija, prethodni algoritam nije lako upotrebljiv. Naime, čak i ako bi se uspešno naučila funkcija  $q(s, a)$ , za dato stanje, najbolja akcija bi se uvek morala tražiti nekim optimizacionima algoritmom nad funkcijom  $q$ . Prosto, funkcija vrednosti  $q$  ne predstavlja dovoljno dobru reprezentaciju znanja za ovakve probleme. Umesto nje, alternativni pristup, koji se često koristi i u drugim kontekstima učenja potkrepljivanjem zasniva se na direktnom učenju politike, umesto učenja funkcije  $q$ . Primetimo da je u slučaju

neprekidnih akcija politiku  $\pi_{\mathbf{w}}(a|s)$  moguće modelovati, na primer, normalnom raspodelom  $\mathcal{N}(f_{\mathbf{w}}(s), \sigma^2)$  za neko  $\sigma^2$ , gde  $f_{\mathbf{w}}(s)$  predstavlja, na primer, regresionu neuronsku mrežu. U tom slučaju, ta mreža za dato  $s$  direktno izračunava preporučene neprekidne akcije.

Kao što je rečeno, cilj učenja potkrepljivanjem je maksimizacija očekivanog dobitka u odnosu na zajedničku raspodelu promenljivih u trajektoriji definisanu jednakošću 13.1, odnosno potrebno je maksimizovati veličinu:

$$J(\mathbf{w}) = \mathbb{E}_{P_{\mathbf{w}}}[G_0]$$

pri čemu je  $P_w$  pomenuta zajednička raspodela u slučaju korišćenja politike  $\pi_w$ :

$$P_{\mathbf{w}}(\tau) = P(S_0) \prod_{t=0}^{\infty} \pi_{\mathbf{w}}(A_t|S_t) p(S_{t+1}, R_t|S_t, A_t)$$

Gradijent datog izraza nije trivijalno izračunati jer gradijent po parametrima  $\mathbf{w}$  ne može da prođe kroz očekivanje čija raspodela zavisi od  $\mathbf{w}$ . Ipak gradijent je moguće izvesti. U narednom izvođenju vodimo računa da ne mešamo slučajne promenljive i njihove realizacije. Očekivanja se primenjuju na slučajne promenljive, a izračunavaju se uobičajeno, tako što se integrale vrednosti njihovih realizacija. Stoga  $\mathbf{e}$  označava epizodu, odnosno realizaciju trajektorije  $\tau$ , a  $r(\mathbf{e})$  realizaciju  $\sum_{t=0}^{\infty} \gamma^t r_t$  odgovarajućeg dobitka  $G_0$ .

$$\begin{aligned} \nabla_{\mathbf{w}} J(\mathbf{w}) &= \nabla_{\mathbf{w}} \mathbb{E}_{P_{\mathbf{w}}}[G_0] \\ &= \nabla_{\mathbf{w}} \int r(\mathbf{e}) P_{\mathbf{w}}(\mathbf{e}) d\mathbf{e} \\ &= \int r(\mathbf{e}) \nabla_{\mathbf{w}} P_{\mathbf{w}}(\mathbf{e}) d\mathbf{e} \\ &= \int r(\mathbf{e}) \frac{\nabla_{\mathbf{w}} P_{\mathbf{w}}(\mathbf{e})}{P_{\mathbf{w}}(\mathbf{e})} P_{\mathbf{w}}(\mathbf{e}) d\mathbf{e} \\ &= \int r(\mathbf{e}) [\nabla_{\mathbf{w}} \log P_{\mathbf{w}}(\mathbf{e})] P_{\mathbf{w}}(\mathbf{e}) d\mathbf{e} \\ &= \mathbb{E}_{P_{\mathbf{w}}}[G_0 \nabla_{\mathbf{w}} \log P_{\mathbf{w}}(\tau)] \end{aligned}$$

Dalje, važi

$$\nabla_{\mathbf{w}} \log P_{\mathbf{w}}(\mathbf{e}) = \nabla_{\mathbf{w}} \left[ \log P(s_0) + \sum_{t=0}^T (\log \pi_{\mathbf{w}}(a_t|s_t) + \log p(s_{t+1}, r_t|s_t, a_t)) \right] = \sum_{t=0}^T \nabla_{\mathbf{w}} \log \pi_{\mathbf{w}}(a_t|s_t)$$

odnosno

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbb{E}_{P_{\mathbf{w}}}\left[G_0 \sum_{t=0}^T \nabla_{\mathbf{w}} \log \pi_{\mathbf{w}}(a_t|s_t)\right]$$

Gradijent koji je izražen očekivanjem se uobičajeno aproksimira uzoračkim prosekom. Algoritam 13.4 prikazuje algoritam REINFORCE u svom najjednostavnijem obliku, pri čemu se gradijent aproksimira na osnovu uzorka veličine 1.

Dati algoritam je osnova velikog broja algoritama korišćenih u praksi. Njegova mana u odnosu na algoritme zasnovane na  $q$  funkciji je veća nestabilnost treninga, koja se često ublažuje dodatnim tehnikama koje izlaze van okvira ove knjige.

### 13.3 Primene učenja potkrepljivanjem

U nastavku prikazujemo nekoliko primera primena učenja potkrepljivanjem. Pritom, diskutovani algoritmi, iako bi se teorijski mogli primeniti na datim problemima, na njima ne bi dali upotrebljive rezultate. Stvarno korišćeni algoritmi su, iako izgrađeni na sličnim osnovima, ipak osetno komplikovani.

#### 13.3.1 Igranje igara sa Atarija

Iako nije zaista praktična, jedna od poznatijih primena učenja potkrepljivanje je igranje igara sa Atarija, poput prethodno pomenute igre Pong, Space Invaders i drugih. Za svaku igru, treniran je zaseban agent zasnovan na konvolutivnoj mreži. Potrebno je da agent igra igru samo na osnovu slike ekrana, isto kao i čovek. Model agenta je konvolutivna mreža čiji je ulaz reprezentacija stanja dobijena na osnovu slike, poput stanja diskutovanog za igru Pong ilustrovanog na slici 13.1. Pored ulaza, algoritam učenja dobija i nagrade na kraju partije. U slučaju Ponga, nagrada je 1 kad god je partija dobijena, a -1 kad je izgubljena.

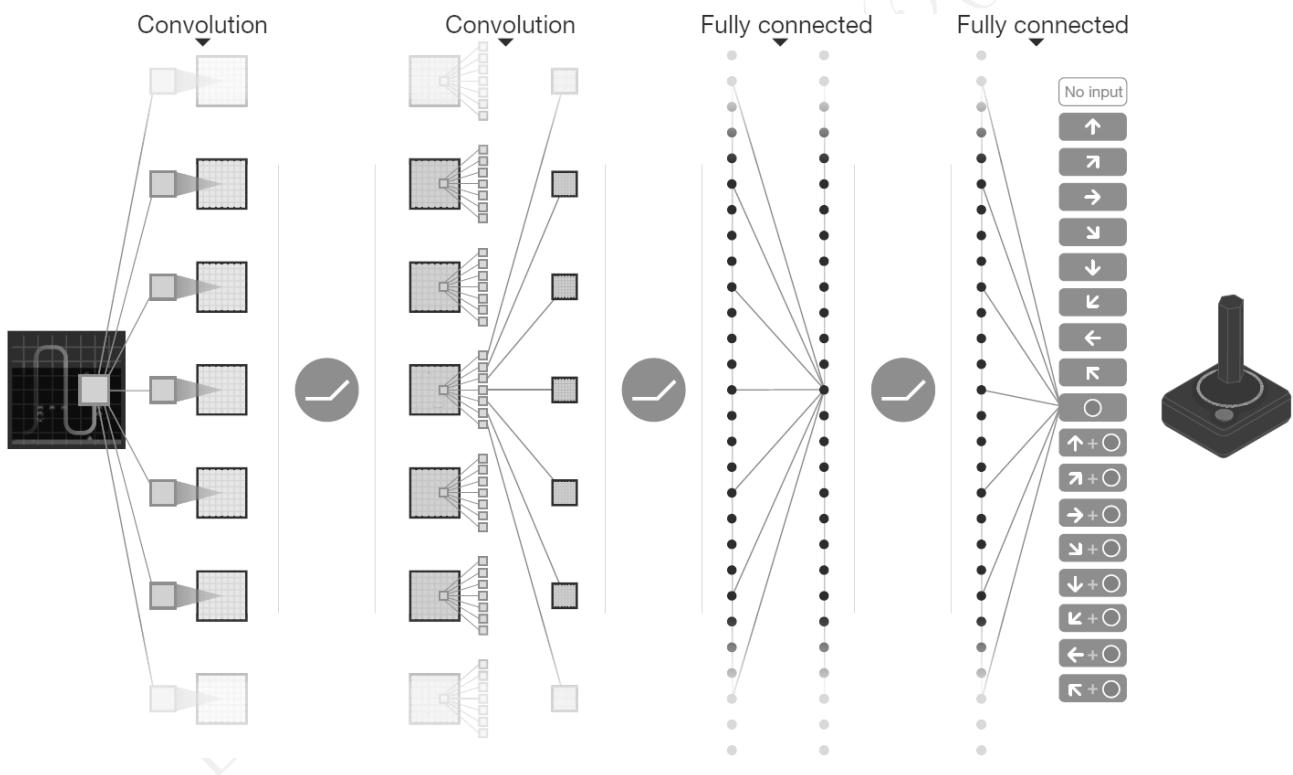
**Algoritam:** Algoritam REINFORCE.

**Ulaz:** Broj iteracija  $N$

**Izlaz:** Aproksimacija  $\pi_w$  optimalne politike

- 1: nasumično inicijalizuj  $w$
- 2:  $t \leftarrow 1$
- 3: **ponavljam**
- 4: generisati epizodu  $e$  korišćenjem politike  $\pi_w$
- 5: izračunati nagradu  $r(e)$
- 6:  $w \leftarrow w + \alpha \left[ r(e) \sum_{t=0}^T \nabla_w \log \pi_w(a_t | s_t) \right]$
- 7:  $t \leftarrow t + 1$
- 8: **dok nije ispunjen uslov  $t = N$**
- 9: vrati  $\pi_w$  kao rešenje

Slika 13.4: Algoritam REINFORCE.



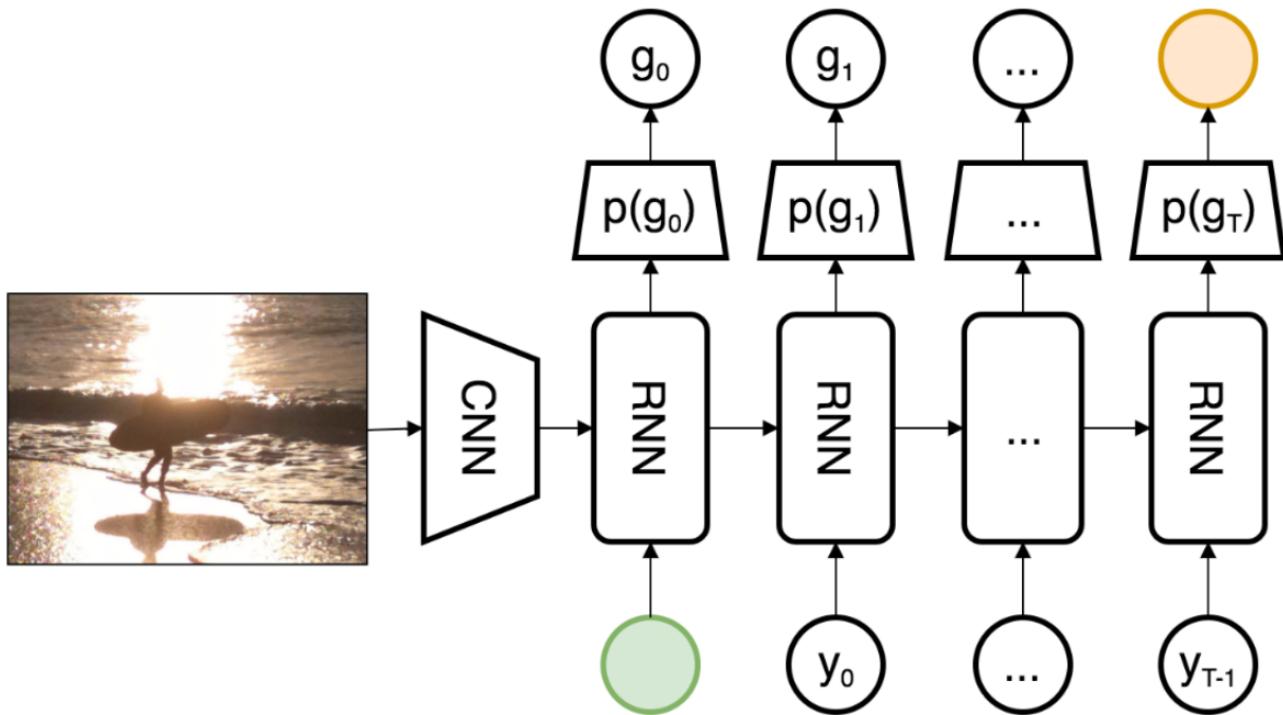
Slika 13.5: Arhitektura mreže za igranje igara sa Atarija.

Konvolutivna mreža koja se koristi u svim igrama modeluje funkciju vrednosti stanja i akcije. Ilustrovana je na slici 13.5. Zanimljivo je da umesto dva ulaza – stanja i akcije, ima samo jedan ulaz – sliku koja predstavlja stanje, dok za svaku akciju ima po jedan izlaz koji daje vrednost te akcije u datom stanju. Ovakva odluka je donesena kako se pri izboru poteza u datom stanju ne bi za svaku akciju iz početka ne bi vršilo izračunavanje u mreži, već se izračunavanje vrši samo jednom i odjednom se dobijaju vrednosti svih akcija i preduzima se najbolja.

Algoritam korišćen za treniranje agenta zasnovan je na pravilu  $q$ -učenja iz prehodnog odeljka:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_t (q_{\mathbf{w}}(s, a) - (r + \max_{a'} q_{\mathbf{w}}(s', a))) \nabla_{\mathbf{w}} q_{\mathbf{w}}(s, a)$$

Ipak, algoritam sadrži nekoliko poboljšanja čiji je cilj da stabilizuju proces učenja, što je potvrđeno empirijski. Prvo, trojke stanja akcija i nagrada se čuvaju u jednom baferu unapred definisane veličine kako bi se iz njih moglo ponovo učiti bez potrebe da se ponovo dese u igri. Mreža se u svakom koraku učenja trenira na osnovu



Slika 13.6: Gruba ilustracija arhitekture enkoder-dekoder mreže za opisivanje slika.

uzorka trojki izabranog iz tog bafera, a ne na osnovu poslednje trojke. Drugo, vrednosti  $q_w(s, a)$  i  $\nabla_w q_w(s, a)$  se izračunavaju na osnovu tekuće verzije mreže, dok se vrednost  $r + \max_{a'} q_w(s', a)$  izračunava na osnovu takozvane „zamrznute“ verzije te mreže čiji se parametri ne ažuriraju u svakom koraku učenja, već povremeno, recimo na nekoliko hiljada gradijentnih koraka, prosto preuzimajući parametre iz tekuće mreže.

Ovaj algoritam postiže rezultate u nivou profesionalnih ljudskih igrača na 49 različitih igara sa Atarija.

### 13.3.2 Opisivanje slika tekstrom

U glavi o nadgledanom učenju, pomenut je jedan pristup opisivanju slika koji se zasniva na enkoder-dekoder arhitekturi, nalik prevođenju teksta sa jednog na drugi jezik, pri čemu se slika kodira pomoću konvolutivne mreže, a potom se generisanje teksta, odnosno dekodiranje vrši rekurentnom LSTM mrežom, kao što je ilustrovano na slici ???. Pritom, pretpostavlja se da je za svaku sliku dato nekoliko tačnih rečenica, imajući u vidu da različiti ljudi ne bi istovetno opisali sliku. Kako je izbor reči u svakom koraku generisanja klasifikacioni problem, osnovni pristup bi podrazumevao korišćenje unakrsne entropije kao funkcije greške prilikom treniranja mreže. Međutim, primećeno je da rečenice koje se dobijaju, često ne liče dovoljno na rečenice kakve bi formulisali ljudi. Primera radi, jedan od problema je ponavljanje istih reči ili sintagmi u izlazu, ali svakako ima i drugih. Istraživači godinama koriste alternativne metrike za evaluaciju tekstova koje imaju za cilj da uporede ispravan tekst sa mašinski generisanim tekstrom. Jedna jednostavna metrika je BLEU. Neka je data jedna generisana rečenica dužine  $N$  i nekoliko referentnih rečenica koje se smatraju ispravnim. Tada je za svaku reč generisane rečenice moguće izračunati maksimalan broj  $m$  njenih pojavljivanja među referentnim rečenicama i broj njenih pojavljivanja  $n$  u generisanoj rečenici. Tada se toj reči može pridružiti skor  $\min(m, n)/N$ , a ovakvi skorovi se mogu uprosećiti po svim generisanim rečima, što predstavlja BLEU metriku. Očigledno, prekomerno ponavljanje neke reči u generisanoj rečenici vodi povećavanju broja  $N$  i time smanjenju skora. Ovo je samo jedna jednostavna metrika, koja između ostalog ima svojstvo da kažnjava nepotrebna ponavljanja. Postoji mnoštvo drugih metrika koje reflektuju različita svojstva generisanog teksta, a najsfisticiranije počivaju na definisanju grafova koji oslikavaju semantiku rečenica i čijim uparivanjem se računa poklapanje rečenica po smislu. Ovakve metrike se često linearno kombinuju kako bi ukupna metrika uzela u obzir sve aspekte koje pojedinačne metrike mere. Dodatno, istraživanja sa ljudima su pokazala da neke od ovih metrika dobro koreliraju sa ljudskim sudom o dobrom prevodu. Otud bi bilo poželjno koristiti ovakve metrike kao funkcije greške u procesu treniranja. Međutim, one nisu diferencijabilne i čak su deo po deo konstantne u odnosu na parametre modela, što predstavlja veliki problem za gradijentne metode. Jedna od ideja kako bi se učenje zasnovalo na optimizaciji ovakvih metrika počiva na učenju potkrepljivanjem.

Primetimo da ovaj problem predstavlja problem nadgledanog učenja imajući u vidu da su za slike dati tačni opisi. Otud bi se trebalo zapitati zašto bi učenje potkrepljivanjem bilo adekvatan pristup učenju u ovakvom kontekstu. U uobičajenom kontekstu nadgledanog učenja, očekuje se da funkcija greške ima upotrebljiv gradijent, što ovde, kao što je već konstatovano, nije slučaj. S druge strane, u učenju potkrepljivanjem, informacija o kvalitetu modela može se dati nagradom, za koju nema zahteva da bude diferencijabilna u odnosu na parametre modela. Otud je ključni razlog za korišćenje učenja potkrepljivanjem tehnički, a ne suštinski, ali ništa manje važan ukoliko je potrebno optimizovati pomenute metrike.

Problem se formuliše kao problem učenja potkrepljivanjem na sledeći način. Agent se modeluje rekurentnom neuronskom mrežom poput one na slici ???. Stanje agenta se predstavlja skrivenim stanjem mreže koje se na početku inicijalizuje na izlaz konvolutivnog enkodera. Akcije su reči. Epizoda se sastoji u generisanju rečenice kao niza uzastopno preduzetih akcija – pojedinačnih reči. Nagrada je 0 u svakom koraku, osim u poslednjem, kada je nagrada jednak vrednosti metrike koja se optimizuje za tako generisani rečenicu i skup referentnih rečenica. Algoritam koji se koristi u treningu zasnovan je na algoritmu REINFORCE, ali koristi niz relativno naprednih trikova koji stabilizuju i ubrzavaju proces optimizacije.

Ovaj primer pruža još važniji nauk od toga kako opisivati slike pomoću mašinskog učenja. On ukazuje da je generalno moguće optimizovati nediferencijabilne funkcije greške ako se problem nadgledanog učenja formuliše kao problem učenja potkrepljivanjem. Pritom se plaća cena sporijeg i nestabilnijeg treninga, kao i kompleksnijeg dizajna algoritma kojim se nastoje ublažiti ti problemi.

### 13.3.3 Automatski dizajn novih lekova

Automatski dizajn novih jedinjenja predstavlja važan problem farmaceutske industrije i aktivna oblast istraživanja već oko 15 godina. Raniji pristupi su se obično zasnivali na optimizacionim metodama primenjenim na neku funkciju koja oslikava željena svojstva molekula. Ovaj problem je vrlo težak jer se procenjuje da je broj molekula koji se mogu sintetizovati, a koji predstavljaju potencijalne lekove između  $10^{30}$  i  $10^{60}$ . Ovde će biti prikazan jedan pristup zasnovan na učenju potkrepljivanjem.

Arhitektura podrazumeva postojanje dve mreže – jedne generativne, koja predlaže nove hemijski moguće molekule i igra ulogu agenta, odnosno politike i druge prediktivne, koja predviđa različita svojstva od interesa predloženog molekula (poput tačke topljenja, hidrofobnosti, broja benzenovih prstena, broja hemijskih grupa poput -OH, -NH<sub>2</sub>, itd) i koja predstavlja pomoćni element u svrhe definisanja nagrade. Obe mreže prvo se treniraju nadgledanim učenjem pomoću dostupnih obeleženih podataka (koje čini preko milion jedinjenja), a potom se generativna mreža trenira učenjem potkrepljivanjem. Teorijski, inicijalizacija nadgledanim učenjem nije neophodna, ali se u praksi često radi zarad brže konvergencije učenja potkrepljivanjem.

Molekuli su visoko strukturirani objekti kojima je teže baratati nego slikama i vektorima. Otud se prvo postavlja pitanje njihove reprezentacije u kontekstu neuronskih mreža. Na sreću, postoje tekstualna kodiranja grafova i specifično hemijskih struktura. Jedna takva reprezentacija se naziva SMILES i u njoj se recimo molekul aspirina zapisuje kao [CC(=O)OC1=CC=CC=C1C(=O)O], ali nećemo ulaziti u njenu sintaksu. Onda se skup stanja agenta može predstaviti kao skup validnih SMILES niski do neke unapred predviđene dužine. Akcije odgovaraju slovima SMILES azbuke. Nagrade su 0 u svim nezavršnim stanjima, a u završnom stanju s se definišu kao

$$r(s) = f(P(s))$$

gde je  $P$  prediktivna mreža, a  $f$  ručno dizajnirana funkcija čiji izbor zavisi od željenih svojstava jedinjenja. Na primer, ako je potrebno maksimizovati tačku topljenja i minimizovati hidrofobnost jedinjenja, funkcija  $f$  bi bila razlika procenjene tačke topljenja i procenjene hidrofobnosti, pri čemu su te procene dobijene od strane mreže  $P$ . Učenje se sprovodi REINFORCE algoritmom, pri čemu treba imati u vidu da se prediktivna mreža ne trenira nakon inicijalnog treninga nadgledanim učenjem.

Sintaksa SMILES niski, odnosno struktura molekula koju predstavljaju, nije jednostavna i obične rekurentne mreže se nisu pokazale dovoljno dobrim u njihovoj obradi, tako se umesto njih koriste specijalizovane arhitekture. Generativna mreža predstavlja rekurentnu mrežu sa pridruženim stekom, kojim je ona u stanju da upravlja – da na njemu čuva međurezultate svojih izračunavanja i da odlučuje kada će im ponovo pristupiti zarad daljeg izračunavanja. Prediktivna mreža je jednostavnija i zasnovana je na LSTM modelu.

Veliki problem postojećih mehanizama za dizajn novih molekula je što često generišu hemijski neispravne kandidate. Ispostavlja se da generativna mreža trenirana na ovaj način generiše validne molekule u 95% slučajeva, od čega manje od 0.1% od milion generisanih struktura predstavljaju molekule iz trening skupa, odnosno, mreža nije samo naučila trening podatke napamet. Štaviše, prema standardnoj metodologiji procene težine hemijske sinteze predloženih molekula, ispostavlja se da mreža skoro uvek predlaže molekule koje je lako sintetizovati.



Slika 13.7: Lebdenje helikoptera u mestu u obrnutom položaju.

#### 13.3.4 Autonomno upravljanje helikopterom

Autonomno upravljanje helikopterima smatra se težim problemom od upravljanja vazduhoplovima sa fiksiranim krilima. Upravljanje helikopterom pri malim brzinama posebno je teško. Dodatno, pri malim brzinama, jedan od najtežih manevara je lebdenje u mestu u obrnutom položaju — sa elisom naniže, kao što prikazuje slika 13.7. U ovom položaju poseban izazov je stabilnost, pošto je centar mase visoko. Ovaj manevr nije od velikog praktičnog značaja, ali je zanimljiv kao ozbiljan izazov za automatizaciju upravljanja, pa zato diskutujemo kako je rešen učenjem potkrepljivanjem. Navedeni pristup može se upotrebiti za učenje lebdenja i u drugim položajima, pretpostavljajući da su fizički održvi. Obrnuti položaj je samo zanimljiva ilustracija. U eksperimentima je korišćen helikopter dužine 150cm, visine 56 cm i težine oko 8kg i motorom od 46 kubika.

Opišimo prvo način upravljanja helikopterom. Većinom helikoptera se upravlja pomoću četiri kontrole:

- uzdužni nagib, kojim se helikopter nagnje napred ili nazad, čime se kontroliše ubrzanje u pravcu duž helikoptera;
- nagib u stranu, čime se helikopter nagnje levo ili desno, čime se kontroliše ubrzanje u stranu;
- brzina rotacije glavne elise, čime se kontroliše potisak koji elisa generiše;
- potisak pomoćnog rotora kojim se helikopter rotira levo-desno.

Bitan problem u učenju upravljanja helikopterom je taj što se pri tom učenju ne može eksperimentisati sa nasumično izabranim akcijama, kao što učenje potkrepljivanjem pretpostavlja, pošto su helikopteri skupi, a takve akcije mogu voditi njihovom padu. Otud je prvi korak izgradnja verodostojnog simulatora u kojem će potom biti vršeno treniranje. Simulator je modelovan nadgledanim učenjem na osnovu 391 sekunde ljudski kontrolisanog leta u toku kojeg je helikopter leteo u obrnutom položaju, pri čemu su pamćene informacije o letu 10 puta u sekundi. Svaki od ovih zapisa sadrži informaciju o stanju helikoptera i preduzetoj akciji pilota. Na osnovu ovih podataka, trenira se model nadgledanog učenja kojim se za dato stanje i akciju predviđa buduće stanje, a simulator na to predviđanje dodaje određeni šum, kako bi modelovao stohastičnost okruženja. Važno pitanje je naravno, šta su akcije i stanja. Akcije su 4 nabrojane kontrole numerički predstavljene brojevima u intervalu  $[-1, 1]$ . Reprezentacija stanja je nešto komplikovanija. Ona treba da izrazi poziciju, orientaciju i brzinu helikoptera. Ako bi se koordinatni sistem vezao za neku fiksiranu tačku spoljnog sveta i imao fiksiranu orientaciju, modelu simulatora bi bilo teško da nauči invarijantnost zakona fizike u odnosu na translacije i rotacije. Naime, ako je helikopter u jednoj tački prostora za datu akciju imao određenu promenu stanja, model bi usled malog uzorka mogao naučiti da je za tu promenu stanja bila bitna i lokacija na kojoj se helikopter nalazio, iako nije. Otud se spoljni koordinatni sistem koristi za definisanje uglova rotacije helikoptera oko tri ose (eng. *roll*, *pitch*, *yaw*) i ugaone brzine rotacije oko tih ose, ali se za lokaciju i ubrzanje koristi referentni sistem vezan za sam helikopter. Kako je u njemu lokacija konstantna, ona se ni ne čuva eksplicitno, već samo

brzina duž osa tog koordinatnog sistema. Stanje se onda definiše vektorom ubrzanja, uglova rotacije u ugaonih brzina. Prilikom modelovanja simulatora, model ne mora da predviđa uglove rotacije, pošto se oni uvek mogu dobiti integracijom na osnovu ugaonih brzina.

Kada je nadgledanim učenjem definisan simulator, koristi se učenje potkrepljivanjem kako bi se trenirao helikopter da leti u nekom (recimo obrnutom) položaju u odnosu na taj simulator. Akcije su očito kontinualne, pa je pogodno koristiti algoritam REINFORCE, a politiku predstaviti normalnom raspodelom u odnosu na izlaze mreže koja predviđa vrednosti za 4 akcije. Nagrada se može definisati na sledeći način:

$$-(\alpha_{\mathbf{x}} \|\mathbf{x} - \mathbf{x}^*\|_2^2 + \alpha_{\dot{\mathbf{x}}} \|\dot{\mathbf{x}}\|_2^2 + \alpha_\phi \|\phi - \phi^*\|_2^2 + \alpha_{\dot{\phi}} \|\dot{\phi}\|_2^2)$$

pri čemu vektori  $\mathbf{x}$  i  $\phi$  označavaju vektore pozicije i uglova rotacije helikoptera pri čemu zvezdica označava željene vrednosti. Koeficijenti  $\alpha$  su emprijski izabrani tako da svi izrazi budu istog reda veličine. Ova nagrada je utoliko veća što je helikopter bliže željenoj tački i orientaciji i što je mirniji.

Ishod učenja prikazan je na slici 13.7.

---

## Literatura

---

- [1] D. Gabbay, C. J. Hogger, A. Robinson: *Handbook of Logic in Artificial Intelligence and Logic Programming*, Clarendon Press, 1998.
- [2] A. Biere, M. Heule, H. Van Maaren, T. Walsh: *Handbook of Satisfiability*, IOS Press, 2009.
- [3] B. Goertzel, N. Geisweiller, L. Coelho, P. Janičić, C. Pennachin: *Real-World Reasoning: Toward Scalable, Uncertain Spatiotemporal, Contextual and Causal Inference*, Atlantis Thinking Machines, 2011.
- [4] P. Janičić: *Matematička logika u računarstvu* (peto izdanje), Matematički fakultet, 2009.
- [5] M. Nikolić: *Mašinsko učenje*, Matematički fakultet, 2020.
- [6] A. Robinson, A. Voronkov (eds): *Handbook of Automated Reasoning*, Elsevier, 2001.
- [7] S. Russell, P. Norvig: *Artificial Intelligence: A Modern Approach* (Fourth edition), Pearson, 2020.