UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Nikola Belaković

ALGORITMI ZA REŠAVANJE PROBLEMA HORNOVOG JEZGRA

master rad

Beograd, 2024.

Mentor:

dr Aleksandar KARTELJ, redovni 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 i računarskih nauka. Ovaj problem se može opisati kao pronalaženje najvećeg podskupa modela M' iz skupa M, takvog da postoji Hornova formula čiji su svi modeli sadržani u 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 obrađen je u ovom radu i za njega je dokazano da je NP-kompletan, a detaljan dokaz može se naći u radu [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 3, 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 logici koja predstavlja atomičku formulu (poznatu i kao atom ili osnovna formula) ili njenu negaciju [13]. Hornova klauza je klauza koja sadrži najviše jedan pozitivan literal [9]. Hornova formula je konjukcija Hornovih klauza, koja može biti predstavljena pomoću skupa Hornovih klauza [11], jer su klauze u konjuktivnoj normalnoj formi. Konjunktivna normalna forma (CNF) je logički izraz koji se sastoji od konjunkcije jedne ili više disjunkcija literala.

Model t je skup $\{0,1\}^n$, istinitosnih valuacija boolovskih promenljivih. Model zadovoljava klauzu ako je neki literal klauze x_i i $t_i = 1$ ili je neki literal klauze $\neg x_i$ i $t_i = 0$. 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). Funkcija cilja za ovaj problem, koju je potrebno maksimizovati, je kardinalnost Hornovog jezgra [11, 1].

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

U nastavku će biti prikazana računska kompleksnost ovog algoritamskog problema:

a) Postoji polinomijalni algoritam koji za dato M generiše jedan maksimalni podskup M' koji predstavlja Hornovo jezgro. Zapravo, svi maksimalni podskupovi mogu biti izračunati u polinomijalnom vremenu. Maksimalnost podskupa se ogleda u tome da ne može biti proširen dodavanjem još neke istinitosne valuacije iz M tako da i dalje ostane Hornovo jezgro.

   Dokaz: Kreira se Hornovo jezgro koristeći modele iz M. Za svaki par t i $t' \in M$ se proverava da li $t \wedge t' = t'' \in M$. Ako ovo važi, onda se u jezgro dodaju t, t' i t''. U suprotnom, ako $t'' \notin M$, samo jedno od t ili t' se mogu naći u jezgru.

b) Pronalazak Hornovog jezgra maksimalne kardinalnosti M' je NP-kompletan problem

Dokaz: Redukcija NP-kompletog problema klike. Za graf $G = (V, E)$ i ceo broj k konstruiše se skup modela M $\subseteq\{0,1\}^{|E|}$ (ekvivalentno, podskupovi od E) tako što M sadrži sve jednostruke skupove $\{e\}$, i, za svaki čvor $v \in V$, skup $\{e \in E : v \in e\}$. Iz ovoga sledi da je Hornovo jezgro maksimalne kardinalnosti M' veličine $|E| + k$ ako i samo ako je klika za graf G veličine k.

c) Ovaj problem je NP-težak za aproksimaciju bilo kojim konstantnim faktorom

Dokaz: Koristi pojačanu konstrukciju iz prethodnog dokaza [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 kompleksnosti 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 CNF-ova [6].

Pošto se metod 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. metod promenljivih okolina (*eng.* variable neighborhood search).

Osim ove dve metaheuristike, biće opisan i polinomijalni algoritam 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 $v \in \{0,1\}^n$, čija je i-ta komponenta označena sa v_i. Teorija je bilo koji skup $M \subseteq \{0,1\}^n$ modela. Nad komponentama v je definisano bitovsko uređenje u oznaci $v \leq w$, gde važi $0 \leq 1$.

Teorija je Hornova ako je $M = Cl_\wedge(M)$, gde je $Cl_\wedge(S)$ zatvoranje skupa $S \subseteq \{0,1\}^n$ za operaciju bitovske konjukcije (tj. preseka) modela v i w, označeno sa $v \wedge w$.

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 možemo jednostavno kreirati algoritam za pronalaženje jednog Hornovog jezgra teorije M. Na početku se Hornovo jezgro M' inicijalizuje praznim skupom. Zatim se iterira 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 konjukciju, tako što se za svaki par $v, w \in M'$ proverava da li je $v \wedge w \in M'$. Ako jeste, dodatu 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

Input: Skup modela M
Output: Hornovo jezgroM'

```
1: Inicijalizuj M' kao prazan skup
2: Dodaj komponentu modela 1 u M' - Uvek dodaj prvu komponentu
3: for i = 2 to n do
4:    Dodaj komponentu modela i u M'
5:    zadrzi_komponentu := true
6:    for each 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:      Obriši komponentu modela i iz M'
14:   end if
15: end for
16: return M'
```

2.2 Pohlepna pretraga

Pohlepna pretraga je algoritam koji se koristi u matematici i računarstvu za rešavanje optimizacionih problema. Ovaj pristup donosi 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 algo-

ritam 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. Najvažniji je taj što ne garantuje uvek optimalno rešenje. Još jedan njen problem je što se može zaglaviti u lokalnom optimumu, i na taj način će pronaći dobro, ali ne i najbolje moguće rešenje. Zbog ovih nedostataka, upotreba ovog algoritma nije predviđena za probleme u kojima je ključno pronaći globalno optimalno rešenje [4].

Algoritam 2 Greedy search

Input: Početno rešenje x, Maksimalan broj iteracija $maxIter$
Output: 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 Metod promenljivih okolina

Metoda promenljivih okolina je metaheuristički algoritam za rešavanje problema optimizacije [12]. VNS[1] se bazira na 3 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 sistematskog i stohastičkog menjanja različitih okolina tokom pretrage prostora rešenja, čime se izbegava lokalni optimum i povećava šansa za pronalaženje globalnog optimuma i njeni osnovni principi su:

[1]U nastavku teksta VNS će biti korišćeno kao skraćenica za metodu promenljivih okolina

- **Definisanje početnog rešenja:** Pretraga počinje od nekog početnog rešenja, koje može biti nasumično generisano ili dobijeno nekom heuristikom
- **Lokalna popravka početnog rešenja:** Primenom lokalne pretrage na početno rešenje dobija se prvo poboljšanje i optimizuje se početno rešenje
- **Razmrdavanje (*eng.* shaking:)** Promena trenutnog rešenja slučajnim izborom rešenja iz trenutne okoline, koja ima za cilj izalzak 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.
- **Pomeraj:**
  - 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 okruženje potencijalnih rešenja.
- **Kriterijum zaustavljanja:** 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 Variable Neighborhood Search (VNS)

Input: Početno rešenje x, Skup okolina $\{N_1, N_2, \ldots, N_k\}$, Maksimalan broj iteracija $maxIter$

Output: Najbolje rešenje x^*

```
1: x* ← x
2: iter ← 0
3: while iter < maxIter do
4:    x' ← LokalnaPopravka(x*) {Lokalna popravka početnog rešenja}
5:    k ← 1
6:    while k ≤ k_max do
7:       x'' ← Razmrdavanje(x', N_k) {Faza razmrdavanja (shaking)}
8:       x''' ← LokalnaPretraga(x'') {Faza popravljanja}
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, kreiraju 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

Input: Skup modela
Output: Indeksi zajedničkih jedinica

```
for svaki indeks i u dužini modela do
    if svi modeli imaju jedinicu na indeksu i then
        Dodaj i u listu zajedničkih jedinica
    end if
end for
```

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

Input: Skup modela, zajedničke jedinice
Output: Skraćeni modeli

```
for svaki model do
    Ukloni zajedničke jedinice iz modela
    Sačuvaj skraćeni model uz originalni
end for
```

Generisanje svih mogućih binarnih kombinacija predstavlja sledeći korak. Ove kombinacije su osnova za stvaranje formule koja zadovoljava kompletan skup modela.

Algoritam 6 Generisanje kombinacija i formule

Input: Broj varijabli, zajedničke jedinice
Output: CNF formula

```
Generiši sve binarne kombinacije
for svaku kombinaciju do
    Dodaj zajedničke jedinice na odgovarajuće pozicije
    if kombinacija nije u skraćenim modelima then
        Dodaj kao klauzu u CNF
    end if
end for
```

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

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

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

Input: CNF formula
Output: Reducirana Hornova formula

```
1: while postoje ne-Hornove klauze do
2:    for sve parove klauza do
3:       Pokušaj rezoluciju
4:       if rezolucija uspešna then
5:          Zameni klauze novom
6:          Prekini petlju
7:       end if
8:    end for
9:    if rezolucija neuspešna then
10:      Promešaj klauze i pokušaj ponovo
11:   end if
12: end while
```

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 kombinacija, kreiranje CNF formule, i redukciju klauza do Hornove forme. Proces omogućava strukturirano i sistematično 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 kombinacija:** Generisanje svih mogućih binarnih kombinacija za n varijabli dovodi do 2^n kombinacija. Svaka kombinacija se obrađuje i dodaje u skup klauza, što može biti eksponencijalno u odnosu na broj varijabli. 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 varijabli. 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 varijabli 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

Evaluacija algoritma

U ovom poglavlju biće prikazani i analizirani rezultati implementiranih algoritama, uključujući testne podatke koji su korišćeni za evaluaciju 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 pet testnih skupova podataka: dva mala i tri velika. Ova podela omogućila je detaljnu analizu i upoređivanje različitih pristupa, uključujući metaheuristike i bruteforce metode.

Testni skupovi su generisani pomoću algoritma koji kreira modele sa binarnim vrednostima. Algoritam prvo identifikuje sve moguće kombinacije vrednosti varijabli, zatim nasumično odabira deo tih kombinacija kako bi stvorio specifične skupove podataka. Ovi skupovi su zatim zapisani u tekstualne datoteke za dalju analizu i testiranje.

Algoritam 8 Generisanje testnih skupova

Input: Broj varijabli n
Output: Lista generisanih modela
1: Izračunaj maksimalan broj kombinacija kao 2^n
2: Odredi minimalan broj kombinacija
3: Nasumično izaberi broj kombinacija između minimalnog i maksimalnog broja
4: **while** broj generisanih kombinacija manji od odabranog broja **do**
5: Generiši nasumičnu binarnu kombinaciju
6: Dodaj kombinaciju u skup ako nije već prisutna
7: **end while**
8: **return** Lista kombinacija

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 bruteforce metode, 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 osam i devet promenljivih, testirane su isključivo metaheuristike. Ovi veliki skupovi su uključivali 93 valuacija za jedan skup, 330 valuacija za drugi i 221 za treći skup. Testiranje ovih skupova sa bruteforce metodom 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 varijable. Svaki model je niz od četiri binarne vrednosti koje odgovaraju stanjima varijabli. 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
- **Kernel:** 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
- **RAM memorija:** Kingston 16GB DDR4
- **Grafička kartica:** Mesa Intel Graphics (ADL GT2)

Iako procesor ima 12 jezgara, algoritmi nisu implementirani za paralelno izvršavanje, što može uticati na brzinu obrade. Svi algoritmi i funkcije napisani su u Pythonu, koristeći Jupyter notebook za razvoj i testiranje. 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 numerisanim tekstualnim fajlovima, što omogućava lakšu organizaciju i ponovnu upotrebu i analizu tih podataka.

Programski kod je organizovan u šest skripti, od kojih se tri koriste za implementaciju algoritama i metaheuristika (bruteforce, VNS i greedy search). 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 kreiranje grafičkog prikaza i vizualizacije podataka.

3.3 Rezultati eksperimenta

U ovom delu biće prikazani eksperimentalni rezultati izvršavanja algoritama. Na slikama 3.1 i 3.2 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 tačnosti njihove tačnosti.

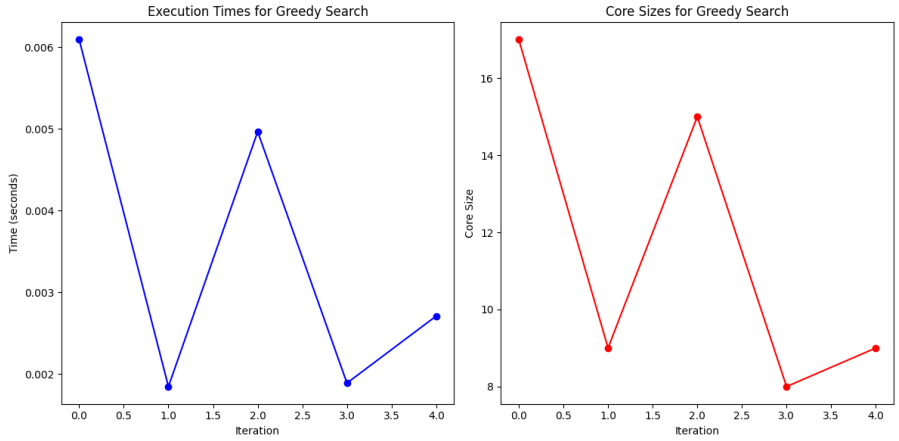
```
Number of assignments: 4
M: [(0, 0, 0, 1), (0, 1, 0, 0), (1, 0, 0, 1), (0, 1, 1, 1)]

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 3.1: Rezultati za mali testni skup 1

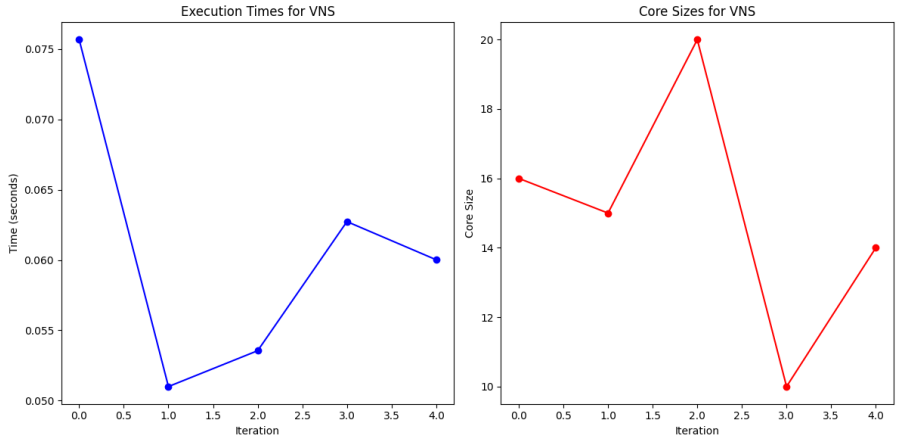
```
Number of assignments: 10
M: [(0, 0, 0, 1), (0, 1, 0, 1), (0, 0, 1, 0), (0, 1, 0, 0), (0, 1, 1, 0), (1, 0, 1, 0), (0, 0, 0, 0), (1, 0, 0, 1), (1, 0, 0, 0), (0, 0, 1,
1)]

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 3.2: Rezultati za mali testni skup 2

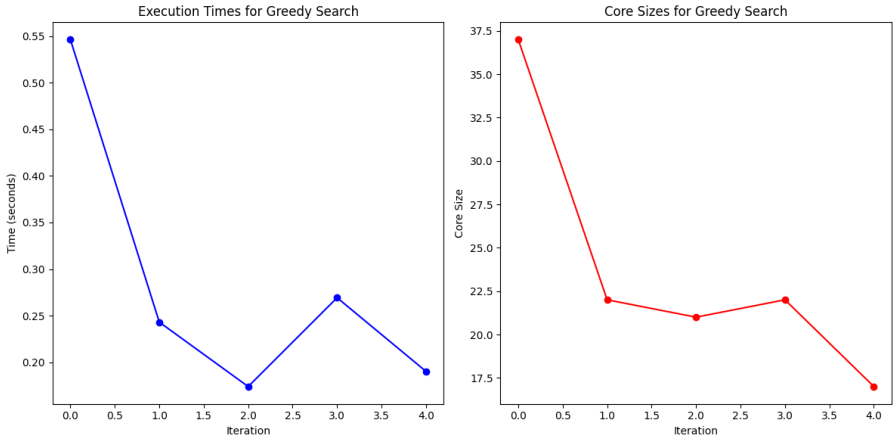
U nastavku su prikazani rezultati testiranja algoritma VNSa i pohlepne pretrage na tri velika testna skupa. Na slikama 3.3, 3.4, 3.5, 3.6, 3.7, i 3.8 možete videti grafike vremena izvršavanja i veličine Hornovog jezgra ovih algoritama na sva tri testna skupa. Svaka slika prikazuje varijacije u performansama tokom različitih poziva funkcije za odgovarajući testni skup. Tabela 3.1 sadrži detaljne rezultate performansi. Ova tabela prikazuje testni skup, vreme izvršavanja, prosečno vreme, veličinu najboljeg Hornovog jezgra i prosečnu veličinu jezgra za oba algoritma.



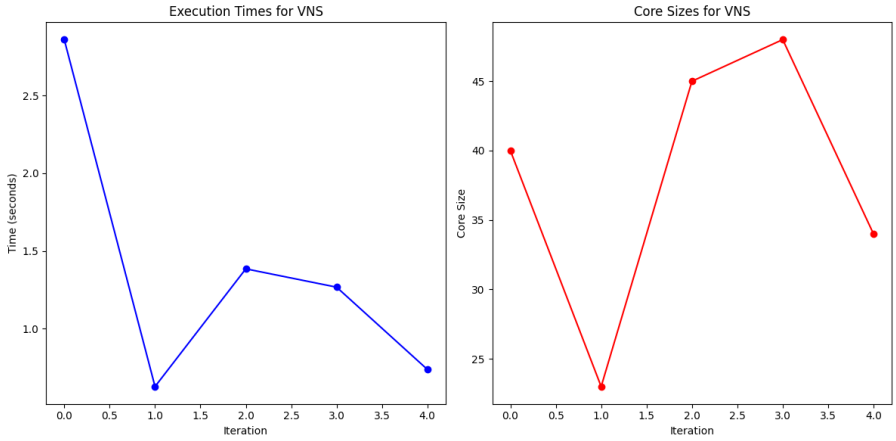
Slika 3.3: Grafik vremena izvršavanja i veličine Hornovog jezgra za pohlepnu pretragu na prvom velikom testnom skupu.



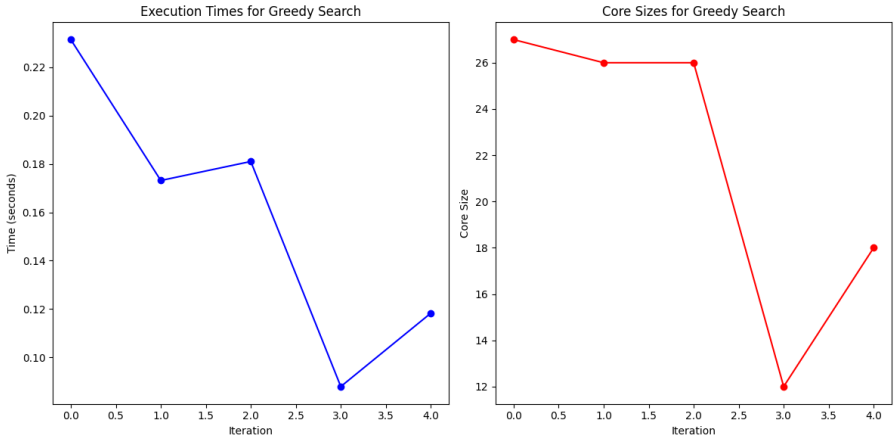
Slika 3.4: Grafik vremena izvršavanja i veličine Hornovog jezgra za VNS algoritam na prvom velikom testnom skupu.



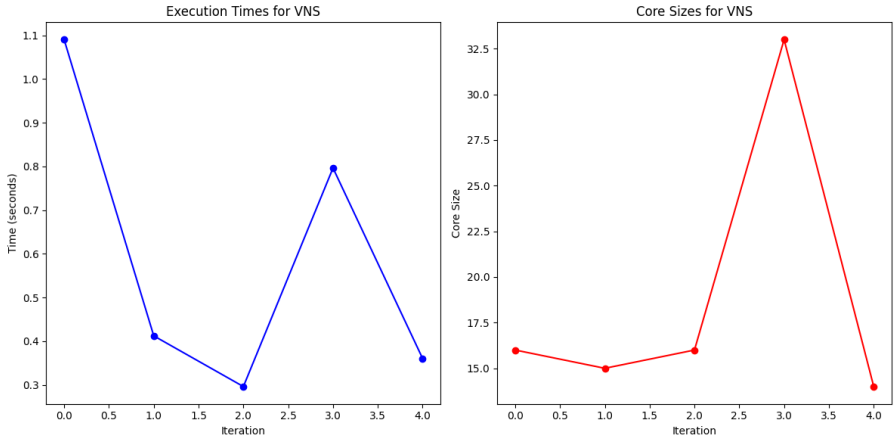
Slika 3.5: Grafik vremena izvršavanja i veličine Hornovog jezgra za pohlepnu pretragu na drugom velikom testnom skupu.



Slika 3.6: Grafik vremena izvršavanja i veličine Hornovog jezgra za VNS algoritam na drugom velikom testnom skupu.



Slika 3.7: Grafik vremena izvršavanja i veličine Hornovog jezgra za pohlepnu pretragu na trećem velikom testnom skupu.



Slika 3.8: Grafik vremena izvršavanja i veličine Hornovog jezgra za VNS algoritam na trećem velikom testnom skupu.

Test skup	Veličina testnog skupa	Algoritam	Prosečno vreme (s)	Ukupno vreme (s)	Prosečna veličina jezgra	Najveća veličina jezgra
Skup 1	93	Pohlepna pretraga	0.004	0.018	11.6	17
		VNS	0.061	0.303	15.0	20
Skup 2	330	Pohlepna pretraga	0.285	1.423	23.8	37
		VNS	1.376	6.880	38.0	48
Skup 3	221	Pohlepna pretraga	0.158	0.792	21.8	27
		VNS	0.591	2.956	18.8	33

Tabela 3.1: Rezultati na tri velika testna skupa

3.4 Diskusija rezultata

U ovoj sekciji razmatramo performanse algoritama pohlepne pretrage i VNSa, analizirajući efikasnost i tačnost dobijenih rezultata. Cilj je da pružimo uvid u prednosti i nedostatke svakog algoritma.

Analiza malih testnih primera pokazuje da algoritmi rade ispravno, jer su rezultati u skladu s onima dobijenim putem bruteforce metode. 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.

Rezultati pokazuju da VNS algoritam nadmašuje pohlepnu pretragu u pronalaženju najvećih Hornovih jezgara. VNS, koji koristi lokalne pretrage i promenljivu strukturu susedstva, uspeva da pronađe optimalnija rešenja. Pohlepna pretraga, s druge strane, ističe se brzinom izvršavanja, što je čini korisnom kada je vreme kritični faktor. Međutim, pohlepna pretraga, iako brza, često pravi kompromise u kvalitetu rešenja. Ovaj algoritam često zaglavi u lokalnim optimumima zbog svoje determinističke prirode i zavisnosti od početnih uslova i nasumičnih izbora tokom izvršavanja. Takođe, kvalitet rezultata pohlepne pretrage dosta osciluje tokom više iteracija izvršavanja algoritma nad istim podacima.

Pored brzine izvršavanja, veličina Hornovog jezgra je ključna metrika performansi. VNS algoritam dosledno pronalazi veća jezgra u poređenju sa pohlepnom pretragom. Na primer, u testnom skupu 2, VNS je pronašao jezgro veličine 48, dok je pohlepna pretraga dala jezgro veličine 37, što ukazuje na VNSovu sposobnost da bolje istraži prostor pretrage. VNS koristi mehanizme za izbegavanje lokalnih optimuma, što doprinosi pronalaženju globalno boljih rešenja.

Dobijeni rezultati su u skladu sa teorijskim očekivanjima. VNS, zbog svoje stohastičke prirode, pokazuje veću robustnost u pronalaženju rešenja koja pohlepna pretraga može prevideti.

Poredeći naše rezultate sa postojećim studijama, vidimo da su naši zaključci u skladu sa nalazima drugih istraživača koji su primetili slične obrasce performansi između ovih algoritama. VNS se često pokazuje kao bolji izbor za probleme gde je pronalaženje globalno optimalnog rešenja od kritične važnosti, dok pohlepna pretraga služi kao brz i jednostavan pristup.

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 4

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 bruteforce metoda.

Eksperimentalni rezultati su pokazali da metaheuristički algoritmi pružaju značajnu prednost u smislu efikasnosti prilikom obrade velikih skupova podataka, dok bruteforce pristup 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.

Zaključujeno je da su metaheuristički algoritmi, kao što su metod 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 Horn formula 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 značajan 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 „Gimantije 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.