UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Nikola Belaković

ALGORITMI ZA REŠAVANJE PROBLEMA HORNOVOG JEZGRA

master rad

Beograd, 2024.

Mentor:

dr Aleksandar KARTELJ, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Predrag JANIČIĆ, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Vladimir FILIPOVIĆ, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Hvala profesoru Aleksandru Kartelju.

Naslov master rada: Algoritmi za rešavanje problema Hornovog jezgra

Rezime: Ovaj rad istražuje primenu metaheurističkih tehnika za rešavanje problema pronalaska maksimalnog Hornovog jezgra, sa posebnim fokusom na metodu promenljivih okolina (*eng.* Variable Neighborhood Search) i pohlepnu pretragu (*eng.* Greedy search). Problem pronalaska maksimalnog Hornovog jezgra je složen optimizacioni problem koji zahteva efikasne metode za pretragu velikog prostora rešenja. U okviru ovog istraživanja, analizirane su performanse pomenutih metaheuristika kroz eksperimente izvedene na različitim test podacima, uz poređenje sa metodom grube sile. U radu je detaljno opisan problem i konstrukcija pomenutih algoritama i heuristika. Opisan je način na koji su generisane test instance problema i dat je prikaz rezultata na različitim instancama. Pored toga, rad se bavi razvojem i evaluacijom algoritma za generisanje Hornove formule iz skupa modela. Takođe, razmotrene su ideje za buduće usavršavanje predstavljenih algoritama, kao i mogućnosti za razvoj novih metoda.

Ključne reči: optimizacija, Hornovo jezgro, metaheurističke tehnike, metoda promenljivih okolina, pohlepna pretraga, algoritam generisanja formule

Sadržaj

Glava 1

Uvod

Problem pronalaska maksimalnog Hornovog jezgra (eng. maximum Horn core problem) spada u manje poznate NP-teške probleme optimizacije u domenu logike. Ovaj problem se može opisati kao pronalaženje najvećeg podskupa modela M' iz skupa M, takvog da postoji Hornova formula čiji je skup modela jednak M' [1].

Hornove formule imaju široku primenu u oblastima logike [15, 16], računarstva i veštačke inteligencije, posebno u automatskom zaključivanju [3, 2]. Zahvaljujući ovoj širokoj primeni, sve veći broj istraživača se bavi problemima povezanim sa Hornovim formulama. Jedan od tih problema, problem pronalaska maksimalnog Hornovog jezgra, obrađen je u ovom radu i za njega je dokazano da je NP-kompletan [11].

Jedan od algoritama za rešavanje problema maksimalnog Hornovog jezgra predstavljen je u radu [6]. Ovaj algoritam, čiji će detaljan opis biti predstavljen u Glavi 2, korišćen je u ovom istraživanju. Testirana je optimizacija korišćenjem pohlepnog algoritma (*eng.* greedy search) i metode promenljivih okolina (*eng.* variable neighborhood search).

U nastavku ovog uvodnog poglavlja, formalno će biti definisan problem pronalaska maksimalnog Hornovog jezgra i prikazan pregled dosadašnjih istraživanja na ovu temu.

1.1 Problem pronalaska maksimalnog Hornovog jezgra

U ovom poglavlju formalno će biti definisan problem pronalaska maksimalnog Hornovog jezgra, ali pre toga je potrebno definisati i neke bitne pojmove za razumevanje problema.

Literal je osnovna jedinica u iskaznoj logici koja predstavlja atomičku formulu (poznatu i kao atom ili osnovna formula) ili njenu negaciju [13]. Klauza je disjunkcija jednog ili više literala. Hornova klauza je klauza koja sadrži najviše jedan pozitivan literal [9]. Konjunktivna normalna forma (CNF) je logički izraz koji se sastoji od konjunkcije jedne ili više disjunkcija literala. Hornova formula je specifična vrsta formule u CNF-u koja se sastoji isključivo od Hornovih klauza [11].

Model t je skup $\{0,1\}^n$ istinitosnih valuacija logičkih promenljivih. Model t zadovoljava klauzu ako je neki literal klauze x_i i $t_i = 1$ ili je neki literal klauze $\neg x_i$ i $t_i = 0$ – t_i predstavlja vrednost i-te iskazne promenljive u valuaciji t. Neka je dat skup M modela nad n iskaznih promenljivih. Hornovo jezgro skupa M je podskup M' za koji postoji Hornova formula takva da je M' skup svih njenih modela – svaki član skupa zadovoljava Hornovu formulu prema prethodnom opisu zadovoljivosti, dok nijedan model izvan skupa M' ne zadovoljava tu formulu. Funkcija cilja za problem pronalaska maksimalnog Hornovog jezgra je kardinalnost Hornovog jezgra i nju je potrebno maksimizovati [11, 1].

Važno je napomenuti da maksimalno Hornovo jezgro, jezgro maksimalne kardinalnosti, nije jedinstveno. Jedan skup modela M može imati više različitih Hornovih jezgara, svaki sa istim brojem istinitosnih valuacija koje zadovoljavaju neku Hornovu formulu [11].

Primer: Razmotrimo sledeći skup modela:

$$M = \{(0,0,0,1), (0,1,0,1), (1,0,1,1), (0,1,1,1)\}$$

Da bismo pronašli maksimalno Hornovo jezgro, koristimo algoritam koji dodaje valuacije u jezgro i proverava da li njihova „konjunkcija", primena logičke operacije $\wedge$ na svaku poziciju u valuaciji, pripada skupu M. Počinjemo sa valuacijom $(0,0,0,1)$ i dodajemo je u jezgro. Zatim dodajemo valuaciju $(0,1,0,1)$, jer je njena „konjunkcija" sa prvom valuacijom u M. Sledeća valuacija, $(1,0,1,1)$, takođe se dodaje jer njene „konjunkcije" sa prethodnim valuacijama pripadaju skupu M. Međutim, valuacija $(0,1,1,1)$ se ne dodaje, jer „konjunkcija" sa poslednjom dodanom valuacijom

$(1, 0, 1, 1)$ daje:

$$(1, 0, 1, 1) \wedge (0, 1, 1, 1) = (0, 0, 1, 1)$$

koja nije prisutna u skupu M. Dakle, maksimalno Hornovo jezgro je:

$$\mathrm{M}_{\mathrm{core}} = \{(0, 0, 0, 1), (0, 1, 0, 1), (1, 0, 1, 1)\}$$

Dato jezgro je skup modela formule:

$$\text{Formula} = (\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge x_4) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3 \wedge x_4) \vee (x_1 \wedge \neg x_2 \wedge x_3 \wedge x_4)$$

Ova formula je u disjunktivnoj normalnoj formi, a odgovarajuća formula u konjunktivnoj normalnoj formi je:

$$\begin{aligned}\text{Formula} = {} & (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4) \\ & \wedge (x_1 \vee \neg x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4) \\ & \wedge (\neg x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_3 \vee x_4) \\ & \wedge (\neg x_1 \vee \neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \\ & \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4)\end{aligned}$$

Hornova formula koja je ekvivalentna ovim formulama i koja se može dobiti iz obe gore prikazane formule je:

$$\text{Formula} = (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge x_4 \wedge (x_1 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$$

Modeli u maksimalnom Hornovom jezgru zadovoljavaju ovu formulu, što objašnjava zašto su ovi modeli deo Hornovog jezgra.

Za varijantu problema odlučivanja Hornovog jezgra je dokazano da je NP-kompletna u radu [11].

1.2 Pregled dosadašnjih istraživanja

Pojam Hornove formule prvi put je uveden od strane Alfreda Horna u njegovom radu „On sentences which are true of direct unions of algebras," koji je objavljen 1951. godine u časopisu *Journal of Symbolic Logic* [9]. U ovom radu, Horn je proučavao posebnu klasu logičkih formula koje danas nazivamo Hornovim formulama, koje imaju specifičnu strukturu pogodnu za efikasno zaključivanje.

Razvoj pojma Hornovog jezgra započeo je 1991. godine kada je uveden pojam „najveća donja granica" (*eng.* greatest lower bound) i dat prvi algoritam za rešavanje ovog problema pomoću leksikografskog sortiranja istinitosnih valuacija iz skupa modela [16]. Ovaj pojam je uveden kao rezultat potrebe za efikasnijim sistemima za predstavljanje znanja koji ne ograničavaju izražajnu moć jezika za predstavljanje, niti se odriču potpunosti zaključivanja. Raniji radovi, kao što su oni Doylea i Patila (1991), te Horvitza (1989), bavili su se kompromisom između obradivosti i izražajnosti koristeći ograničene jezike ili nepotpune mehanizme zaključivanja [5, 10]. Za razliku od tih pristupa, uvođenje Hornovog jezgra omogućava korišćenje opšteg, neograničenog jezika koji se kompajlira u ograničeni jezik, čime se postiže efikasno zaključivanje bez gubitka izražajne moći. Ovaj novi pristup omogućava pronalaženje najbolje aproksimacije originalne informacije, što dovodi do bržeg i pouzdanijeg zaključivanja.

Godine 1993., pojam „Hornovo jezgro" se prvi put spominje sa tim imenom [11]. Ovaj rad je bio posvećen dokazivanju računske složenosti problema, a ne njegovom rešavanju.

Naredni razvoj dogodio se 1998. godine kada je rešen modifikovani problem pronalaska maksimalnog Hornovog jezgra. U ovom slučaju, tražena su Hornova jezgra disjunkcije Hornovih formula u konjunktivnoj normalnoj formi [6].

Pošto se metoda promenljivih okolina pokazao kao uspešna metaheuristika za rešavanje mnogih problema, u ovom istraživanju pristupilo se rešavanju problema korišćenjem ove metaheuristike i njenom poređenju sa algoritmom pohlepne pretrage. Ovaj pristup omogućava istraživanje različitih struktura rešenja i često daje bolje rezultate u poređenju sa tradicionalnim metodama kao što je pohlepna pretraga.

Glava 2

Algoritmi za rešavanje problema pronalaska maksimalnog Hornovog jezgra

U ovom poglavlju biće dat pregled sledeće dve metaheuristike koje su korišćene za rešavanje problema pronalaska maksimalnog Hornovog jezgra:

1. pohlepna pretraga (*eng.* greedy search);
2. metoda promenljivih okolina (*eng.* variable neighborhood search).

Osim ove dve metaheuristike, biće opisan i algoritam polinomske vremenske složenosti generisanja jednog Hornovog jezgra, koji je pomenut u prethodnom poglavlju, kao i algoritam za generisanje Hornove formule iz skupa modela koje ta formula zadovoljava.

2.1 Algoritam pronalaska Hornovog jezgra

Model je vektor $t \in \{0,1\}^n$, čija je i-ta komponenta označena sa t_i. Teorija je bilo koji skup $M \subseteq \{0,1\}^n$ modela. Nad modelima t_j i t_k se definiše bitovsko uređenje tako da za svaku komponentu i vektora važi $t_j[i] \leq t_k[i]$, gde $0 \leq 1$ označava da je 0 manja ili jednaka od 1 u binarnom poređenju. Dakle, vektor t_j je manji ili jednak vektoru t_k ako nijedna komponenta vektora t_j nije veća od odgovarajuće komponente vektora t_k.

Teorija je Hornova ako je $M = Cl_\wedge(M)$, gde je $Cl_\wedge(M)$ zatvorenje skupa $M \subseteq \{0,1\}^n$ za operaciju bitovske konjunkcije (tj. preseka) valuacija t_i i t_j, označeno sa $t_i \wedge t_j$.

Hornova teorija M' je Hornovo jezgro teorije M ako je $M' \subseteq M$ i ne postoji Hornova teorija M'' takva da je $M' \subseteq M'' \subseteq M$. Primećujemo da, uopšteno, teorija M može imati više od jednog Hornovog jezgra [6].

Iz ovog opisa Hornovog jezgra se može jednostavno formirati algoritam za pronalaženje jednog Hornovog jezgra teorije M. Na početku se Hornovo jezgro M' inicijalizuje praznim skupom. Zatim se prolazi kroz sve komponente modela iz M koje se prethodno mogu sortirati prema bitovskom uređenju i dodaju se u M'. Nakon toga se proverava da li je trenutni skup M' zatvoren za konjunkciju, tako što se za svaki par $v, w \in M'$ proverava da li je $v \wedge w \in M'$. Ako jeste, dodata komponenta se zadržava u M', a u suprotnom se izbacuje. Opisanim postupkom dobijamo jedno Hornovo jezgro početnog skupa modela M.

Algoritam 1 Pronalazak Hornovog jezgra

Ulaz: Skup modela M
Izlaz: Hornovo jezgro M'

```
1:  M' ← ∅
2:  M' ← M' ∪ {m_1}
3:  for i = 2 to n do
4:     M' ← M' ∪ {m_i}
5:     zadrzi_komponentu ← True
6:     for v, w ∈ M' do
7:        if v ∧ w ∉ M' then
8:           zadrzi_komponentu ← False
9:           break
10:       end if
11:    end for
12:    if ¬zadrzi_komponentu then
13:       M' ← M' \ {m_i}
14:    end if
15: end for
16: return M'
```

2.2 Pohlepna pretraga

Algoritmi pohlepne pretrage donose odluke u svakom koraku na osnovu trenutno najboljeg izbora, bez obzira na posledice tih odluka u budućnosti. Zbog svoje jednostavnosti i brzine, pohlepna pretraga se često koristi u različitim oblastima, ali nije uvek garantovano da će pronaći globalno optimalno rešenje.

Pohlepna pretraga ima nekoliko prednosti i nedostataka koje je važno razmotriti prilikom njenog korišćenja. Jedna od glavnih predosti je brzina, zato što ovaj algoritam donosi odluke brzo, bez potrebe za pretraživanjem svih potencijalnih opcija. Takođe, ovaj algoritam je jednostavan za implementaciju i postoje određeni problemi u kojima će algoritam pohlepne pretrage pronaći globalno optimalno rešenje. Međutim, pohlepna pretraga ima i značajne nedostatke. Problem je što se može zaglaviti u lokalnom optimumu, i na taj način će pronaći dobro, ali ne i najbolje moguće rešenje. Zbog toga, upotreba ovog algoritma nije predviđena za probleme u kojima je ključno pronaći globalno optimalno rešenje [4].

Algoritam 2 Pohlepna pretraga

Ulaz: Početno rešenje x, maksimalan broj iteracija $maxIter$
Izlaz: Najbolje rešenje x^*

```
1: x* ← x
2: iter ← 0
3: while iter < maxIter do
4:     x' ← GenerišiSusednoRešenje(x*)
5:     if f(x') < f(x*) then
6:         x* ← x'
7:     end if
8:     iter ← iter + 1
9: end while
10: return x*
```

2.3 Metoda promenljivih okolina

Metoda promenljivih okolina (VNS) je metaheuristika za rešavanje problema optimizacije [12]. VNS se bazira na tri jednostavne činjenice:

- lokalni minimum za jednu okolinu ne mora biti minimum u odnosu na drugu okolinu;

- globalni minimum je lokalni minimum za sve okoline;
- za mnoge probleme, lokalni minimumi za razne okoline su relativno bliski.

Ova metoda se zasniva na ideji smena faza sistematičnih i nasumičnih (stohastičkih) promena okolina tokom pretrage prostora rešenja, čime se izbegava lokalni optimum i povećava šansa za pronalaženje globalnog optimuma. Njeni osnovni principi su:

- **definisanje početnog rešenja** - pretraga počinje od nekog početnog rešenja, koje može biti nasumično generisano ili dobijeno nekom heuristikom;
- **razmrdavanje (*eng.* shaking)** - promena trenutnog rešenja slučajnim izborom rešenja iz trenutne okoline, koja ima za cilj izlazak iz lokalnog optimuma i pomeranje u novo, potencijalno bolje rešenje;
- **popravljanje** - lokalnom pretragom se od izabranog rešenja iz faze razmrdavanja dolazi do lokalnog optimuma u njegovoj okolini;
- **pomeranje** - ako lokalna pretraga pronađe bolje rešenje od trenutno najboljeg rešenja, pretraga se nastavlja oko tog rešenja. U suprotnom, menja se okolina za razmrdavanje kako bi se istražila nova okruženja potencijalnih rešenja;
- **zaustavljanje** - pretraga se završava kada je ispunjen određeni kriterijum zaustavljanja, kao što je vreme izvršavanja algoritma ili maksimalan broj iteracija [7].

Metoda VNS ima mnoge prednosti koje je čine efikasnim alatom za rešavanje problema optimizacije, što je čini popularnim izborom među istraživačima. Njena jednostavna implementacija i fleksibilnost omogućavaju primenu na širok spektar problema. Jedan od ključnih prednosti je sposobnost izbegavanja lokalnih optimuma. Faza razmrdavanja i promena okolina pretrage pružaju mogućnost istraživanja većeg dela prostora rešenja, povećavajući šanse za pronalaženje globalnog optimuma. VNS je skalabilan i može se prilagoditi za rad sa velikim instancama problema, čemu doprinosi i njegova efikasnost za pronalaženje kvalitetnog rešenja u kratkom vremenskom okviru [8].

Algoritam 3 Metoda promenljivih okolina

Ulaz: Početno rešenje x, skup okolina $\{N_1, N_2, \ldots, N_k\}$, maksimalan broj iteracija $maxIter$

Izlaz: Najbolje rešenje x^*

```
1: x* ← x
2: iter ← 0
3: while iter < maxIter do
4:    x' ← LokalnaPopravka(x*)
5:    k ← 1
6:    while k ≤ k_max do
7:       x'' ← Razmrdavanje(x', N_k)
8:       x''' ← LokalnaPretraga(x'')
9:       if f(x''') < f(x*) then
10:         x* ← x'''
11:         k ← 1
12:      else
13:         k ← k + 1
14:      end if
15:   end while
16:   iter ← iter + 1
17: end while
18: return x*
```

2.4 Algoritam generisanja Hornove formule na osnovu skupa modela

Ovaj algoritam generiše Hornovu formulu na osnovu skupa modela (istinitosne tablice). Proces je sistematičan i podeljen u nekoliko ključnih koraka. Cilj je da se identifikuju zajednički elementi modela, formiraju kombinacije koje zadovoljavaju sve modele i na kraju da se formira i pojednostavi formula do Hornove forme.

Prvo, algoritam identifikuje zajedničke jedinice u svim modelima. Ovo su pozicije gde svaki model ima vrednost 1 (tačno).

Algoritam 4 Prepoznavanje zajedničkih jedinica

Ulaz: Skup modela M
Izlaz: Indeksi zajedničkih jedinica I

```
1: I ← ∅
2: for i = 1 to |M| − 1 do
3:    sve_jedinice ← True
4:    for m ∈ M do
5:       if m_i ≠ 1 then
6:          sve_jedinice ← False
7:          break
8:       end if
9:    end for
10:   if sve_jedinice then
11:      I ← I ∪ {i}
12:   end if
13: end for
14: return I
```

Primer: Razmotrimo Hornovo jezgro

$$M = \{(0, 0, 0, 1), (0, 1, 0, 1), (1, 0, 1, 1)\}.$$

Pretražuju se sve pozicije u modelima da bi se pronašle zajedničke jedinice. U ovom slučaju, samo je pozicija 3 (indeksiranje od 0) zajednička za sve modele. Dakle, $I = \{3\}$.

Identifikacija zajedničkih jedinica osigurava da su ove pozicije prisutne u svim modelima. Na osnovu tih zajedničkih jedinica formira se klauza koja mora biti zadovoljena u svakom modelu u skupu. U ovom slučaju, formira se klauza x_4. Ovaj korak garantuje da formula ostaje tačna za sve modele u skupu.

Nakon što su zajedničke jedinice identifikovane, sledeći korak je skraćivanje modela uklanjanjem tih jedinica. Ovo omogućava efikasniju obradu u kasnijim koracima.

Algoritam 5 Skraćivanje modela

Ulaz: Skup modela M, skup zajedničke jedinica I
Izlaz: Skraćeni modeli M'

```
M' ← ∅
for m ∈ M do
    m' ← ""
    for i = 1 to |m| − 1 do
        if i ∉ J then
            m'_i ← m_i
        end if
    end for
    M' ← M' ∪ {(m, m')}
end for
return M'
```

Primer: Koristeći skup zajedničkih jedinica $I = \{3\}$, skraćuju se modeli i pravi novi skup modela:

$$M' = \{((0,0,0,1),'000'), ((0,1,0,1),'010'), ((1,0,1,1),'101')\}.$$

Skraćivanje modela uklanjanjem zajedničkih jedinica omogućava fokusiranje na preostale promenljive, čime se poboljšava efikasnost obrade u sledećim koracima. Ovaj korak pomaže u smanjenju složenosti problema. Kada se zajedničke jedinice uklone iz modela, preostaje manji broj promenljivih i njihovih vrednosti koje treba obraditi. Ovo smanjenje složenosti pojednostavljuje dalje analize i generisanje klauza, jer se sada razmatraju samo promenljive koje nisu jedinice u svim modelima. Time se značajno smanjuje broj varijacija koje treba razmotriti, što poboljšava efikasnost celokupnog algoritma i omogućava brže i preciznije rezultate.

Generisanje svih mogućih binarnih varijacija sa ponavljanjem predstavlja sledeći korak. Ove varijacije su osnova za stvaranje formule koja zadovoljava potpun skup modela.

Algoritam 6 Generisanje formule

Ulaz: Broj promenljivih n, skup zajedničkih jedinica J, skup skraćenih model S, skup svih proširenih modela E

Izlaz: CNF formula F

```
1: F ← ""
2: for (e, c) ∈ E do
3:    if c ∉ S then
4:       C ← [ ]
5:       for j = 0 to |e| − 1 do
6:          if j ∉ J then
7:             if e_j = 1 then
8:                C.append("¬X_(j+1)")
9:             else
10:               C.append("X_(j+1)")
11:            end if
12:         end if
13:      end for
14:      F ← F + " ∨ " + C
15:   end if
16: end for
17: return F
```

Primer: Na osnovu skraćenih modela {('000'), ('010'), ('101')} i skupa zajedničkih jedinica, generišemo sve moguće klauze. Prvo, kreiraju se sve moguće binarne varijacije za preostale promenljive, za promenljive koje nisu zajedničke svim modelima. U ovom slučaju, ostaju promenljive x_1, x_2 i x_3, a njihove varijacije su '000', '001', '010', '011', '100', '101', '110', i '111'. Zatim, za svaku varijaciju koja nije prisutna među skraćenim modelima, formira se klauza koja predstavlja negaciju te varijacije. Na primer, varijacija '001' generiše klauzu $(x_1 \vee x_2 \vee \neg x_3)$. Konačno, klauza za zajedničku promenljivu x_4 se dodaje formuli, jer ona mora biti prisutna u svim modelima. Rezultat će biti:

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

Generisanje klauza na osnovu binarnih varijacija omogućava da se obuhvate sve moguće varijacije vrednosti promenljivih koje nisu prisutne u modelima. Kada se identifikuju sve binarne varijacije koje nisu zastupljene u modelima, za svaku od njih se formira odgovarajuća klauza. Kreiranjem klauza za te varijacije isključuju se vrednosti koje ne zadovoljavaju modele. Ovaj proces osigurava da formula ostane tačna za sve modele iz skupa i nijedan model van njega.

Kao završni korak, koristi se metoda rezolucije za pojednostavljenje formule. Rezolucija je ključna tehnika u logici koja pomaže u rešavanju problema sa logičkim izrazima tako što identifikuje i kombinuje klauze koje se mogu rešiti da bi se smanjio broj klauza i pojednostavila formula. Cilj rezolucije je da se eliminišu konfliktne i redundantne klauze, čime se formula pojednostavljuje i postaje efikasnija za dalje analize.

Definicija. Neka su C_1 i C_2 klauze, i neka su L_1 i L_2 komplementarni literali, takvi da se L_1 nalazi u C_1, a L_2 u C_2. Rezolventa klauza C_1 i C_2 po literalima L_1 i L_2 je klauza dobijena kao:

$$R(C_1, C_2, L_1, L_2) = (C_1 \setminus \{L_1\}) \cup (C_2 \setminus \{L_2\})$$

Gde R predstavlja rezolventu klauzu.

Teorema 1. *Neka su C_1 i C_2 klauze i R njihova rezolventa. Tada je:*

$$\{C_1, C_2\} \equiv \{C_1, C_2, R\}.$$

Dokaz. Neka je I interpretacija za koju važi da je $I \models \{C_1, C_2, R\}$. Tada je trivijalno da važi $I \models \{C_1, C_2\}$. Obrnuto, pretpostavimo da je I interpretacija za koju važi $I \models \{C_1, C_2\}$. Neka su L i $\neg L$ literali po kojima se vrši rezolucija klauza C_1 i C_2, takvi da je $L \in C_1$ i $\neg L \in C_2$. Ako $I \models C_1$, onda mora važiti $I \models C_2 \setminus \{\neg L\}$, pa sledi da $I \models R$. Slično, ako $I \models C_2$, tada $I \models C_1 \setminus \{L\}$, pa $I \models R$.

□

Ova teorema pokazuje da dodavanje rezolventne klauze R u skup klauza $\{C_1, C_2\}$ ne menja skup modela koji zadovoljava te klauze. Drugim rečima, klauze C_1 i C_2 su ekvivalentne skupu klauza koji uključuje i njihovu rezolventu R. Ova osobina rezolucije omogućava sistematsko pojednostavljenje klauza bez promene njihovih osnovnih logičkih svojstava.

U procesu redukcije, rezolucija se koristi za identifikaciju i eliminaciju neslaganja između klauza. Ako se pronađe par klauza koje sadrže komplementarne literale, one se mogu kombinovati da bi se formirala nova klauza koja ne uključuje te komplementarne literale. Ova nova klauza se zatim dodaje u skup klauza, a proces se ponavlja sve dok se ne postigne željeni oblik formule, kao što je Hornova forma, ili dok se ne ispune drugi kriterijumi za jednostavnost formule.

Algoritam 7 Redukcija formule

Ulaz: CNF formula F
Izlaz: Reducirana Hornova formula F'

$F' \leftarrow$ ""
$klauze \leftarrow$ split(formula, &)
while $PostojiNeHornovaKlauza(klauze)$ **do**
  $rezolvirane\_klauze \leftarrow \emptyset$
  $rezolvirana\_klauza \leftarrow$ None
  $rezolviran \leftarrow$ False
  for $k1, k2 \in klauze$ **do**
    $rezolvirana\_klauza, potrebne\_izmene \leftarrow PrimeniRezoluciju(k1, k2)$
    if $rezolvirana\_klauza \neq$ None **and** $potrebne\_izmene$ **then**
      $rezolvirane\_klauze \leftarrow rezolvirane\_klauze \cup \{rezolvirana\_klauza\}$
    end if
    $rezolviran \leftarrow$ True
    break
  end for
  if $\neg rezolviran$ **then**
    $klauze \leftarrow originalne\_klauze$
    $klauze.shuffle()$
  end if
  if $potrebne\_izmene$ **then**
    $klauze \leftarrow klauze \setminus \{k1\}$
    $klauze \leftarrow klauze \setminus \{k2\}$
  end if
  for $klauza \in rezolvirane\_klauze$ **do**
    $klauze \leftarrow klauze \cup \{klauza\}$
  end for
end while
for $klauza \in klauze$ **do**
  $F' \leftarrow F' +$ " $\wedge$ " $+ klauza$
end for

Primer: Primenom algoritma za redukciju formule na formulu dobijenu u prethodnom primeru, kao rezultat dobijemo:

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

Ovaj algoritam pruža efikasan način da se generiše Hornova formula koja zadovoljava sve specifikacije iz skupa modela. Ključni koraci uključuju prepoznavanje zajedničkih jedinica, skraćivanje modela, generisanje varijacija, formiranje CNF formule, i redukciju klauza do Hornove forme. Proces omogućava strukturirano i si-

stematično pristupanje problemu, čineći ga pogodnim za različite primene u oblasti logike i računarstva [14].

Analiza vremenske složenosti algoritma za generisanje Hornove formule ključna je za razumevanje njegovih performansi, posebno kada se koristi za velike skupove modela. Iako je algoritam testiran na malim skupovima modela i pokazao zadovoljavajuće performanse, važno je sagledati kako se njegovo vreme izvršavanja ponaša sa povećanjem broja modela.

Algoritam se sastoji iz nekoliko glavnih faza, čija vremenska složenost se može analizirati na sledeći način:

- **prepoznavanje zajedničkih jedinica** - ova faza uključuje iteraciju kroz sve modele i proveru da li su svi elementi na određenim pozicijama jednaki. Ako je broj modela m i dužina modela n, tada je vremenska složenost ove faze $O(m \cdot n)$, jer se za svaku poziciju u modelima proverava svaki model;

- **skraćivanje modela** - u ovoj fazi, za svaki model, uklanjaju se pozicije koje sadrže zajedničke jedinice. Ako je broj modela m i dužina svakog modela n, skraćivanje se može smatrati kao $O(m \cdot n)$, jer se za svaki model prolazi kroz sve pozicije i uklanjaju se zajedničke jedinice.

- **generisanje binarnih varijacija** - generisanje svih mogućih binarnih varijacija za n promenljivih dovodi do 2^n varijacija. Svaka varijacija se obrađuje i dodaje u skup klauza, što može biti eksponencijalno u odnosu na broj promenljivih. Dakle, vremenska složenost ove faze je $O(2^n)$;

- **formulisanje klauza i rezolucija** - kada se generišu sve klauze, rezolucija se koristi za eliminaciju konflikata između klauza. Rezolucija između dva skupa klauza može dovesti do dodatnih klauza, a broj klauza može eksponencijalno rasti u odnosu na broj promenljivih. U najgorem slučaju, složenost može biti $O(2^n)$ zbog potrebe za obradom svih mogućih klauza i njihovih kombinacija.

Na osnovu ove analize, jasno je da algoritam može postati veoma spor za velike skupove modela zbog eksponencijalne složenosti u vezi sa brojem promenljivih i brojem generisanih klauza. Iako je za male skupove modela performansa zadovoljavajuća, optimizacija ili korišćenje efikasnijih tehnika može biti neophodno za rad sa većim skupovima modela.

Glava 3

Implementacija

U ovom poglavlju biće prikazani ključni aspekti implementacije algoritama, kao i testni podaci koji su korišćeni za evaluaciju, uključujući i metodologiju njihovog generisanja. Takođe, biće pružen i kratak pregled eksperimentalnog okruženja u kojem je izvršeno testiranje, uključujući hardverske i softverske specifikacije računarskog sistema, kao i specifičnosti koje su uticale na implementaciju i izvršenje algoritma.

3.1 Test podaci i njihovo generisanje

Za evaluaciju performansi razvijenih algoritama, korišćeni su specijalno generisani testni podaci. Testni skupovi su predstavljali različite konfiguracije modela kako bi se obuhvatila različita scenarija i proverila efikasnost algoritama u različitim uslovima. Ukupno je generisano 17 testnih skupova podataka: dva mala i 15 velikih. Ova podela omogućila je detaljnu analizu i upoređivanje različitih pristupa, uključujući metaheuristike i metode grube sile.

Testni skupovi su generisani pomoću algoritma koji pravi modele sa binarnim vrednostima. Algoritam pravi nasumične varijacije sa ponavljanjem vrednosti promenljivih za zadati broj promenljivih n i broj modela m, dok broj kreiranih jedinstvenih varijacija ne postane m. Ovi skupovi su zatim zapisani u tekstualne datoteke za dalju analizu i testiranje.

Primer: U nastavku možemo videti izgled tekstualne datoteke 'TestInstances2.txt' (mali skup podataka):

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

Algoritam 8 Generisanje testnih skupova

Ulaz: Broj promenljivih n, broj modela m
Izlaz: Lista generisanih modela L

$L \leftarrow \emptyset$
while $|L| < m$ **do**
    $variation \leftarrow [\text{random}(0, 1) \text{ for } i \text{ in range}(n)]$
    $L \leftarrow L \cup \{variation\}$
end while
return $[L]$

Za male testne skupove korišćeni su modeli sa četiri promenljive. Prvi mali skup uključivao je 4 valuacije u modelu, dok je drugi je imao 10 valuacija. Ovi mali skupovi su korišćeni za testiranje obe metaheuristike kao i algoritma grube sile. Testiranje na ovim skupovima omogućilo je poređenje tačnosti i efikasnosti metaheurističkih metoda u odnosu na klasičnu metodu grube sile.

Osim testiranja metaheuristika i metode grube sile, na ovim malim skupovima testirano je i generisanje Hornove formule iz skupa modela. Iako generisanje Hornove formule nije bio primarni cilj istraživanja, uključivanje ove faze omogućilo je procenu efikasnosti i tačnosti algoritma za generisanje formula. Ova analiza pružila je uvid u dodatne aspekte performansi algoritama, pokazujući kako dobro algoritam može transformisati skup modela u odgovarajuću Hornovu formulu. Nažalost, zbog velike vremenske složenosti ovog algoritma, nije bilo moguće testirati njegov rad na velikim skupovima podataka.

Na velikim testnim skupovima, koji su uključivali modele sa 5, 6, 7, 8, 9 i 10 promenljivih, testirane su isključivo metaheuristike. Ovi veliki skupovi su sadržavali od 2 do 3 različita broja valuacija za svaki broj promenljivih. Testiranje ovih skupova sa metodom grube sile bilo je nepraktično zbog velike vremenske složenosti algoritma, što je rezultiralo predugim vremenima izvršavanja.

Kao ilustraciju, radi boljeg razumevanja testnih podataka, razmotrićemo mali testni skup sa četiri promenljive. U ovom skupu, svaki model je predstavljen kao binarna sekvenca od četiri vrednosti, gde svaka vrednost može biti 0 ili 1. Evo primera skupa:

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

U ovom primeru, svaki red predstavlja jedan model sa četiri promenljive. Svaki model je niz od četiri binarne vrednosti koje odgovaraju stanjima promenljive. Prvi model `0 0 0 1` pokazuje da su prve tri promenljive netačne (0), dok je poslednja promenljiva tačna (1).

Ovi testni skupovi i metodologija generisanja podataka omogućili su detaljnu analizu i upoređivanje performansi različitih algoritamskih pristupa. Proces generisanja podataka, zajedno sa evaluacijom rezultata na različitim skupovima, omogućio je dobijanje uvida u efikasnost i tačnost razvijenih algoritama, kao i identifikaciju mogućih područja za unapređenje.

3.2 Eksperimentalno okruženje

Svi rezultati dobijeni u ovom istraživanju generisani su i analizirani u okruženju koje koristi *Python* programski jezik i *Jupyter notebook* za razvoj i testiranje. Eksperimentalno okruženje je postavljeno na računaru sa sledećim specifikacijama:

- **Operativni sistem:** Ubuntu 22.04.4 LTS
- **Jezgro:** Linux 6.5.0-41-generic

Hardverska specifikacija računara korišćenog za testiranje su sledeće:

- **Arhitektura:** x86-64

- **Procesor:** 12th Gen Intel Core i5-1235U x 12, 4.4GHz
- **RAM memorija:** Kingston 16GB DDR4

Programski kod[1] je organizovan u šest skripti, od kojih se tri koriste za implementaciju algoritama i metaheuristika (metoda grube sile, VNS i pohlepna pertraga). Skripta (*generate_test_instance.ipynb*) omogućava generisanje dodatnih skupova podataka. Program se može testirati i bez pokretanja ove skripte jer postoje već generisani testni podaci. Ostale dve skripte (*test_small.ipynb* i *test_large.ipynb*) izvršavaju algoritme nad definisanim skupovima podataka. Dizajnirane su tako da prvo pokrenu sve potrebne skripte sa algoritmima, a zatim testiraju i prikazuju rezultate testiranja algoritama.

Za pokretanje skripti, potrebno je samo izvršiti odgovarajuće komande unutar Jupyter okruženja, osiguravajući da su sve potrebne biblioteke instalirane. Ove biblioteke omogućavaju lako upravljanje podacima i fleksibilnost u eksperimentalnom procesu. Možete ih instalirati korišćenjem sledeće komande u terminalu:

```
pip install naziv_biblioteke
```

Korišćene su sledeće biblioteke:

- **NumPy:** Koristi se za rad sa nizovima i matematičkim operacijama.
- **Random:** Omogućava generisanje slučajnih brojeva.
- **OS:** Koristi se za interakciju sa operativnim sistemom.
- **RE:** Omogućava rad sa regularnim izrazima za pretragu i manipulaciju tekstom.
- **Itertools**: Koristi se za efikasno generisanje iterativnih funkcija.
- **Time**: Omogućava merenje vremena izvršavanja.
- **Matplotlib:** Omogućava pravljenje grafičkog prikaza i vizualizacije podataka.

[1] `https://github.com/NikolaBelak17/master`

3.3 Implementacija algoritama

U ovom delu biće detaljno opisana implementacija algoritama korišćenih u istraživanju. Fokus će biti na metodi grube sile, metodi promenljive okoline, pohlepnoj pretrazi, kao i algoritmu za generisanje Hornovih formula.

Sva tri navedena algoritma za rešavanje problema optimizacije koriste metodu *is_horn_theory*, koja je ključna za proveru da li trenutni skup valuacija predstavlja Hornovu teoriju. Ova metoda proverava da li je zadata valuacija u početnom skupu modela:

```
1 def is_horn_theory(assignment, M):
2     if tuple(assignment) not in M:
3         return False
4     return True
```

Implementacija metode grube sile koristi jednostavan pristup isprobavanja svih mogućih rešenja. Na početku se *horn_core* postavlja na praznu listu, u koju će biti smešteno trenutno najbolje jezgro. Zatim se prolazi kroz sve moguće varijacije modela iz početnog skupa za različite veličine, od 1 do broj modela u skupu, koristeći funkciju *combinations* iz biblioteke *itertools*. Za svaku varijaciju se proverava da li rešenje „konjunkcije" svih mogućih parova valuacija pripada početnom skupu, korišćenjem metode *is_horn_theory*. Ako je varijacija validna Hornova teorija i duža od do tada najboljeg jezgra, ažurira se *horn_core*. Na kraju, funkcija vraća najbolje pronađeno jezgro.

Implementacija algortima pohlepne pretrage iterativno gradi rešenje, dodajući valuacije koje proširuju trenutni skup, dokle god taj prošireni skup ostaje validna Hornova teorija. Algoritam započinje tako što se inicijalizuje prazan skup *horn_core* koji predstavlja trenutno jezgro i skup *used_assignment* u koji će biti smeštene već korišćene valuacije. Zatim, slučajnim odabirom iz početnog skupa modea, korišćenjem funkcije *choice* iz biblioteke *random*, se bira početna valuacija koja se dodaje u oba skupa.

U glavnoj petlji algoritma, dokle god nisu iskorišćene sve dostupne valuacije, slučajno se bira nova valuacija koja nije korišćena. Ova nova valuacija se privremeno dodaje u *horn_core*, formirajući novi kandidat za jezgro *new_core*:

```
for _ in range(len(M)):
    if len(used_assignment) == len(M):
        break
    while True:
        random_assignment = random.choice(M)
        if random_assignment not in used_assignment:
            break
    used_assignment.add(random_assignment)
    new_core = horn_core | set([random_assignment])
```

Zatim, za svaki par valuacija u tom novom kandidatu se izračunava „konjunkcija". Ovo se postiže korišćenjem funkcija *zip* i *all* koje su deo standardne *Python* biblioteke. Funkcija *zip* iterira kroz oba modela paralelno, spajajući odgovarajuće promenljive, dok *all* proverava da li su sve promenljive istovremeno istinite u oba modela. Rezultat ove operacije je nova valuacija koja se konvertuje u binarni format:

```
for pair in combinations(new_core, 2):
    new_assignment = [all(var_values) for var_values in zip(*pair)]
    new_assignment = [int(val) for val in new_assignment]
```

Ako bilo koje rešenje „konjunkcije" ne zadovoljava uslov da pripada početnom skupu modela, što se proverava korišćenjem funkcije *is_horn_theory*, kandidat za jezgro se odbacuje. Ukoliko sve valuacije zadovolje uslov, novi kandidat postaje novo jezgro *horn_core*.

Tokom svake iteracije, dužina trenutnog jezgra se beleži u listi *cores*, koja prati razvoj jezgra kroz algoritam. Na kraju, algoritam vraća konačno Hornovo jezgro *horn_core* i listu *cores* koja prikazuje rast jezgra tokom pretrage.

Metoda promenjlivih okolina je implementirana u nekoliko ključnih koraka za optimizaciju rešenja. Algoritam započinje inicijalizacijom početnog Hornovog jezgra *horn_core*, koje se inicijalno postavlja na prazan skup, a zatim slučajnim odabirom iz početnog skupa modela bira valuacija koja se dodaje u jezgro. Algoritam se oslanja na dve osnovne funkcije *shaking* i *local_search*.

Funkcija *shaking* se koristi za generisanje izmenjenog jezgra uklanjanjem slučajno izabrane valuacije iz trenutnog jezgra:

```
1 def shaking(current_core, M):
2     new_core = set(current_core)
3     if new_core:
4         assignment_to_remove = random.choice(list(new_core))
5         new_core.remove(assignment_to_remove)
6     return new_core
```

Nakon razmrdavanja, novo jezgro se prosleđuje funkciji *local_search*, koja dodaje nove valuacije i pokušava da pronađe bolje rešenje, pod uslovom da prošireni skup ostaje validna Hornova teorija kao prethodno opisan algoritam pohlepne pretrage.

Tokom svake iteracije, veličina trenutnog jezgra se beleži u listi *cores*, koja prati njegov razvoj. Algoritam koristi parametre *max_iterations* za broj ukupnih iteracija i *k_max* za broj različitih okolina koje se istražuju u svakoj iteraciji. Na kraju, funkcija *variable_neighborhood_search* vraća konačno Hornovo jezgro *horn_core* i listu *cores* koja prikazuje rast jezgra tokom pretrage.

Implementacija algoritma za generisanje Hornove formule počinje identifikacijom zajedničkih jedinica u datom skupu modela. Funkcija *find_common_ones* prolazi kroz sve modele i identifikuje promenljive koje su jednake 1 u svim modelima. Zatim se koristi funkcija *shorten_models* kako bi se ovi modeli skratili uklanjanjem zajedničkih jedinica. Ova funkcija koristi metodu *join* iz standardne *Python* biblioteke, koja spaja elemente liste u jedan string.

Nakon toga, funkcija *generate_all_combinations* generiše sve moguće varijacije binarnih vrednosti za promenljive koje nisu zajedničke jedinice. Svaka od ovih kombinacija se proširuje kako bi se ponovo uključile zajedničke jedinice.

Sledeći korak je generisanje formule u konjuktivnoj normalnoj formi. Funkcija *generate_formula* prolazi kroz sve moguće varijacije i generiše CNF klauze za one varijacije koje nisu prisutne u skraćenim modelima. Svaka klauza se sastoji od literala koji su suprotne vrednosti odgovarajućih promenljivih. Kada je vrednost promenljive 1, koristi se negativni literal, a kada je vrednost 0, koristi se pozitvni literal. Na kraju se formula dopunjuje dodavanjem literala koji odgovaraju zajedničkim jedinicama.

```
formula=""
    for i in range(len(all_models)):
        if all_models[i][1] not in [model[1] for model in
   shortened_models]:
            cnf_clauses = []
            for j, bit  in enumerate(all_models[i][0]):
                if j not in common_ones:
                    if bit == "1":
                        cnf_clauses.append(f"~x{j+1}")
                    else:
                        cnf_clauses.append(f"x{j+1}")

            formula += "(" + " | ".join(cnf_clauses) + ") & "
    for i in common_ones:
        formula += f"x{i+1} & "
    formula = formula[:-3]
```

Da bi se osigurala ispravnost Hornove formule, funkcija $reduce\_formula$ koristi rezoluciju klauza. Ova funkcija iterativno proverava da li postoji neka klauza sa više od jednog pozitivnog literala i pokušava da je zameni sa novom, pojednostavljenom klauzom koristeći rezoluciju između parova klauza pomoću funkcije $resolve\_clause$. Ako se uspešno izvrši rezolucija, originalne klauze se zamenjuju novom, dok se formula ne sastoji isključivo od Hornovih klauza. Konačan rezultat je Hornova formula u CNF obliku.

Iako procesor ima 12 jezgara, algoritmi nisu implementirani za paralelno izvršavanje, što može uticati na brzinu obrade. Implementacija svih struktura podataka poput skupova, listi i uređenih parova, koristi standardne biblioteke *Python* jezika, kao što su *set*, *list* i *tuple*. Pomoćni algoritmi za mešanje i izbor elementa iz skupa su realizovani korišćenjem ugrađenih funkcija poput $shuffle$ i *choice*. Generisanje test instanci koristi ugrađenu biblioteku *random* i ti podaci su smešteni u tekstualnim datotekama, što omogućava lakšu organizaciju i ponovnu upotrebu i analizu tih podataka.

Glava 4

Evaluacija algoritama

U ovom poglavlju biće prikazani eksperimentalni rezultati izvršavanja algoritama. Prvo će biti analizirani rezultati za male testne instance, a zatim za velike testne instance.

Mali testni skupovi omogućavaju detaljno ispitivanje performansi algoritama u kontrolisanim uslovima. Ovi testovi su korisni za proveru tačnosti i pouzdanosti implementacije algoritama na jednostavnim i manje zahtevnim primerima.

Analiza malih testnih primera treba da nam pokaže da li algoritmi funkcionišu kako je očekivano i da li su rezultati u skladu sa onima dobijenim putem metode grube sile. Ova konzistentnost potvrđuje tačnost i pouzdanost implementacija korišćenih algoritama. Male instance omogućavaju detaljnu proveru ispravnosti svakog koraka u algoritmu, osiguravajući da implementacija funkcioniše kako je predviđeno.

Na slikama 4.1 i 4.2 prikazani su rezultati rada algoritma grube sile, pohlepne pretrage i VNS algoritma na malim test primerima. Rezultati uključuju detaljne informacije o ulaznim podacima (veličinu ulaznog skupa modela i njegove članove), broj valuacija u maksimalnom Hornovom jezgru, članove maksimalnog Hornovog jezgra, kao i formule generisane na osnovu Hornovog jezgra (CNF formula i Hornova formula dobijena redukcijom CNF formule). Svaka slika daje uvid u performanse korišćenih metoda i omogućava analizu njihove tačnosti.

```
Number of assignments: 4
M: [(0, 0, 0, 1), (0, 1, 0, 0), (1, 0, 0, 1), (0, 1, 1, 1)]

Brute force results:
Number of assignments in Horn core: 3
Maximal Horn Core: [(0, 0, 0, 1), (1, 0, 0, 1), (0, 1, 1, 1)]
Formula: (x1 | x2 | ~x3) & (x1 | ~x2 | x3) & (~x1 | x2 | ~x3) & (~x1 | ~x2 | x3) & (~x1 | ~x2 | ~x3) & x4
Horn formula:  (~x1 | ~x2 | ~x3)  &  x4 & (x2 | ~x3) & (x3 | ~x2)

Greedy search results:
Number of assignments in Horn core: 3
Maximal Horn Core: [(0, 0, 0, 1), (1, 0, 0, 1), (0, 1, 1, 1)]
Formula: (x1 | x2 | ~x3) & (x1 | ~x2 | x3) & (~x1 | x2 | ~x3) & (~x1 | ~x2 | x3) & (~x1 | ~x2 | ~x3) & x4
Horn formula:  (~x1 | ~x2 | ~x3)  &  x4 & (x2 | ~x3) & (x3 | ~x2)

Variable neighborhood results:
Number of assignments in Horn core: 3
Maximal Horn Core: [(0, 0, 0, 1), (1, 0, 0, 1), (0, 1, 1, 1)]
Formula: (x1 | x2 | ~x3) & (x1 | ~x2 | x3) & (~x1 | x2 | ~x3) & (~x1 | ~x2 | x3) & (~x1 | ~x2 | ~x3) & x4
Horn formula:  (~x1 | ~x2 | ~x3)  &  x4 & (x2 | ~x3) & (x3 | ~x2)
```

Slika 4.1: Rezultati za mali testni skup 1

```
Number of assignments: 10
M: [(0, 0, 0, 1), (0, 1, 0, 1), (0, 0, 1, 0), (0, 1, 0, 0), (0, 1, 1, 0), (1, 0, 1, 0), (0, 0, 0, 0), (1, 0, 0, 1), (1, 0, 0, 0), (0, 0, 1,
1)]

Brute force results:
Number of assignments in Horn core: 10
Maximal Horn Core: [(0, 0, 0, 1), (0, 1, 0, 1), (0, 0, 1, 0), (0, 1, 0, 0), (0, 1, 1, 0), (1, 0, 1, 0), (0, 0, 0, 0), (1, 0, 0, 1), (1, 0,
0, 0), (0, 0, 1, 1)]
Formula: (x1 | ~x2 | ~x3 | ~x4) & (~x1 | x2 | ~x3 | ~x4) & (~x1 | ~x2 | x3 | x4) & (~x1 | ~x2 | x3 | ~x4) & (~x1 | ~x2 | ~x3 | x4) & (~x1 |
~x2 | ~x3 | ~x4)
Horn formula:  (~x1 | x2 | ~x3 | ~x4)  &  (~x1 | ~x2 | ~x3 | x4)  & (~x4 | ~x2 | ~x3) & (x3 | ~x2 | ~x1)

Greedy search results:
Number of assignments in Horn core: 10
Maximal Horn Core: [(0, 0, 0, 1), (0, 1, 0, 1), (0, 0, 1, 0), (0, 1, 0, 0), (0, 1, 1, 0), (0, 0, 0, 0), (1, 0, 1, 0), (1, 0, 0, 1), (1, 0,
0, 0), (0, 0, 1, 1)]
Formula: (x1 | ~x2 | ~x3 | ~x4) & (~x1 | x2 | ~x3 | ~x4) & (~x1 | ~x2 | x3 | x4) & (~x1 | ~x2 | x3 | ~x4) & (~x1 | ~x2 | ~x3 | x4) & (~x1 |
~x2 | ~x3 | ~x4)
Horn formula:  (~x1 | x2 | ~x3 | ~x4)  &  (~x1 | ~x2 | ~x3 | x4)  & (~x4 | ~x2 | ~x3) & (x3 | ~x2 | ~x1)

Variable neighborhood results:
Number of assignments in Horn core: 10
Maximal Horn Core: [(0, 0, 0, 1), (0, 1, 0, 1), (0, 0, 1, 0), (0, 1, 0, 0), (0, 1, 1, 0), (0, 0, 0, 0), (1, 0, 1, 0), (1, 0, 0, 1), (1, 0,
0, 0), (0, 0, 1, 1)]
Formula: (x1 | ~x2 | ~x3 | ~x4) & (~x1 | x2 | ~x3 | ~x4) & (~x1 | ~x2 | x3 | x4) & (~x1 | ~x2 | x3 | ~x4) & (~x1 | ~x2 | ~x3 | x4) & (~x1 |
~x2 | ~x3 | ~x4)
Horn formula:  (~x1 | x2 | ~x3 | ~x4)  &  (~x1 | ~x2 | ~x3 | x4)  & (~x4 | ~x2 | ~x3) & (x3 | ~x2 | ~x1)
```

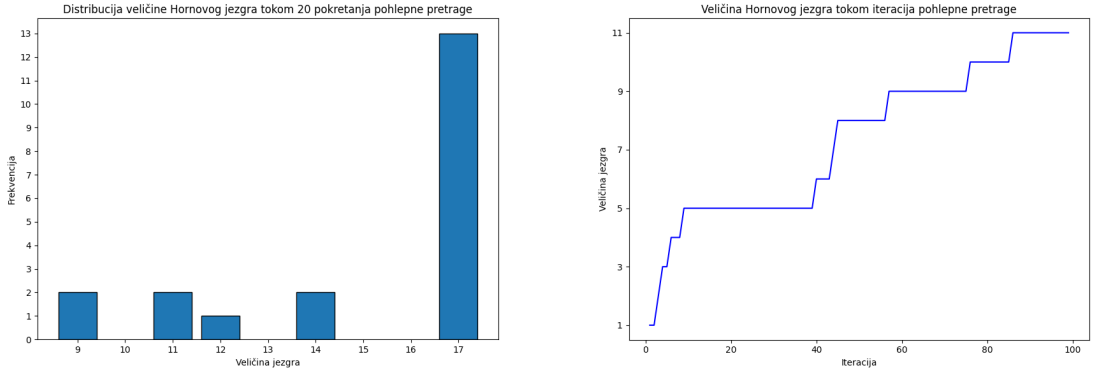
Slika 4.2: Rezultati za mali testni skup 2

Rezultati testiranja malih testnih skupova su u skladu sa očekivanjima. Rezultati pokazuju da su algoritmi uspeli da pronađu maksimalna Hornova jezgra kako je predviđeno, potvrđujući tačnost i pouzdanost njihovih implementacija. Ovi testovi pružaju osnovu za analizu performansi na složenijim i većim testnim skupovima.

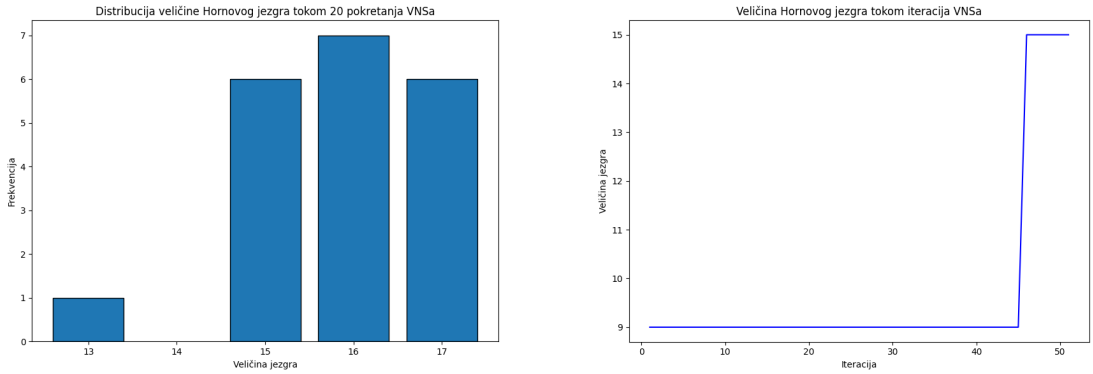
Prethodne studije i teorijska očekivanja sugerišu da bi VNS algoritam trebalo da nadmaši pohlepnu pretragu u pronalaženju najvećih Hornovih jezgara. Očekuje se da VNS, zbog svoje sposobnosti da koristi lokalne pretrage i promenljivu strukturu susedstva, pronađe bolja rešenja u poređenju sa pohlepnom pretragom. Pohlepna

pretraga, iako brza i jednostavna, može praviti kompromise u kvalitetu rešenja zbog svoje determinističke prirode i sklonosti da se zaglavi u lokalnim optimumima. Teorijski, pohlepna pretraga je pogodna za situacije kada je brzina kritični faktor, ali njena sposobnost da pronađe globalno optimalna rešenja je ograničena.

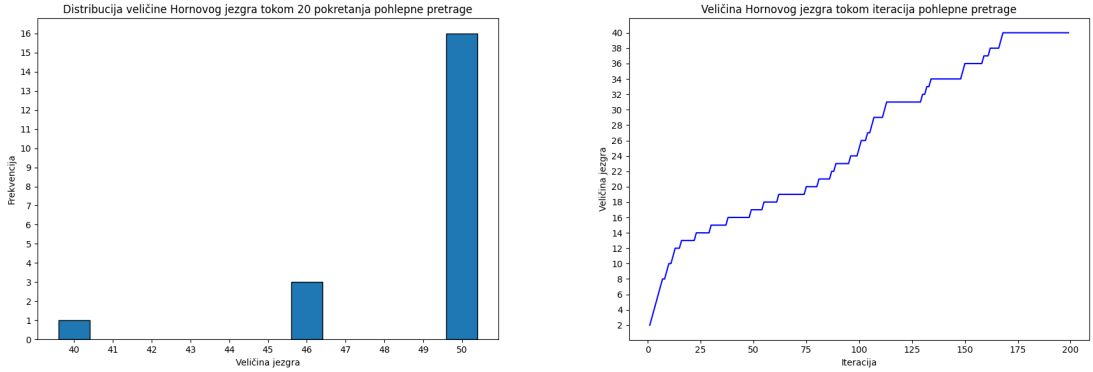
U nastavku su prikazani rezultati testiranja algoritma VNSa i pohlepne pretrage na tri odabrana velika testna skupa. Na slikama 4.3, 4.4, 4.5, 4.6, 4.7, i 4.8 mogu se videti stubasti dijagrami koji prikazuju veličinu Hornovog jezgra za 20 pokretanja ovih algoritama, kao i promenu veličine jezgra tokom iteracija jednog pokretanja algoritma. Svaka slika omogućava uvid u ova dva aspekta performansi algoritama, varijacije u veličini jezgra između različitih pokretanja i dinamiku veličine jezgra kroz iteracije. Tabela 4.1 sadrži detaljne rezultate performansi. Ova tabela prikazuje redni broj testnog skupa (M), broj promenljivih (N), broj modela u testnom skupu ($|M|$), prosečno vreme po pokretanju algoritma (T_{avg}) i ukupno vreme izvršavanja svih 20 pokretanja algoritma (T_{tot}), kao i prosečnu veličinu najboljeg Hornovog jezgra za sva pokretanja algoritma (V_{avg}) i veličinu najboljeg jezgra pronađenog tokom svih pokretanja (V_{tot}). Veličina jezgra predstavlja broj modela uključenih u to jezgru.



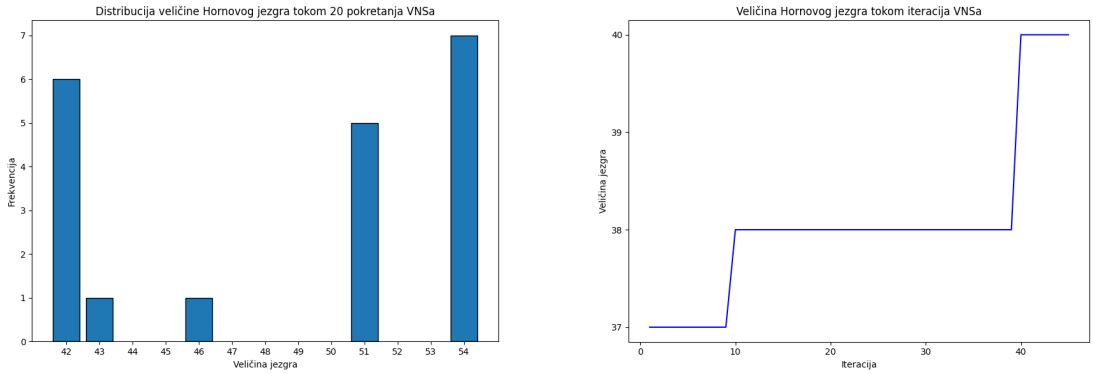
Slika 4.3: Grafik performansi pohlepne pretrage na testnom skupu sa 8 promenljivih i 100 modela



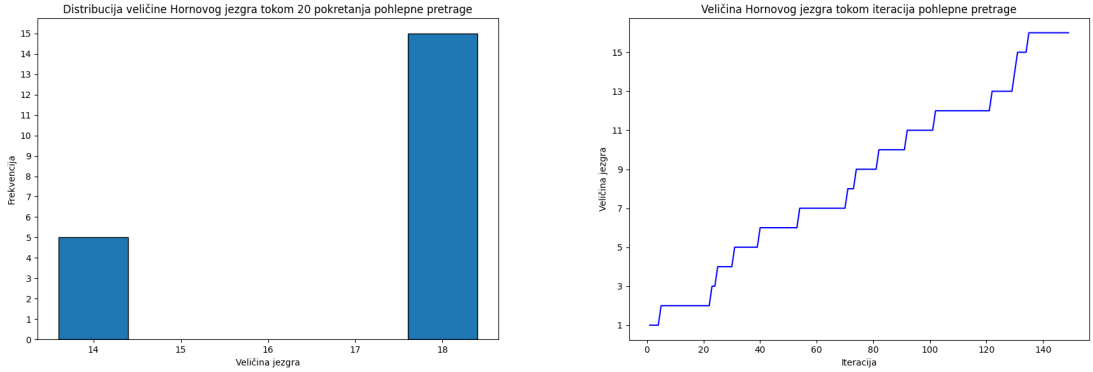
Slika 4.4: Grafik performansi VNSa na testnom skupu sa 8 promenljivih i 100 modela



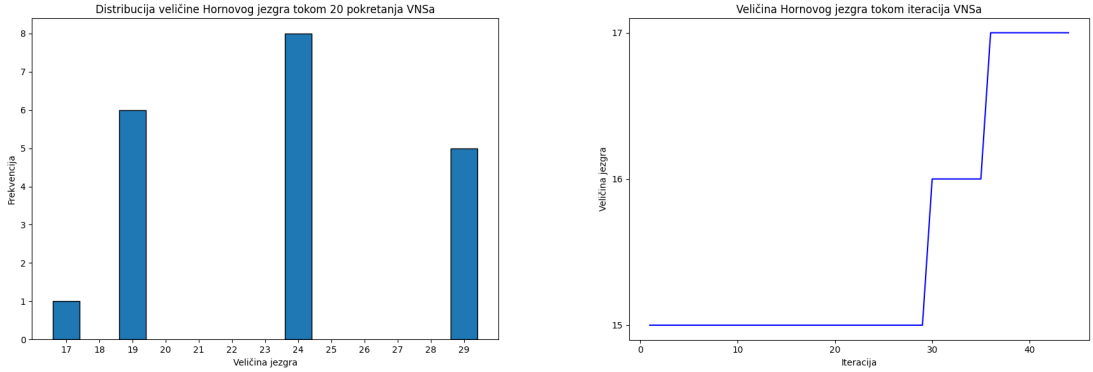
Slika 4.5: Grafik performansi pohlepne pretrage na testnom skupu sa 8 promenljivih i 200 modela



Slika 4.6: Grafik performansi VNSa na testnom skupu sa 8 promenljivih i 200 modela



Slika 4.7: Grafik performansi pohlepne pretrage na testnom skupu sa 9 promenljivih i 150 modela



Slika 4.8: Grafik performansi VNSa na testnom skupu sa 9 promenljivih i 150 modela

M	**N**	**|M|**	**Pohlepna pretraga**				**VNS**			
			$\mathbf{T}_{avg}$	$\mathbf{T}_{tot}$	$\mathbf{V}_{avg}$	$\mathbf{V}_{tot}$	$\mathbf{T}_{avg}$	$\mathbf{T}_{tot}$	$\mathbf{V}_{avg}$	$\mathbf{V}_{tot}$
Skup 1	5	15	0.001	0.010	5.85	7	0.007	0.143	6.60	7
Skup 2	5	25	0.002	0.031	16.75	18	0.046	0.920	17.55	18
Skup 3	6	30	0.001	0.015	8.25	10	0.022	0.431	9.35	10
Skup 4	6	50	0.003	0.056	19.80	24	0.146	2.914	21.30	24
Skup 5	7	60	0.001	0.029	11.40	17	0.128	2.566	16.75	22
Skup 6	7	100	0.008	0.166	25.55	34	0.833	16.665	31.40	37
Skup 7	8	50	0.001	0.012	5.40	8	0.043	0.861	8.75	10
Skup 8	8	100	0.002	0.044	11.85	17	0.218	4.351	13.30	17
Skup 9	8	200	0.045	0.896	39.20	50	4.646	92.920	41.95	54
Skup 10	9	75	0.001	0.016	5.50	8	0.049	0.974	6.25	8
Skup 11	9	150	0.004	0.085	13.20	18	0.535	10.703	17.90	29
Skup 12	9	225	0.030	0.600	28.40	35	5.008	100.158	33.60	40
Skup 13	10	100	0.001	0.024	6.10	12	0.134	2.675	9.75	13
Skup 14	10	200	0.006	0.116	12.20	19	0.684	13.673	14.00	20
Skup 15	10	300	0.008	0.158	13.20	18	1.409	28.189	17.80	22

Tabela 4.1: Rezultati na svim velikim testnim skupovima

Rezultati testiranja velikih testnih skupova potvrđuju prethodna očekivanja. VNS algoritam zaista nadmašuje pohlepnu pretragu u pronalaženju većih Hornovih jezgara. Na primer, na testnom skupu 12, VNS je pronašao jezgro veličine 40, dok je pohlepna pretraga pronašla jezgro veličine 35. Ovi rezultati potvrđuju da VNS koristi svoje napredne tehnike pretrage kako bi pronašao globalno bolja rešenja.

Takođe, pohlepna pretraga pokazuje brže vreme izvršavanja, ali često sa većim oscilacijama u kvalitetu rešenja tokom različitih iteracija. Ove oscilacije ukazuju na njenu sklonost da se zaglavi u lokalnim optimumima, što je u skladu sa teorijskim očekivanjima. VNS se, s druge strane, pokazuje kao robustan pristup u pronalaženju rešenja koja pohlepna pretraga može prevideti, što potvrđuje njegovu prednost u preciznosti i stabilnosti rešenja.

Dobijeni rezultati su u skladu sa postojećim istraživanjima, koji sugerišu da VNS pruža bolje rezultate u pronalaženju globalno optimalnih rešenja. Dok pohlepna pretraga nudi prednosti u brzini i jednostavnosti, prethodne studije su istakle da VNS koristi složenije tehnike za izbegavanje lokalnih optimuma, što je evidentno u njegovoj sposobnosti da dosledno identifikuje veća jezgra u poređenju sa pohlepnom pretragom.

U zaključku, dok VNS pokazuje superiornost u preciznosti, pohlepna pretraga ostaje važan alat zbog svoje jednostavnosti i brzine. Izbor između ova dva algoritma zavisi od specifičnih potreba aplikacije i raspoloživih resursa. Ova analiza omogućava donošenje informisanih odluka o tome koji algoritam koristiti u zavisnosti od specifičnih zahteva problema.

Glava 5

Zaključak i pravac daljeg rada

U ovom radu razmotrena je problematika pronalaženja maksimalnog Hornovog jezgra, koristeći različite algoritme i tehnike za evaluaciju efikasnosti pristupa. Eksperimenti su sprovedeni na malim i velikim skupovima podataka. Analizom su obuhvaćeni kako metaheuristički pristupi, kao što su metoda promenljivih okolina (VNS) i pohlepna pretraga, tako i klasična metoda grube sile.

Eksperimentalni rezultati su pokazali da metaheuristički algoritmi pružaju značajnu prednost u smislu efikasnosti prilikom obrade velikih skupova podataka, dok pristup grube sile ostaje koristan za male instance zbog svoje tačnosti. Analizom rezultata na velikim skupovima podataka može se zaključitit da VNS pruža nešto bolje rezultate u poređenju sa pohlepnom pretragom, što ukazuje na njegovu superiornost u ovom kontekstu.

Može se zaključiti da su metaheuristički algoritmi, kao što su metoda promenljivih okolina i pohlepna pretraga, veoma korisni za rešavanje problema maksimalnog Hornovog jezgra zbog svoje sposobnosti da nalaze približna rešenja u razumnom vremenskom okviru.

Razvijeni algoritam za generisanje Hornove formule pokazao je zadovoljavajuće rezultate u testiranju, ali postoje mogućnosti za dalja unapređenja. Trenutna verzija algoritma koristi osnovne tehnike za generisanje formula, ali u budućem radu bi moglo biti korisno implementirati sofisticiranije metode koje bi mogle poboljšati kvalitet generisanih formula i ubrzati proces generacije.

Za budući rad, preporučuje se dalja optimizacija postojećih metaheurističkih algoritama kako bi se poboljšala njihova tačnost i brzina, posebno u kontekstu vrlo velikih skupova podataka. Takođe, istraživanje novih metaheurističkih pristupa i njihova komparacija sa tradicionalnim metodama može pružiti dodatne uvide u

efikasnost različitih tehnika. Implementacija naprednih tehnika kao što su hibridni algoritmi, koji kombinuju prednosti različitih metaheuristika, mogla bi biti korisna za dalje unapređenje rešenja problema maksimalnog Hornovog jezgra.

Pored toga, uvođenje novih evaluacijskih kriterijuma i testiranje na različitim vrstama instanci može doprineti razumevanju ograničenja i prednosti trenutnih metoda. Takođe, istraživanje mogućnosti primene razvijenih algoritama u drugim sličnim problemima i oblastima može otvoriti nove pravce istraživanja i primene.

Zaključno, ovaj rad pruža koristan doprinos razumevanju i primeni metaheurističkih tehnika za rešavanje kompleksnih problema, kao što je problem pronalaženja maksimalnog Hornovog jezgra. Istraživanje je omogućilo uvid u efikasnost različitih metoda i njihove prednosti i ograničenja. Takođe, pružena su osnova i smernice za buduća istraživanja i unapređenja, otvarajući prostor za dalje inovacije i optimizacije u ovoj oblasti.

Bibliografija

[1] Maximum Horn core. on-line at: `https://www.csc.kth.se/~viggo/wwwcompendium/node239.html`.

[2] Nikolaj Bjørner, Arie Gurfinkel, Ken McMillan, and Andrey Rybalchenko. *Horn Clause Solvers for Program Verification*, pages 24–51. Springer International Publishing, Cham, 2015.

[3] Nikolaj Bjørner, Fabio Fioravanti, Andrey Rybalchenko, and Valerio Senni. Proceedings first workshop on horn clauses for verification and synthesis. *Electronic Proceedings in Theoretical Computer Science*, 169, 2014.

[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms, fourth edition.* MIT Press, 2022.

[5] John Doyle and Ramesh S. Patil. Two theses of knowledge representation: language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence*, 48(3):261–297, 1991.

[6] Thomas Eiter, Toshihide Ibaraki, and Kazuhisa Makino. Disjunctions of horn theories and their cores. In *Algorithms and Computation*, pages 50–60, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[7] Pierre Hansen and Nenad Mladenovic. *Variable Neighborhood Search Methods*, volume 22, pages 3978–. 2009.

[8] Pierre Hansen, Nenad Mladenovic, and José Moreno-Pérez. Variable neighbourhood search: Methods and applications. *4OR*, 175:367–407, 2010.

[9] Alfred Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, pages 14–21, 1951.

[10] Eric J. Horvitz, Gregory F. Cooper, and David E. Heckerman. Reflection and action under scarce resources: theoretical principles and empirical study. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 2*, page 1121–1127, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[11] Dimitris Kavvadias, Christos H. Papadimitriou, and Martha Sideri. On horn envelopes and hypergraph transversals. In *Algorithms and Computation*, pages 399–405, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[12] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[13] Ben-Ari Mordechai. Mathematical logic for computer science. London, 2012. Springer London.

[14] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a modern approach.* Pearson, 2009.

[15] Roberto Sebastiani and Michele Vescovi. Axiom pinpointing in lightweight description logics via horn-sat encoding and conflict analysis. In *Automated Deduction – CADE-22*, pages 84–99, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[16] Bart Selman and Henry Kautz. Knowledge compilation using horn approximations. Anaheim, 1991. Proceedings of the Ninth National Conference on Artificial Intelligence.

Biografija autora

Nikola Belaković je rođen u Kraljevu 14. septembra 2000. godine. Išao je u osnovnu školu „Milun Ivanović" u Kraljevu, završio je odličnim uspehom i bio nagrađen Vukovom diplomom. Nakon završene osnovne škole upisao je prirodno-matematički smer „Gimanzije Kraljevo". Po završetku srednje škole, odlučuje da upiše smer Informatika na Matematičkom fakultetu u Beogradu. Osnovne studije završava u roku od četiri godine i upisuje master studije na istom smeru.