Seminarski rad iz Konstrukcije i analize algoritma 2
Five balltree construction algorithms

Emilija Stevanović

Jul, 2024

Sažetak

Balltree je jednostavna geometrijska struktura podataka sa širokim spektrom praktičnih primena. U nastavku je opis i upoređivanje pet različitih algoritama za konstrukciju ove strukture podataka. Naglašen je kompromis između vremena konstrukcije i kvaliteta konstruisanog stabla. Dva od algoritama su onlajn, dva konstruišu strukture iz skupa podataka odozgo nadole, a jedan algoritam koristi pristup odozdo nagore.

Sadržaj

1 Uvod

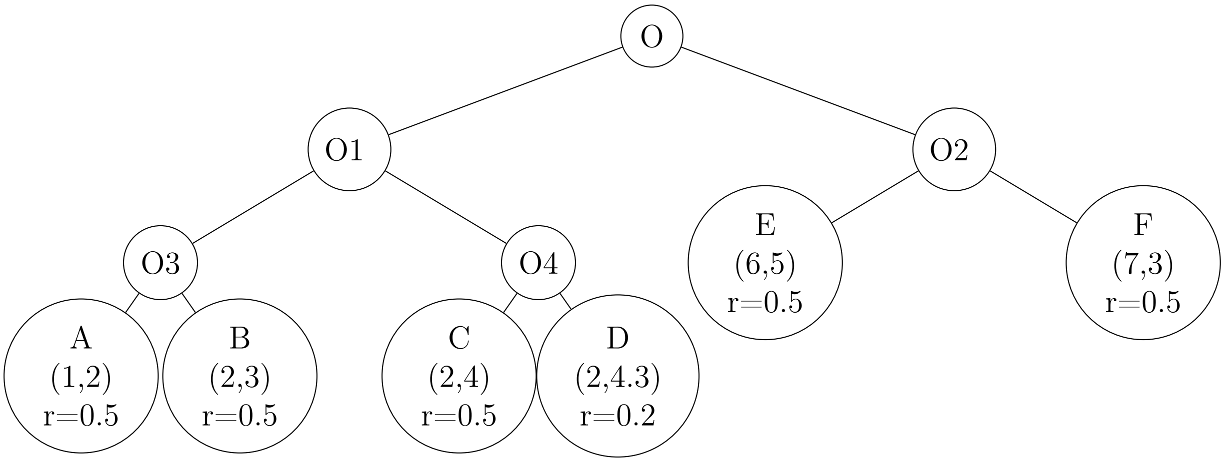
Mnogi zadaci u robotici, viziji, govoru i grafici zahtevaju konstrukciju i manipulaciju geometrijskim reprezentacijama. Balltree struktura podataka je dobro prilagođena geometrijskim zadacima učenja. Balltree struktura se prilagođava strukturi prikazanih podataka, podržava dinamičko umetanje i brisanje, ima dobru prosečnu efikasnost, dobro se nosi sa visoko-dimenzionalnim entitetima i jednostavna je za implementaciju. Podržane su i operacije koje uključuju pretragu najbližih suseda, upite za preseke i ograničenja, kao i maksimizaciju verovatnoće. Osnovne tehnike konstrukcije opisane ovde mogu biti primenjene na širok spektar drugih hijerarhijskih geometrijskih struktura podataka u kojima se lopte zamenjuju kutijama, kockama, elipsoidima itd.

2 Balltree

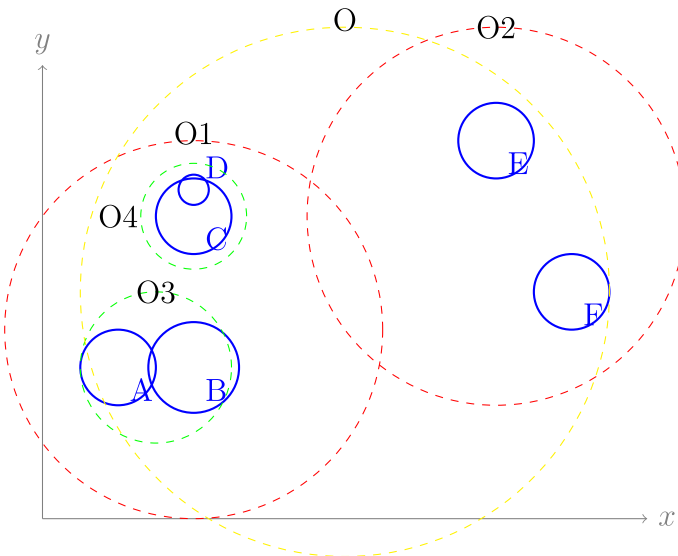
Lopta je oblast unutar hipersfere n-dimenzionalnog Euklidskog prostora $\mathbb{R}^n$. Na primer u 3-dimenzionalnom prostoru, lopta je unutrašnjost sfere. Svaka lopta je predstavljena koordinatama centara lopte i poluprečnikom lopte. Balltree je kompletno binarno stablo u kome su čvorovima pridružene lopte. U kontekstu Balltree algoritma hipersfera je generalizacija sfere na veću dimenziju i predstavlja sve tačke u prostoru koje su na rastojanju manjem ili jednakim od radijusa u odnosu na centar. Unutrašnji čvor je najmanja moguća hipersfera koja sadrži lopte koje pripadaju njegovoj deci. Centar i poluprečnik hipersfere se određuju na osnovu tačaka unutar tog čvora. Listovi čuvaju relevantne informacije. Unutrašnji čvorovi omogućavaju bržu i efikasniju pretragu.U balltree strukturi podataka, regije sinova istog roditelja smeju se preklapati i ne moraju podeliti čitav prostor.

Jedan od scenarija kad se koristi balltree je kada je prostor male dimenzije ugrađen u prostor velike dimenzije(eng. low-dimensional manifold). U visokodimenzionalnom prostoru najbliži sused nekom podatku je na istoj razdaljini kao i najdalji i ne mogu se upotrebiti k-d stabla.

K-d stabla i baltree imaju slične primene kao što je određivanje najbližeg suseda, klasterovanje, klasifikacija, određivanje sličnosti itd. Međutim, balltree struktura podataka je sporija od k-d stabala u dimenzijama ≤ 3, ali mnogo brža u većim dimenzijama i balltree obično i dalje dobro funkcioniše ako podaci pokazuju lokalnu strukturu.[4] Na slikama koje slede je prikazan jedan primer ovakve stukture podataka.



Slika 1: Struktura binarnog Balltree stabla



Slika 2: Grafički prikaz

3 Implementacija klasa u Pajtonu

Klase koje su neophodne za ovu strukturu podataka su implementirane u programskom jeziku Pajton. Klasa Ball predstavlja jednu loptu i određena je x i y koordinatom centra, što u implementaciji predstavlja vektor *coordinates* i poluprečnikom r. Klasa BallTreeNode predstavlja čvor stabla i sadrži instancu klase Ball (*ball*) kao polje, referencu na roditeljski čvor (*parent*), kao i reference na levog (polje u klasi pod nazivom *left*), i desnog potomka(polje klase: *right*). Klasa koja predstsavlja stablo, BallTree, u implementaciji sadrži referencu na koren (polje: *root*) i dodatne metode za konstrukciju.

4 K-d algoritam

Ime ovog algoritma je inspirisano algoritmom za konstrukciju k-d stabla. Ovo je algoritam izgradnje stabla odozgo nadole. U svakoj fazi algoritma lopte se dele na dva skupa koja se obrađuju rekurzivno i tako se dobija binarno stablo. Kriterijum za podelu je odabir dimenzije i vrednosti. Loptice se razvrstavaju na dva dela. U prvom delu su loptice sa manjom odabranom koordinatom centra, a u drugom delu sa većom od vrednosti u odnosu na pivot element. Vrednost u odnosu na koju se formiraju skupovi je medijana (ovde m). Razvrstavanje loptica podseća na QuickSelect algoritam gde se sortiranje vrši u odnosu na koordinatu koja ima najveci opseg vrednosti (ovde c). Složenost određivanja medijane je linerana u proseku (kvadratna u najgorem slučaju), a ima $logN$ faza, ukupna slozenost je $O(N \log N)$. Sledi pseudokod ovog algoritma.

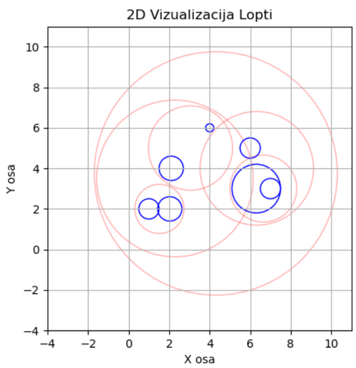
Funkcija 1: build_balltree_range(l, u): BALLTREE_NODE

```
                    ▷ vraća binarno stablo za lopte ciji je indeks u nizu lopti u opsegu [l, u)
if u == l then                                                              ▷ List
    Result := new BallTreeNode()
    Result.ball := bls[u]
    return Result
else                                         ▷ Trazimo po kojoj koordinati sortiramo
    c := bls.most_spread_coord(l, u)
    m := (l + u)//2
    balls.select_on_coord(c, m, l, u)                            ▷ Preuređivanje lopti

    Result := create_balltree_node

    Result.left := build_balltree_range(l, m)
    Result.leftt.parent := Result                              ▷ Roditelj za levog sina

    Result.right := build_balltree_range(m + 1, u)
    Result.right.parent := Result                             ▷ Roditelj za desnog sina

                  ▷ Pravljenje lopte koja obuhvata levog i desnog sina za trenutni čvor
    ball := create_ball
    ball.to_bound_balls(Result.left.ball, Result.right.ball)
    Result.ball := ball
end if
return Result
```

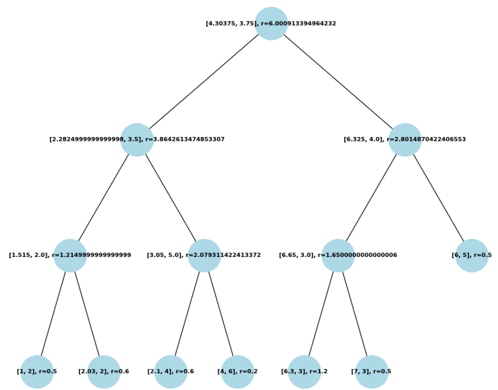
Funkcija 2: select_on_coord(c, k, l, u)

```
                                    ▷ menja niz lopti koji se čuva u klasi BALLTREE
  tako da je k-ta najmanja po koordinati c na svom mestu, levo su one koje imaju c-tu
  koordinatu manju ili jednaku od k-te,a desno one koje imaju vecu
  l := li;
  u := ui;
  while l < u do
      r := random_integer_in_range(l, u);
      pivot := balls[r];
      balls[r] := balls[l];
      balls[l] := pivot;                                                      ▷ swap
      l := l + 1;
      m := l;
                              ▷ Particionisanje elopti u odnosu na odabranu koordinatu c
      for i := l + 1 to u do
          if balls[i].coordinates[c] < pivot.coordinates[c] then
              m := m + 1;
              pom := balls[m];
              balls[m] := balls[i];
              balls[i] := pom;                                                ▷ swap
          end if
      end for

                    ▷ Konacna razmena elemenata tako da je k-ti element na tacnoj poziciji
      pom := elements[l];
      balls[l] := balls[m];
      balls[m] := pom;                                                        ▷ swap

      if m <= k then
          l := m + 1;
      end if
      if m <= k then
          u := m − 1;
      end if
  end while
```

- **Lopta A:** Centar $(-1, 2)$, Radijus 0.8
- **Lopta C:** Centar $(2.1, 4)$, Radijus 0.6
- **Lopta B:** Centar $(2.03, 9)$, Radijus 0.6
- **Lopta E:** Centar $(6, 8)$, Radijus 0.5
- **Lopta D:** Centar $(1, 6)$, Radijus 2
- **Lopta F:** Centar $(7, 3)$, Radijus 0.75
- **Lopta G:** Centar $(8, 4.5)$, Radijus 1.2



(a) Stablo



(b) Stablo

Slika 3: Rezultat

5 Algoritam odozgo nadole

K-d algoritmom se eksplicitno ne minimizuje zapremina stabla. Međutim, asimtotski se dobro ponaša ako imamo podatke sa normalnom raspodelom. Za razliku od k-d pristupa algoritam odozgo nadole (eng. *top down*)koristi heuristiku za odabir dimezije po kojoj se vrši podela skupa lopti. Rekurzivno se "seče" po dimenziji koja ce minimizovati ukupnu zapreminu dve granične lopte dva skupa lopti. Pitanje optimalne dimenzije se rešava konstrukcijom niza troškova po svakoj dimenziji, tako što se loptice najpre sortiraju po jednoj koordinati, a zatim se niz popunjava sekvencijalno u dva prolaza. U prvom prolazu kroz niz, s leva na desno, na i-tu poziciju upisujemo zapreminu lopte koja se dobija proširavanjem prethodne tako da se obuhvati i-ta loptica. U drugom prolazu, sa desna na levo, na i-1 poziciju dodajemo cenu ekspanzije i-tom lopticom. Ovako je na svakoj poziciji dobijen trošak, odnosno zbir zapremina leve i desne lopte ukoliko je to pozicija preseka. Najbolja dimenzija i mesto sečenja se određuje u složenosti $O(Nlog(N))$, a ceo algoritam je složenosti $O(N(log(N))^2)$.

Funkcija 3: fill_in_costs(l,u): void

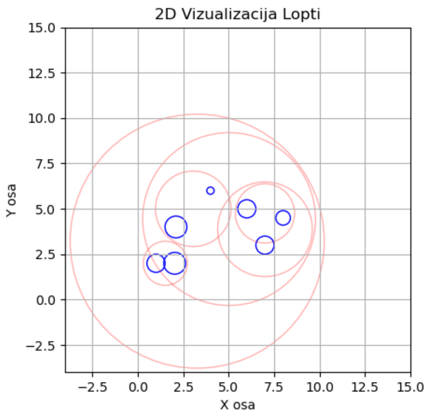
ball.to(balls[l]) ▷ ball preuzima od balls[l] vrednosti radijusa i centara
for $i = l$ **to** $u - 1$ **do**
  ball.expand_to_ball(balls[i]); ▷ ball dobija vrednosti za radijus i centar lopte koja obuhvata i ball i balls[i]

  costs[i] $\leftarrow$ ball.pvol;
end for
ball.to(balls[u]);
for $i = u$ **to** $l + 1$ **do**;
  ball.expand_to_ball(balls[i]);
  costs[$i - 1$] := costs[$i - 1$] + bl.pvol;
end for

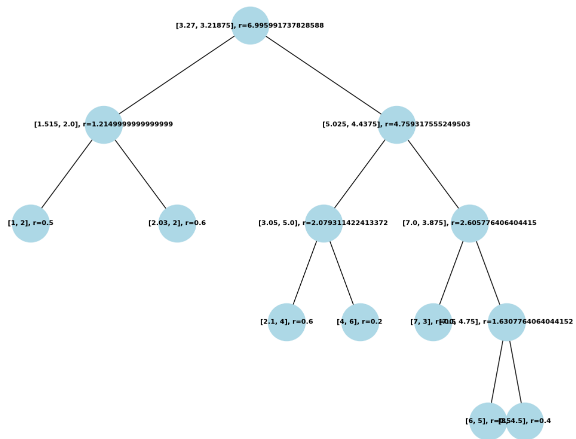
Funkcija 4: build_balltree_range(l,u): BALLTREE_NODE

```
if l = u then
    Result := create_node;
    Result.ball := bls[u];
    return Result;
else
    bdim := 0; bloc := l;
    for i = 0  to  bls.dim do
        Sort bls_on_coordinate(i, l, u);          ▷ Sortira lopte po i-toj koordinati
        Call fill_in_costs(l, u);
        if i = 0 then
            bcst := cst[l];
        end if
        for j = 1  to  u do
            if costs[j] < bcst then
                bcst := cst[j];
                bdim := i;
                bloc := j'
            end if
        end for
    end for
    Sort bls_on_best_dim(bdim, l, u)

    Result := create_node;

    Result.left := build_balltree_range(l, m);
    Result.leftt.parent := Result;

    Result.right := build_balltree_range(m + 1, u);
    Result.right.parent := Result ;

    bl := new Ball();
    bl.to_bound_balls(Result.left.ball, Result.right.ball);
    Result.ball := bl;
    return Result;
end if
```

- **Lopta A:** Centar $(1, 2)$, Radijus 0.5
- **Lopta C:** Centar $(2.1, 4)$, Radijus 0.6
- **Lopta B:** Centar $(2.03, 2)$, Radijus 0.6
- **Lopta E:** Centar $(6, 5)$, Radijus 0.5
- **Lopta D:** Centar $(4, 6)$, Radijus 0.2
- **Lopta F:** Centar $(7, 3)$, Radijus 0.5
- **Lopta G:** Centar $(8, 4.5)$, Radijus 0.4



(a) Stablo



(b) Stablo

Slika 4: Rezultat

6 Online algoritam umetanja

Onlajn algoritam je algoritam koji obrađuje podatke postepeno, deo po deo, kako pristižu, bez potrebe da ima pristup celokupnom ulazu od samog početka. To znači da donosi odluke u hodu, na osnovu trenutnog stanja i do tada pristiglih podataka.
U nastavku je prikazan jedan takav algoritam. Prilikom izgradnje stabla dozvoljava se da novi čvor bude brat bilo kog čvora u postojećem stablu. Mesto za umetanje se bira tako da izazove najmanje povećanje zapremine stabla, a tome doprinose: zapremina novog lista, zapremina novog roditeljskog cvora i povećanje zapremine svih predaka novog čvora u stablu. Prilikom pretrage za odgovarajućim bratom čvora koji se umeće u stablo, održava se pririotetni red svih kandidata, koji se porede po proširenju nadređenih čvorova. Takođe, prati se najbolje mesto umetanja do sada, kao i njegova cena umetanja. Pretraga se odmah zaustavlja ukoliko je najmanje proširenje nadređenih čvorova u redu veće od cene umetanja najboljeg čvora u tom trenutku. Nova zapremina unutrašnjih lopti koji se menja ovom operacijom sastoji se od čvora plus količine proširenja stvorenog u svim nadređenim čvorovima tog čvora. Nakon nalaženja brata A čvora koji se umeće N, formira se novi roditelj koji obuhvata A i N i postaje naslednik starom roditelju A. Nakon ove operacije, neophodno je sinhronizovati zapremine nadređenih čvorova do korena.
Odabir mesta za umetanje prema kriterijumu minimizacije novog volumena ima kao posledicu nekoliko pozitivnih osobina. Nove lopte koje su velike u poređenju sa ostatkom stabla obično se stavljaju blizu vrha, dok se male lopte koje leže unutar postojećih lopti nalaze blizu dna. Nove lopte koje su daleko od postojećih lopti takođe se nalaze blizu vrha. Na ovaj način, struktura stabla obično odražava strukturu grupisanja listova. Sledi pseudokod neophodnih funkcija za izgradnju stabla.

- `insert_at_node` (Funkcija 5) umeće čvor nl kao brata čvoru n (koji je izabran kao najbolji)
- `repair_parents` (Funkcija 6) je jednostavna, ažurira zapremine svih predaka do korena, tako što opet računa zapreminu lopte koja treba da obuhvati levo i desno dete
- `best_sibiling` (Funkcija 7) vraća pokazivač na najbolju poziciju umetanja

Procena složenosti algoritma: pronalazak najboljeg brata u stablu je složenosti $O((logN)^2)$, jer je složenost vađenja elementa iz reda sa pririoritetom $O(logN)$, a obrađuje se svakog trenutka jedan nivo stabla (ako je izbalansirano to je $logN$ nivoa). Par tražimo svakom čvoru lista, ako imamo n listova, broj čvorova u potpunom binarnom stablu je $2 \cdot n - 1$. Dakle, ukupna složenost algoritma je $O(N(logN)^2)$.

Funkcija 5: insert_at_node(nl, n:BALLTREE_NODE) : void

```
                                            ▷ Čvor nl postaje brat čvoru n
npar := create_node
nl.set_parent(None);
if tree == None then                                    ▷ Stablo je prazno
    tree := nl;
else
    npar.set_parent(n.parent);
    if n.parent == None then
        tree := npar;                                   ▷ Postavi kao koren
    else if n.parent.left == n then
        n.parent.set_left(npar);
    else
        n.parent.set_right(npar);
    end if
    npar.set_left(n);
    npar.set_right(nl);
    nl.set_parent(npar);
    n.set_parent(npar);
    nbl = create_ball;
    nbl.to_bound_balls(nl.ball, n.ball);
    npar.set_ball(nbl);
    repair_parents(npar);
end if
```

Funkcija 6: repair_parents(node: BALLTREE_NODE): void

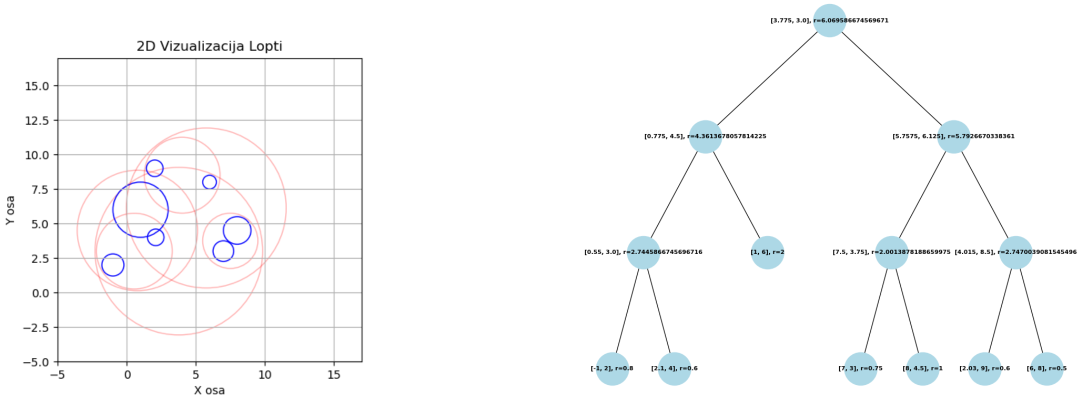
```
current := node;
while current.parent do
    current_ball := create_ball;
    current_ball.to_bound_balls(current.left.ball, current.right.ball);
    current.set_ball(current_bl);
    current := current.parent;
end while
```

Funkcija 7: best_sibling(nl:BALLTREE_NODE):BLT_NODE

```
if tree == None then
    return None                                          ▷ Stabo je prazno
else
    p_queue.clear;
    Result := tree;
    tb.to_bound_balls(tree.ball, nl.ball);
    bcost := tb.pvol;
    if not Result.leaf then
        tf.Create;
        tf.set_aexp(0);
        tf.set_ndVol(bcost);
        tf.set_node(Result);
        frng.ins(tf);
    end if
    while not p_queue.empty or done do
        tf := frng.pop;                                  ▷ Kandidat
        if tf.aexp >= bcost then
            done := true;
        else
            e := tf.aexp + tf.ndvol + tf.nd.pvol;        ▷ Novo proširenje volumena
            tb = create_ball
            tb.to_bound_balls(tf.node.left.ball, nl.ball);
            v := tb.pvol;
            if v + e < bcost then
                bcost := v + e;
                Result := tf.nd.left;
            end if
            if not tf.nd.left.leaf then
                tf2.Create;
                tf2.set_aexp(e);
                tf2.set_ndvol(v);
                tf2.set_node(tf.nd.left);
                p_queue.insert(tf2);
            end if
            tb.to_bound_balls(tf.node.right.ball, nl.ball);
            v := tb.pvol;
            if v + e < bcost then
                bcost := v + e;
                Result := tf.node.right;
            end if
            if not tf.nd.right.leaf then
                tf2.Create;
                tf2.set_aexp(e);
                tf2.set_ndvol(v);
                tf2.set_node(tf.node.right);
                p_queue.insert(tf2);
            end if
        end if
    end while
    p_queue.clear;
    return Result;
end if
```

Algoritam i klase su implementirane u pajton programskom jeziku, sledi rezultat konstrukcije.

- Lopta A: centar $(-1, 2)$, radijus 0.8
- Lopta C: centar $(2.1, 4)$, radijus 0.6
- Lopta B: centar $(2.03, 9)$, radijus 0.6
- Lopta E: centar $(6, 8)$, radijus 0.5
- Lopta D: centar $(1, 6)$, radijus 2
- Lopta F: centar $(7, 3)$, radijus 0.75
- Lopta G: centar $(8, 4.5)$, radijus 1



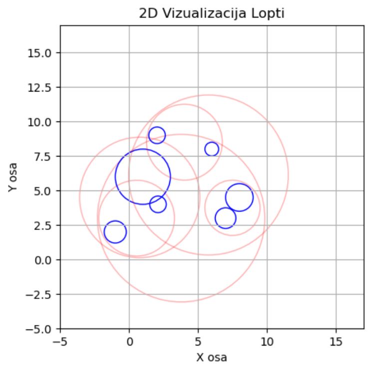
(a) Stablo

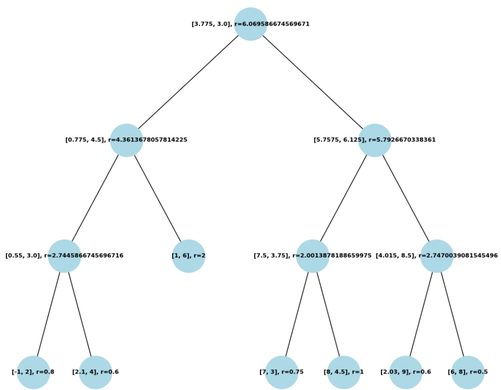
(b) Stablo

Slika 5: Rezultat

7 Jeftiniji on-line algoritam umetanja

U ovom algoritmu se ne koristi pririoritetni red, već se održava i dalje se istražuje samo jeftiniji od dva podređena čvora u bilo kom trenutku. Ponovo je pretraga prekinuta kada proširenje pretka premaši najbolju ukupnu vrednost proširenja.
Algoritam je pokrenut za isti ulaz kao i za obični on-line algoritam i rezultat rada je isti.





Procena složenosti algoritma: pronalazak najboljeg brata u stablu je složenosti $O(logN)$, jer svakim prolaskom kroz petlju obrađujemo jedan nivo stabla (ako je izbalansirano to je $logN$ nivoa). Par tražimo svakom čvoru lista, ako imamo n listova, broj čvorova u potpunom binarnom stablu je $2 \cdot n - 1$. Dakle, ukupna složenost algoritma je $O(NlogN)$.

Funkcija 8: cheap_best_sibling(nt: BALLLTREE_NODE): BALLTREE_NODE

```
if tree == None then
    return None                                          ▷ Stablo je prazno
else
    Result := tree
    tb := create_ball                                    ▷ Globalna test lopta (zapremina)
    tb.to_bound_balls(tree.ball, nl.ball)
    wv := tb.pvol
    bcost := wv
    ae := 0                                              ▷ Početni akumulirano proširenje
    nd := tree
    done := False
    while (not nd.leaf) and (not done) do
        ae := ae + wv - nd.pvol                          ▷ Koriguj za oba deteta
        if ae ≥ bcost then
            done := True                                 ▷ Ne može biti bolje
        else
            if nd.left != None then
                tb.to_bound_balls(nd.left.ball, nl.ball)
                lv := tb.pvol
            else
                lv := ∞
            end if
            if nd.right != None then
                tb.to_bound_balls(nd.right.ball, nl.ball)
                rv := tb.pvol
            else
                rv := ∞
            end if
            if ae + lv <= bcost then
                Result := nd.left
                bcost := ae + lv
            end if
            if ae + rv <= bcost then
                Result := nd.right
                bcost := ae + rv
            end if
            if lv - nd.left.pvol <= rv - nd.right.pvol then
                wv := lv
                nd := nd.left
            else
                wv := rv
                nd := nd.right
            end if
        end if
    end while
    return Result
end if
```

8 Konstrukcija stabla odozdo naviše

Konstrukcija stabla odozdo naviše (eng. *bottom up construction*) iterativno pronalazi dve lopte iz skupa lopti koje treba ubaciti u stablo tako da njihova granična lopta koja ih obuhvata ima najmanji volumen, pravi ih braćom i ubacuje roditeljsku loptu nazad u skup. Najjednostavniji pristup algoritmom grube sile (eng. *brute force*) održava trenutne kandidate u nizu, i u svakoj iteraciji proverava volumen granične lopte svakog para kako bi našla najbolji. Direktna implementacija ovog pristupa zahteva N prolaza, od kojih je većina veličine N^2 , te je složenost algoritma koristeći naivno rešenje $O(N^3)$.

U poboljšanom pristupu svaki čvor prati drugi čvor tako da zapremina njihove zajedničke granične lopte bude minimalna, kao i samu zapreminu te granične lopte. Tada bi čvor sa minimalnim zapamćenim troškom i njegov zapamćeni par bio najbolji par za spajanje u tom trenutku. Ovo se može postići korišćenjem reda sa prtitioritetom, gde je kriterijum za njihovo poređenje granična lopta koja obuhvata čvor i njegovog najboljeg partnera. Većina lopti zadržava svog para, i kada je partner sparen negde drugde, najbolji trošak može samo da poraste. Pri tome, čuvaju se nezavršeni delovi stabla i održavaju se jednostavim operacijama pretrage, umetanja i brisanja.

Na početku skup čvorova koje tek treba ubaciti u stablo sadrži samo listove, tj. podatke. Za svaki list se jednim prolazom kroz skup određuje koji je njegov par iz skupa kao i zapremina njihove granične lopte, nakon čega se čvor i cena ubacuju u red. Kako algoritam napreduje, uklanja se najbolji čvor iz reda sa priorioritetom, i ako već nije uparen, ponovo se računa njegovog najboljeg partnera koristeći skup za umetanje. Ako je ponovo izračunati trošak manji od vrha reda, tada se uklanja čvor i njegov partnertner iz skupa za umetanje, zatim se formira roditeljski čvor iznad njih, izračuna najbolji partner za roditeljski čvor i ubacuje se roditelj u skup za umetanje i red sa priorioritetom. Kada u skupu za umetanje ostane samo jedan čvor, konstrukcija je završena (koren). U nastavku su prikazane funkcije za određivanje najboljeg para i za njihovo spajanje. pq je prioritetni red čvorova koji čekaju, a promenljive b1 i b2 će sadržati najbolji par za spajanje. blt predstavlja skup čvorova koje treba ubaciti u stablo, a has_leaf testira da li blt ima određeni čvor kao list, odnodsno da li je u skupu.

Funkcija 9: find_best_pair(): void

```
b1.forget; b2.forget;                                         ▷ Resetovanje b1 i b2
done := False;
while not done do
    if pq.empty then
        done := True
    else
        btm := pq.pop
        if blt.has_leaf(btm) then
            blt.remove(btm)
            btm.set_bvol(blt.best_vol_to_ball(btm.ball))
            if pq.empty or btm.bvol ≤ pq.top.bvol then
                done := True
            else
                pq.insert(btm); blt.cheap_insert(btm)          ▷ Pokušati ponovo
            end if
        end if
    end if
end while
if not btm.Void then
    b1 := btm; b2 := blt.best_node_to_ball(b1.ball)
    blt.remove(b2)                                           ▷ Ukloni b2 iz blt
end if
```

Funkcija 10: merge_best_pair(b1,b1 : BALLTREE_NODE) : void

if $(b1! = None)$ **and** $(b2! = None)$ **then**

  bn = create_node

  bn.set_left($b1$); $bn.left$.set_parent(bn)

  bn.set_right($b2$); $bn.right$.set_parent(bn)

  bl.create_ball($blt.dim$); bl.to_bound_balls($bn.left.ball, bn.right.ball$)

  bn.set_ball(bl)

  $b1$.create_node

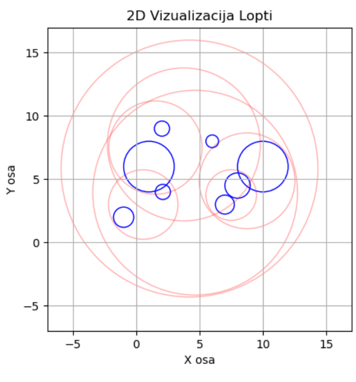
  $b1$.set_node(bn); $b1$.set_ball(bl)

  $b1$.set_bvol(blt.best_vol_to_ball(bl))

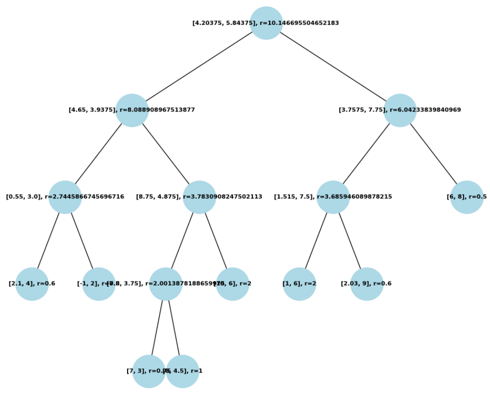
  $pq\_insert(b1)$; blt.cheap_insert($b1$)

  $b1 := None$; $b2 := None$

end if



(a) Stablo

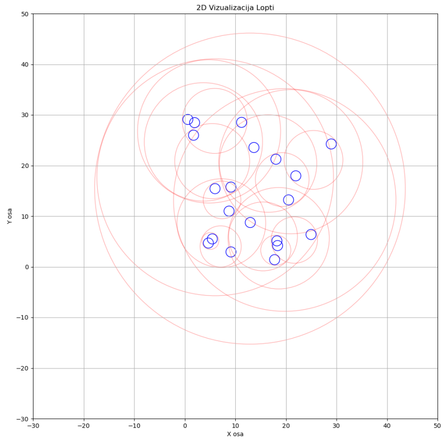


(b) Stablo

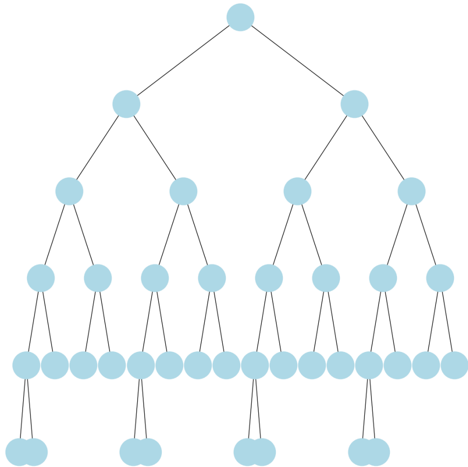
Slika 6: Rezultat

9 Poređenje algoritma

Algoritmi koji su do sada analizirani su pokrenuti na malo većem skupu lopti. Slede njihovi rezultati.

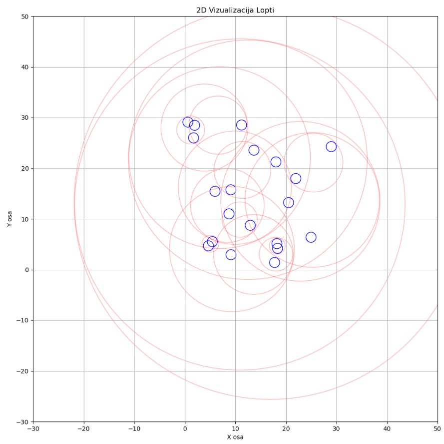


(a) Stablo

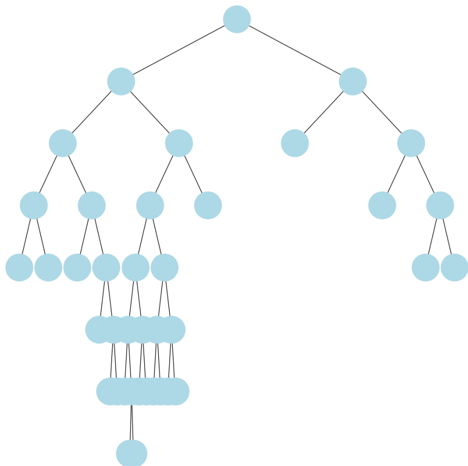


(b) Stablo

Slika 7: Kd

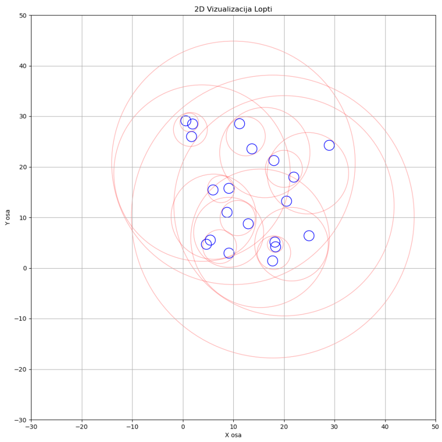


(a) Stablo

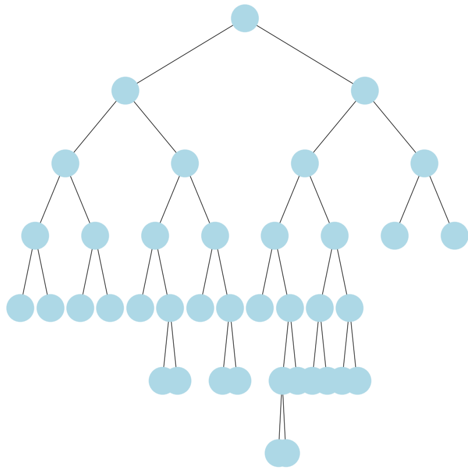


(b) Stablo

Slika 8: Top down

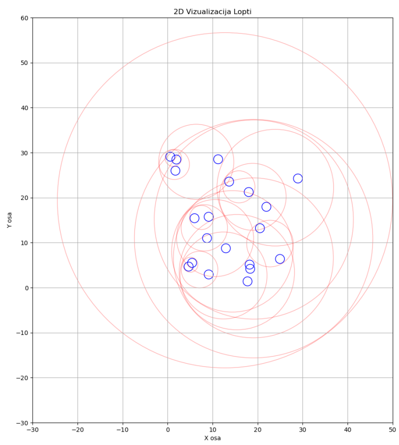


(a) Stablo

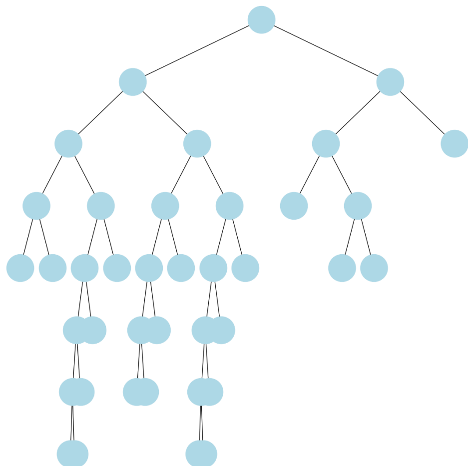


(b) Stablo

Slika 9: Online i cheap online



(a) Stablo



(b) Stablo

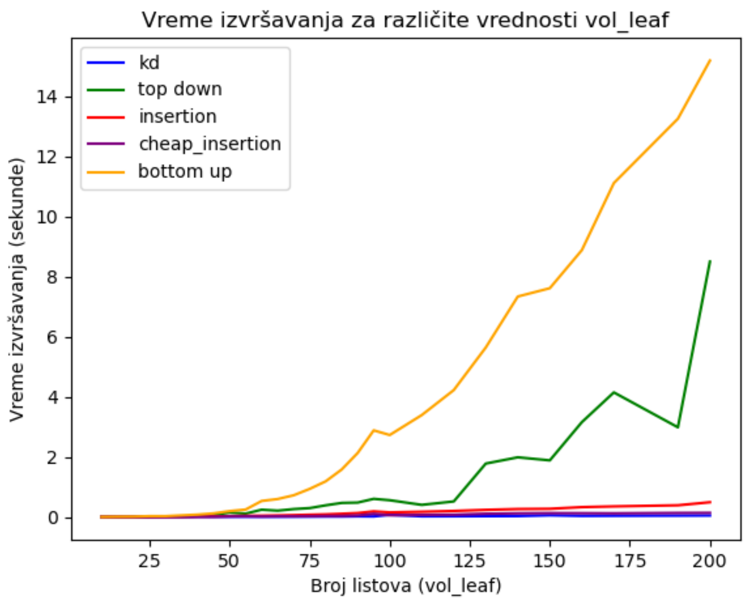
Slika 10: Bottom up

10 Zaključak

K-d algoritam nasumično deli tačke na pola, ne vodeći računa o hijerarhijskoj strukturi. Ovo mu omogućava da proizvede savršeno izbalansirano stablo (Slika 7 (b)), ali po cenu zanemarivanja strukture u podacima.
Dva algoritma online umetanja su proizvela potpuno isto stablo. Čini se da su rano doneli odluku koja je dovela do toga da konačno stablo ima veoma veliku loptu blizu korena. Ovo je tipičan rezultat korišćenja onlajn algoritma.
Odozgo naniže i odozdo naviše pristupi su pronašli veoma slična stabla koja su suštinski istog kvaliteta.

U praksi se pokazalo, ako su podaci glatki i ima ih mnogo, k-d pristup je brz i jednostavan. Ako su podaci grupisani, retki ili imaju dodatnu strukturu, k-d pristup obično nije dobar izbor. Odozdo naviše pristup u svim slučajevima odlično pronalazi bilo kakvu strukturu i, osim troškova konstrukcije, predstavlja prvi izbor. U situacijama gde je umetanje u realnom vremenu neophodno, običan online algoritam umetanja je približan odoydo naviše algoritmu po kvalitetu. Njegov jeftiniji pristup se značajno lošije pokazuje, ali dovodi do vremena konstrukcije koja su bliska k-d algoritmu.

Na slici 11 je prikazano vreme izvršavanja za svaki od opisanih algoritama, sa ciljem omogućavanja njihovog međusobnog upoređivanja.



Slika 11: Vreme izvršavanja algoritama u zavisnosti od broja listova

11 Literatura

[1] Five Balltree Construction Algorithms - Stephen M. Omohundro, 1989., International computer science institute, California
[2] geeksforgeeks
[3] python3 dokumentacija
[4] Machine Learning Lecture 28 Ball Trees - Kilian Weinberger, 2018.