Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Fakulteta za matematiko in fiziko

Leonard Štefančič

Gradnja Vietoris-Ripsovega simplicialnega kompleksa

diplomsko delo
na interdisciplinarnem univerzitetnem študiju

Mentorica: prof. dr. Neža Mramor-Kosta

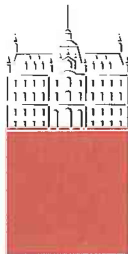
Ljubljana, 2013

Rezultati diplomskega dela so intelektualna lastnina Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko, Fakultete za matematiko in fiziko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si



Št. naloge: 00047/2013
Datum: 02.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko ter Fakulteta za matematiko in fiziko izdaja naslednjo nalogo:

Kandidat: **LEONARD ŠTEFANČIČ**

Naslov: **GRADNJA VIETORIS-RIPSOVEGA SIMPLICIALNEGA KOMPLEKSA**
CONSTRUCTION OF THE VIETORIS-RIPS SIMPLICIAL COMPLEX

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V diplomskem delu opišite pojem simplicialnega kompleksa in pa Vietoris-Ripsov simplicialni kompleks, ki se v računalništvu uporablja za analizo podatkov in rekonstrukcijo objektov iz vzorca točk.

Opišite dvostopenjski algoritem za izgradnjo Vietoris-Ripsovega kompleksa. V prvem koraku algoritem konstruira graf, ki povezuje bližnje točke iz vzorca, v drugem koraku pa graf razširi do simplicialnega kompleksa. Dokažite pravilnost algoritma in prikažite delovanje na preprostem primeru.

Mentor:
prof. dr. Nežka Mramor Kosta

Dekan Fakultete za računalništvo in informatiko:
prof. dr. Nikolaj Zimic

Dekan Fakultete za matematiko in fiziko:

akad. prof. dr. Franc Forstnerič

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Leonard Štefančič, z vpisno številko **63050216**, sem avtor diplomskega dela z naslovom:

Gradnja Vietoris-Ripsovega simplicialnega kompleksa

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Neže Mramor-Kosta,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 5. junija 2013 Podpis avtorja:

Zahvala

Iskreno se zahvaljujem mentorici prof. dr. Neži Mramor Kosta za izkazano pomoč pri izdelavi diplomske naloge.

Kazalo

Povzetek

Abstract

1 Uvod 1

2 Predznanje 3
- 2.1 Simplicialni kompleksi 4
  - 2.1.1 Geometrični simplicialni kompleksi 5
  - 2.1.2 Abstraktni simplicialni kompleksi 8
- 2.2 Vietoris-Ripsov kompleks 9

3 Pristop 10

4 Računanje grafa sosednosti 12
- 4.1 kd-drevo 12

5 Algoritmi Vietoris-Ripsove razširitve 20
- 5.1 Induktivni algoritem 20
- 5.2 Inkrementalni algoritem 22
- 5.3 Maksimalni algoritem 23
- 5.4 Utežna funkcija 27

6 Implementacija kode 28

7 Rezultat 29

8 Zaključek 33

A Priloga: Implementacija programske kode 34
- A.1 Implementacija kode za računanje prve faze 36
- A.2 Implementacija kode za računanje druge faze 40

Povzetek

V diplomski nalogi je opisan simplicialni kompleks kot način za predstavitev topoloških prostorov in pa Vietoris-Ripsov kompleks kot primer simplicialnega kompleksa, ki se v računalništvu uporablja za opis in analizo podatkov. Opisan je dvostopenjski algoritem za izgradnjo Vietoris-Ripsovega kompleksa. V prvem koraku se iz množice podatkov konstruira graf, ki povezuje bližnje točke, v drugem sledi razširitev do simplicialnega kompleksa. Obravnavani so različni algoritmi (induktivni, inkrementalni in maksimalni) in dokazane njihove pravilnosti. Na enem podatkovnem vzorcu je opravljen preizkus vseh algoritmov ter opisana njihova učinkovitost.

Ključne besede:

k-simpleks, simplicialni kompleks, Vietoris-Ripsov kompleks, graf sosednosti, kompleks klik

Abstract

In the thesis simplicial complexes as a means for representing topological spaces are presented. In particular, the Vietoris-Rips complex, which is used in computer science for data reconstruction and analysis, is described. A two-phase algorithm for constructing the Vietoris-Rips complex on a given data set is described. In the first phase, the algorithms constructs the neighborhood graph on a set of points, and in the second phase the graph is extended to a simplicial complex. Three approaches, inductive, incremental and maximal, for the second phase are described and proofs of their correctness are given. The algorithms are tested on an example, and their efficiency is analyzed.

Keywords:

k-simplex, simplicial complex, Vietoris-Rips complex, neighborhood graph, clique complex

1 Uvod

Znanstveno področje diplomske naloge je področje Računske topologije, ki je relativno novo področje vmes med računalništvom in matematiko.

Topologija je veja matematike, ki se ukvarja z raznimi množicami točk v prostoru. Zanimajo jo take lastnosti množic, ki se ohranjajo pri upogibanju, raztezanju ali krčenju (brez raztrganin ali lepljenja). Tako imajo vse krivulje, ki so dobljene iz neke dane krivulje z upogibanjem, raztezanjem ali krčenjem, enake lastnosti. Na primer, elipsa, rob trikotnika ali kvadrata ali katerakoli druga enostavno sklenjena krivulja je s stališča topologije enaka krožnici. Topologije, torej, za razliko od sorodne geometrije ne zanimajo dolžine, ploščine, površine ali velikosti kotov; te lahko še poljubno spreminjamo. Pri topologiji gre bolj za kvalitativen opis pomembnih lastosti objektov in njihovih oblik, na primer število ločenih kosov, število in oblike lukenj in tunelov.

V diplomski nalogi obravnavamo simplicialne komplekse. Recimo, da rešujemo grafično uganko s ciljem, da povežemo množico točk tako, da postane neka risba. Pri računalniški analizi podatkov rešujemo podoben problem. Iz množice točk sestavimo matematični model originala. Najpogostejši matematični model, ki se uporablja, je simplicialni kompleks. V diplomski nalogi je opisan algoritem za konstrukcijo Vietoris-Ripsovega kompleksa. To je simplicialni kompleks, ki ga zgradimo iz množice posameznih točk, za katere poznamo samo njihovo medsebojno razdaljo, samih koordinat pa morda ne. Vietoris-Ripsov kompleks se uporablja na primer za modeliranje brezžičnih senzorskih omrežij, kjer so točke posamezni senzorji in so znani samo podatki o njihovi medsebojni povezanosti, njihovi položaji pa ne [8,16]. Konstrukcija Vietoris-Ripsovega kompleksa je relativno preprost problem, še posebej v primerjavi z nekaterimi drugimi simplicialnimi kompleksi, kot so alpha kompleksi [9], Delaunayev kompleksi [6], "flow" kompleksi [10], "witness" kompleksi [7] in še cela vrsta drugih. Na voljo je kar nekaj implementacij, ki so vključene tudi v nekatere razširjene programske pakete za topološko analizo podatkov, kot so Plex [17], JPLEX [18] in Dionysus [13].

V drugem poglavju je opisan abstraktni simplicialni Vietoris-Ripsov kompleks. V tretjem poglavju je opisana dvofazna konstrukcija Vietoris-Ripsovega kompleksa, v četrtem pa računanje grafa sosednosti ter ena izmed metod,

kd-drevo. V petem poglavju so opisani različni algoritmi Vietoris-Ripsove razširitve: inkrementalni, induktivni in maksimalni. V šestem poglavju je podan pregled uporabljenih knjižnic v programskem jeziku C++. V zadnjem poglavju smo izbrali en vhodni primer (množica točk) za konstrukcijo Vietoris-Ripsovega kompleksa ter tri različna merila. Na podlagi teh meril in množice točk smo preverili delovanje treh različnih algoritmov: inkrementalnega, induktivnega in maksimalnega. Izmerili smo čase izvajanja ter jih primerjali. Na koncu prikažemo rezultat Vietoris-Ripsovega kompleksa pri treh različnih merilih.

V prilogi A je zbrana programska koda vsa razvita.

2 Predznanje

V tem poglavju so zbrane osnovne matematične definicije o topologiji in simplicialnem kompleksu. Več o njih najdemo na primer v literaturi [15,22].

Definicija 2.1: Topologija

Topologija na množici X je podmnožica $T \subseteq 2^x$, za katero velja

(a) če $S_1, S_2 \in T$, potem $S_1 \cap S_2 \in T$

(b) če $\{S_J \mid j \in J\} \subseteq T$, potem $\bigcup_{j \in J} S_j \in T$, kjer je indeksna množica J poljubna

(c) $\emptyset, X \in T$

Topološki prostor je par (X, T), v katerem je X množica in T topologija na njej. Topološki prostor (X, T) običajno na kratko označimo kar z X. Elemente topologije T na X imenujemo *odprte množice* topološkega prostora X. Elemente topološkega prostora X *točke*.

Podmnožica A topološkega prostora X je *okolica* točke $x \in A$, če obstaja takšna odprta podmnožica U prostora X, da velja $x \in U \subset A$. Podmnožica A topološkega prostora X je okolica neke podmnožice $B \subset A$, če, in samo če, je okolica vsake točke iz B.

Definicija 2.2: Metrika

Metrika na množici X je takšna funkcija $d : X \times X \to [0, \infty)$, da za poljubne točke $x, y, z \in X$ velja

(a) $\mathrm{d}(x, y) = \mathrm{d}(y, x)$

(b) $\mathrm{d}(x, z) \leq \mathrm{d}(x, y) + \mathrm{d}(\mathrm{y}, \mathrm{z})$

(c) $\mathrm{d}(x, x) = 0$

(d) če $\mathrm{x} \neq \mathrm{y}$, potem $\mathrm{d}(x, y) > 0$

Primer 2.3: Evklidska metrika

Evklidska metrika ali evklidska razdalja v prostoru $\mathbb{R}^d$ je podana s predpisom $\mathrm{d}(x, y) = \|x - y\|$.

Definicija 2.4: Krogla

Množico $K_r(x) = \{y, d(x, y) < r\}$ imenujemo *odprta krogla* okrog točke x s polmerov r.

Metrika in zveza s topologijo:

Metrika na množici X določa topologijo na tej množici. Odprte množice v tej topologiji so vse možne unije odprtih krogel v dani metriki.

2.1 Simplicialni kompleksi

Natančna predstavitev površin v nekem računalniškem sistemu je na splošno omejena z omejitvijo pomnilnika le-tega. Zato omenjene površine vzorčimo in predstavimo ploskve s pomočjo triangulacije. Triangulacija je simplicialni kompleks, kombinatorični prostor, ki lahko ponazori obliko prostora, neodvisno od njegove geometrije.

2.1.1 Geometrični simplicialni kompleksi

Začnemo z definicijo simplicialnega kompleksa, ki povezuje geometrijo in topologijo. Kombinacija nam dovoli, da predstavimo območje prostora z nekaj točkami.

Afine in konveksne množice

Naj bo V realen vektorski prostor. Podmnožica $A \subset V$ je *afina* , če za poljubna različna vektorja u in v iz A premica $\{(1-t)u + tv \mid t \in \mathbb{R}\}$ cela leži znotraj množice A. *Afina ogrinjača* $\mathrm{Aff}(S)$ poljubne neprazne podmnožice S prostora V je presek vseh afinih podmnožic prostora V, ki vsebujejo vse vektorje iz S. Afina ogrinjača neprazne podmnožice $S \subset V$ je torej najmanjša afina podmnožica prostora V, ki vsebuje vse vektorje iz S. Sestavljena je iz vseh *afinih kombinacij* vektorjev iz S,

$$\mathrm{Aff}(S) = \left\{ \sum_{t=0}^{k} t_i v_i \mid k \in \mathbb{N}_0, v_0, ..., v_k \in S, t_0, ..., t_k \in \mathbb{R}, \sum_{i=1}^{k} t_i = 1 \right\}$$

Vektorji $v_0, ..., v_n \in V$ so *afino neodvisni*, če so vektorji $v_1 - v_0, ..., v_n - v_0$ linearno neodvisni. Ekvivalentno, vektorji $v_0, ..., v_n \in V$ so afino neodvisni, če, in samo če, poljuben vektor $v \in \mathrm{Aff}(\{v_0, ..., v_n\})$ lahko zapišemo kot afino kombinacijo vektorjev $v_0, ..., v_n$ na en sam način.

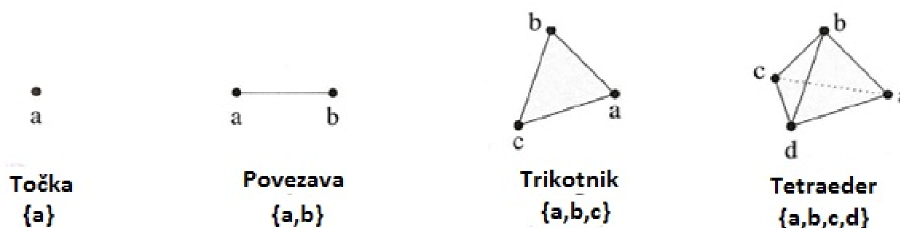
Podmnožica $C \subset V$ je *konveksna*, če za poljubna različna vektorja u in v iz C daljica $\{(1-t)u + tv \mid t \in [0,1]\}$ cela leži znotraj C. *Konveksna ogrinjača* $\mathrm{Conv}(S)$ poljubne neprazne podmnožice S prostora V je presek vseh konveksnih podmnožic prostora V, ki vsebujejo vse vektorje iz S. Konveksna ogrinjača neprazne podmnožice $S \subset V$ je najmanjša konveksna podmnožica prostora V, ki vsebuje vse vektorje iz S, in je sestavljena iz vseh *konveksnih kombinacij* vektorjev iz S,

$$\mathrm{Conv}(S) = \left\{ \sum_{t=0}^{k} t_i v_i \mid k \in \mathbb{N}_0, v_0, ..., v_k \in S, t_0, ..., t_k \in [0,1], \sum_{i=1}^{k} t_i = 1 \right\}$$

Definicija 2.5: k-simpleks

k-simpleks je konveksna ogrinjača $k+1$ afino neodvisnih točk $S = \{v_0, v_1, ..., v_k\}$ Točke v S so *oglišča* simpleksa.

k-simpleks σ je k-dimenzionalna podmnožica prostora $\mathbb{R}^d, d \geq k$, dimσ=k. Pokažimo nižje dimenzionalne simplekse z njihovimi imeni v sliki 2.1.

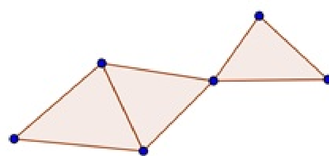


Slika 1: k-simpleks $0 \leq k \leq 3$

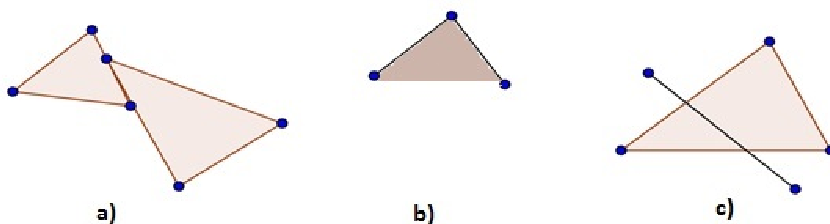
Definicija 2.6: lice, kolice

Naj bo σ k-simpleks, definiran z množico oglišč $S = \{v_0, v_1, ..., v_k\}$. Simpleks τ, definiran s podmnožico $T \subset S$, je *lice* simpleksa σ, σ pa je *kolice* τ. Razmerje je označeno z $\sigma \geq \tau$ in $\tau \leq \sigma$.

k-simpleks ima $\binom{k+1}{l+1}$ lic dimenzije l in $\sum_{l=-1}^{k} \binom{k+1}{l+1} = 2^{k+1}$ lic v celoti, pri čemer štejemo prazno množico kot lice simpleksa dimenzije -1. Torej je simpleks velik, toda zelo enoličen in enostaven kombinatoričen objekt. Za ponazoritev opazovanega prostora več simpleksov sestavimo v simplicialni kompleks.



Slika 2: Srednji trikotnik deli rob z levim trikotnikom in deli oglišče z desnim trikotnikom.



Slika 3: V b), trikotniku manjka rob. Simpleksa na a) in c) se sekata mimo skupnih simpleksov.

Definicija 2.7: Simplicialni kompleks

Simplicialni kompleks K je končna množica simpleksov, za katero velja:

(a) $\sigma \in K, \tau \leq \sigma \Rightarrow \tau \in K$

(b) $\sigma, \sigma' \in K \Rightarrow \sigma \cap \sigma' \leq \sigma, \sigma'$

Dimenzija K je $\dim K = \max\{\dim \sigma \mid \sigma \in K\}$. Oglišča K so 0-simpleksi v K. Simpleks je maksimalen, če nima pravega kolica v K.

Torej je simplicialni kompleks družina simpleksov, ki se dobro ujemajo, kot je prikazano pri sliki 2, v nasprotju s primeri na sliki 3.

2.1.2 Abstraktni simplicialni kompleksi

Definicija simpleksa uporablja geometrijo in geometrijski simplicialni kompleksi so geometrijski objekti z geometrijskimi lastnsotmi. Lahko pa definiramo simplicialne komplekse brez uporabe geometrije. V nadaljevanju bomo predstavili definicijo simplicialnega kompleksa, ki jasno loči topologijo in geometrijo. Takšna abstraktna definicija je predvsem zanimiva za naš problem.

Definicija 2.8: Abstraktni simplicialni kompleks

Abstraktni simplicialni kompleks je takšna družina K končnih nepraznih množic, da za poljubno množico $A \in K$ vsaka neprazna podmnožica B množice A pripada K.

Elemente abstraktnega simplicialnega kompleksa K imenujemo *abstraktni simpleksi*. *Dimenzija* simpleksa $A \in K$ je število $\dim A = \#A - 1$, pri čemer je $\#A$ število elementov množice $A \subset K$. *Lica* simpleksa A so neprazne podmnožice množice A. Lice simpleksa A je pravo, če ni enako množici A. Za poljuben $n \in \mathbb{N}_0$ je družina $K^{(n)} = \{A \in K \mid \dim A \leq n\}$ abstraktni simplicialni kompleks, ki ga imenujemno *n-skelet* abstraktnega simplicialnega kompleksa K.

Podkompleks simpliclanega kompleksa K je podmnožica $L \subset K$, ki je tudi sama simplcialni kompleks.

2.2 Vietoris-Ripsov kompleks

Vietoris-Ripsov kompleks je abstraktni simplicialni kompleks, ki ga lahko definiramo na poljubni končni množici S v nekem metričnem prostoru. Simplekse Vietoris-Ripsovega kompleksa tvorijo tiste podmnožice množice S, ki imajo premer manjši od nekega izbranega merila ϵ [20].

Vietoris-Ripsovi kompleksi se v računalništvu uporabljajo na primer za rekonstrukcije objektov [5], odkrivanje značilk v podatkih (na primer v digitalnih slikah) [3] in za modeliranje brezžičnih senzorskih omrežij [8,16].

Definicija 2.9: Vietoris-Ripsov kompleks

Predpostavimo, da imamo končno množico točk $S \subseteq \mathbb{R}^d$. *Vietoris-Ripsov kompleks* $\nu_\epsilon(S)$ množice S v merilu ϵ je

$$\nu_\epsilon(S) = \{\sigma \subseteq S \mid \mathrm{d}(u, v) \leq \epsilon, \forall u \neq v \in \sigma\}, \tag{1}$$

kjer je d neka metrika (na primer evklidska) na prostoru $\mathbb{R}^d$.

Z drugimi besedami vsak simpleks σ v $\nu_\epsilon(S)$ ima oglišča, ki so paroma oddaljena za razdaljo največ ϵ.

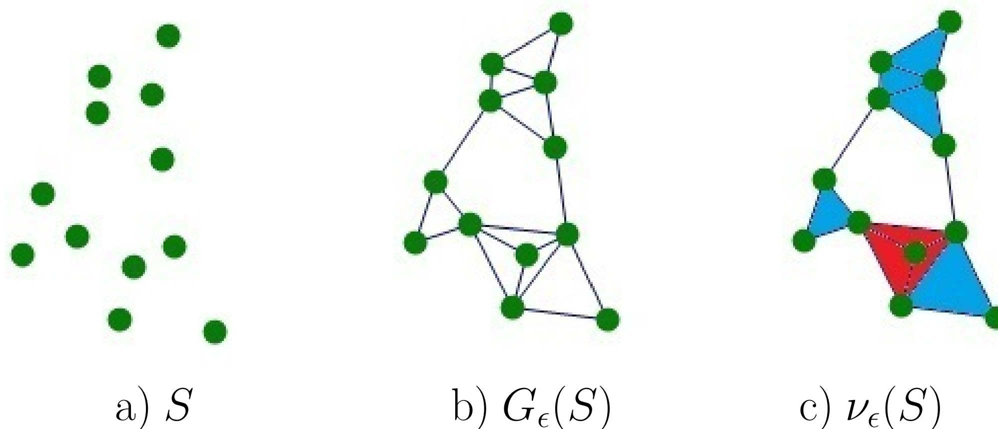
Vietoris-Ripsov kompleks lahko izračunamo pri različnih merilih ϵ in dobljene komplekse primerjamo med seboj. Pri uporabi Vietoris-Ripsovega kompleksa za rekonstrukcije iz podatkov je težko določiti pravo merilo ε, kjer bo rekonstrukcija najboljša možna. Ta problem lahko rešimo tako, da izračunamo kompleks pri nekem izbranem maksimalnem merilu $\hat{\epsilon}$ in na simpleksih dobljenega kompleksa definiramo *utežno funkcijo* $f : \nu_{\hat{\epsilon}}(S) \to \mathbb{R}$.

Definicija 2.10: Uteženi kompleks

Uteženi kompleks je par (K, f), kjer je K simplicialni kompleks in $f : K \to \mathbb{R}$ je funkcija, ki vsakemu smipleksu priredi neko utež. Naš cilj je računati uteženi Vietoris-Ripsov kompleks.

3 Pristop

V tem poglavju opisujemo pristop h konstrukciji Vietoris-Ripsovega kompleksa, povzet po članku [21]. Postopek je razdeljen na dva koraka, prvi je geometrijski in drugi topološki.



Slika 4: *a*) Vhodna množica točk
b) Graf sosednosti, dobljen po 1. koraku
c) Razširitev grafa v Vietoris-Ripsov kompleks

Definicija 3.1: Vietoris-Ripsov graf sosednosti

Naj bo $S \subset \mathbb{R}^d$ končna množica točk in $\epsilon > 0$ merilo. Vietoris-Ripsov graf sosednosti je utežen graf $(G_\epsilon(S), w)$, kjer je $G_\epsilon(S) = (S, E_\epsilon(S))$ neusmerjen graf z oglišči S in povezavami

$$E_\epsilon(S) = \{\{u, v\} \mid \mathrm{d}(u, v) \leq \epsilon, u \neq v \in S\}, \tag{2}$$

in w utežna funkcija, definirana na povezavah s predpisom

$$w(\{u, v\}) = \mathrm{d}(u, v) \text{ za vsak } \{u, v\} \in E_\epsilon(S). \tag{3}$$

Slika 4b) pokaže Vietoris-Ripsov graf sosednosti z 20 povezavami, kjer je w dolžina roba. Poljuben graf sosednosti lahko razširimo do Vietoris-Ripsovega kompleksa.

Definicija 3.2: Vietoris-Ripsova razširitev

Vietoris-Ripsov graf sosednosti razširimo do uteženega Vietoris-Ripsovega kompleksa tako, da simpleks σ dodamo v $\nu(G)$, če so vsi njegovi robovi povezave v grafu sosednosti.

Utežno funkcijo razširimo na vse simplekse Vietoris-Ripsovega kompleksa tako, da je $\omega(\sigma) = 0$, če je σ oglišče, sicer pa je $\omega(\sigma)$ enaka dolžini najdaljšega roba v simpleksu σ.

V naslednjem poglavju si bomo ogledali nekaj metod za računanje grafa sosednosti, v 5. poglavju pa tri algoritme za konstrukcijo Vietoris-Ripsove razširitve.

4 Računanje grafa sosednosti

Poznamo koordinate množice točk, ki se domnevno nahajajo na površini objekta. Naš cilj je poiskati graf sosednosti. Za realizacijo tega (delnega) cilja so potrebni naslednji koraki:

- vhodnim točkam priredimo množico vozlišč;
- vsem parom vhodnih točk, katerih medsebojna razdalja je manjša kot izbrana referenčna razdalja ϵ, priredimo množico povezav;
- vhodne točke skupaj z ustreznimi povezavami tvorijo graf sosednosti.

Možni pristopi za iskanje vseh parov točk, ki so si oddaljeni manj kot ϵ:

- Naivni pristop (angl. Brute-force search):
poiščemo razdalje med vsemi pari točk in nekemu paru priredimo povezavo v grafu sosednosti, če je izračunana razdalja manjša od podanega ϵ

- Reševanje problema ϵ-najbližjih sosedov. To je problem, kjer za dano poizvedbeno točko iščemo vse točke, ki so oddaljene za manj kot ϵ.
Možni pristopi za iskanje ϵ-najbližjih sosedov so: kd-drevo, r-drevo, bdd-drevo("balanced box-decomposition"), itd.

Oglejmo si natančneje kd-drevesa.

4.1 kd-drevo

Drevo je povezan graf brez ciklov. Drevo ima eno korensko vozlišče ("root node"). Vsa vozlišča razen korenskega imajo natanko enega starša, ki je tudi vozlišče drevesa in poljubno število otrok. Vozlišča brez otrok so končna vozlišča ali listi drevesa ("leaf nodes").

Kd-drevo je binarno drevo, kjer vsako nekončno vozlišče razdeli prostor na dva polprostora. Vsako tako vozlišče definira hiperravnino in razdeli objekte na tiste, ki so na eni strani hiperravnine, in na druge, ki so na drugi strani.

Delilne ravnine lahko izberemo na veliko različnih načinov, tako da dobimo mnogo različnih dreves. Običajno se držimo omejitev pri gradnji kd-drevesa: Pri prvi delitvi izberemo delitveno ravnino vzdolž x osi. Pri naslednjem nivoju izberemo delilno ravnino vzdolž y osi, če smo v prostoru

izberemo naslednjo delilno ravnino vzdolž z z osi, potem pa začnemo spet od začetka. [11]

Zgled psevdokode za konstrukcijo kd-drevesa

kd-drevo(množica točk M, *globina*) {

1. **če** M vsebuje eno točko { vrni končno vozlišče }

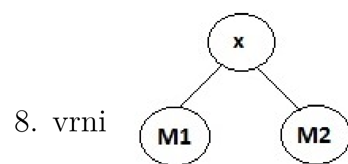
   sicer {

2. os=*globina* mod k
   (os je ostanek pri deljenju $globina/k$. k je dimenzija prostora. *globina* je trenutno število nivoja drevesa)

3. izberimo točko $x \in M$, ki je mediana v M glede koordinate, ki jo trenutna os določa.

4. množico M razdelimo v dve novi množici $M1$ in $M2$.
   (Nova množica $M1$ vsebuje vse tiste točke iz M, ki so manjše ali enako kot mediana x in $M2$ vsebuje vse tiste točke iz M, ki so večje kot x.)

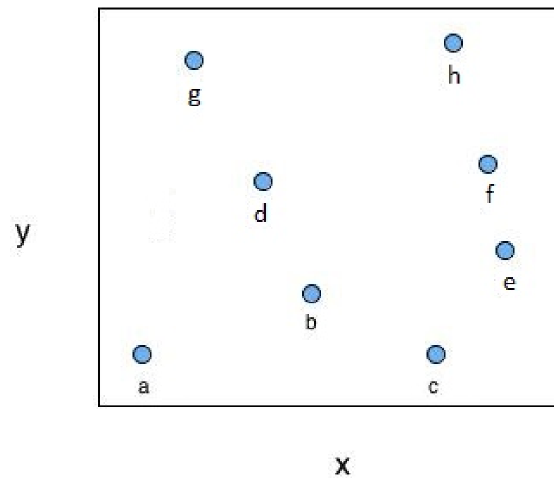
5. konstruirajmo novo vozlišče z vrednostjo x.
   Za konstrukcijo poddrevesa kličemo rekurzivno funkcijo kd-drevo z novim argumentom:
6. za levo poddrevo $M1$ kličemo funkcijo kd-drevo(M1, globina+1)
7. za desno poddrevo $M2$ kličemo funkcijo kd-drevo(M2,globina+1)

8. vrni 

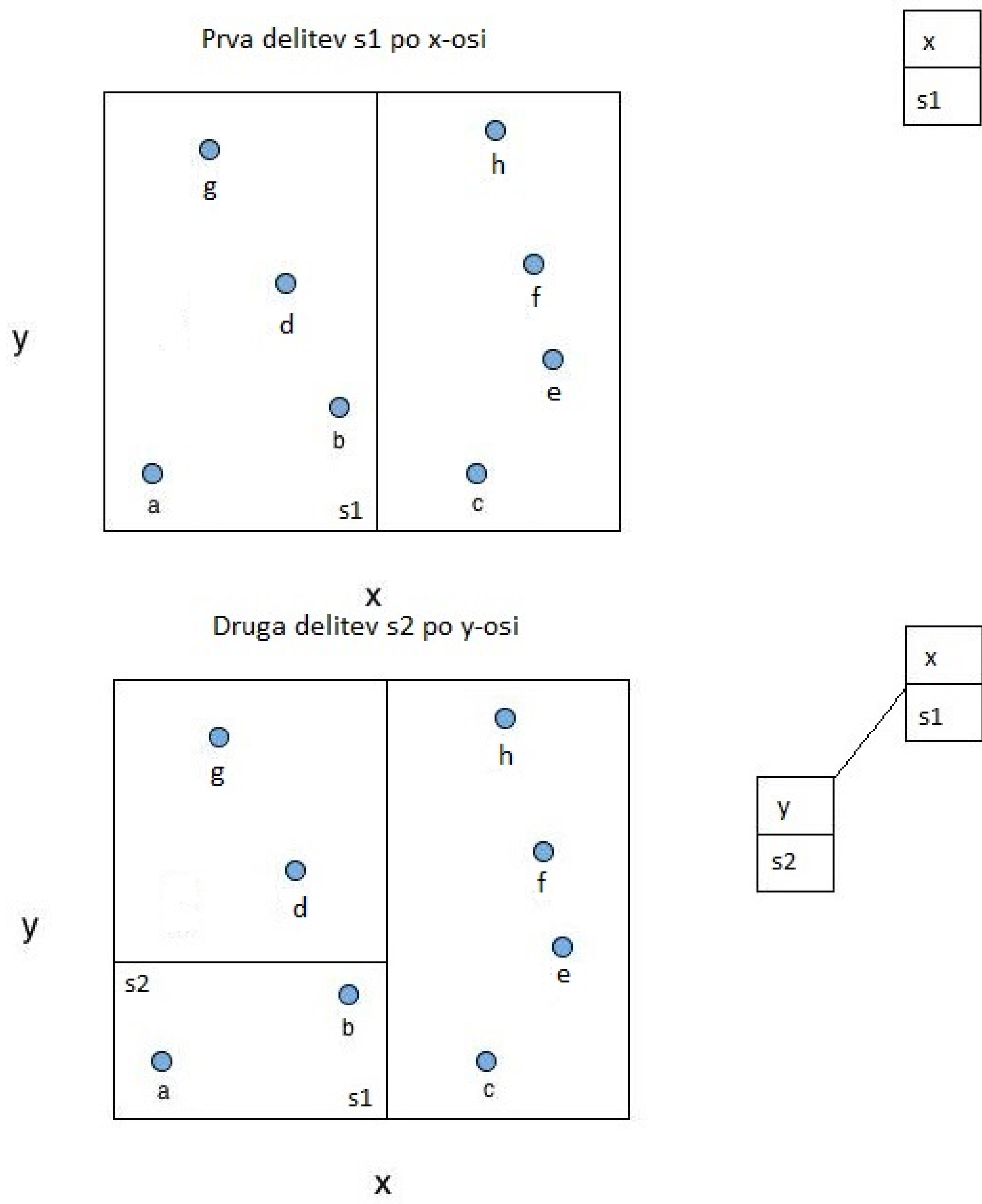
   } }

PRIMER konstrukcije kd-drevesa

Podan je dvodimenzionalen prostor z 8 točkami:



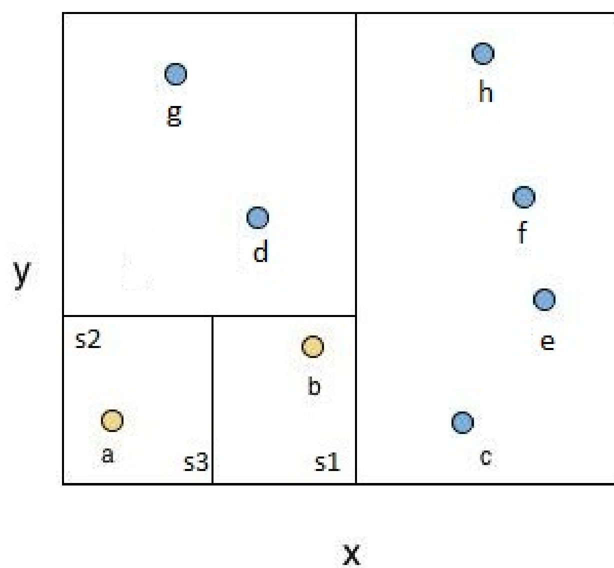
Slika 5: Dvodimenzionalen prostor z 8 točkami

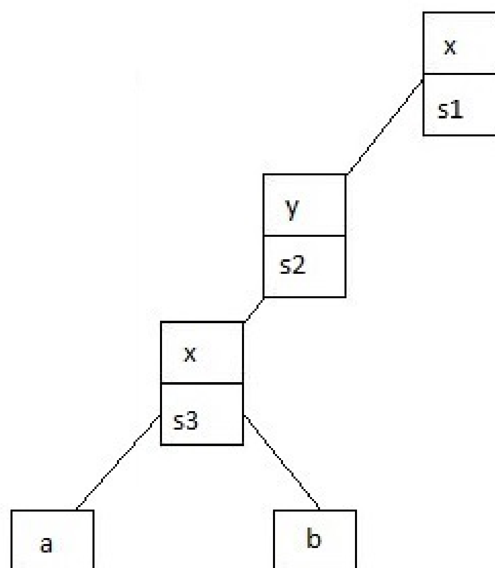


Slika 6: Konstrukcija kd-drevesa (1. in 2. korak)

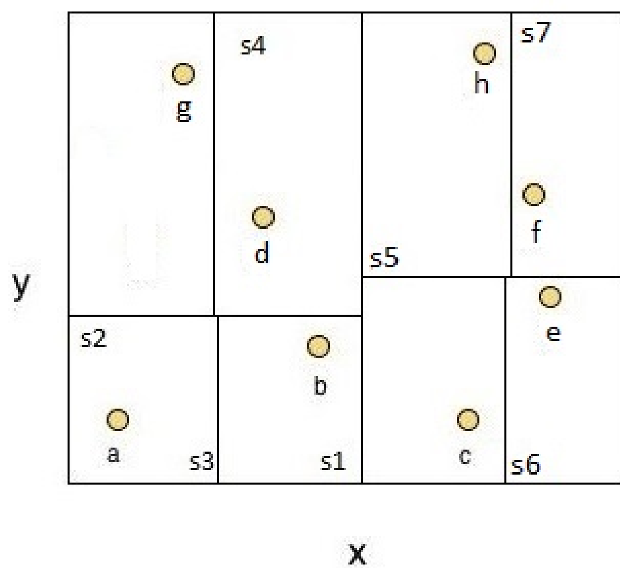
Slika 7: Konstrukcija kd-drevesa (3. in 4. korak)

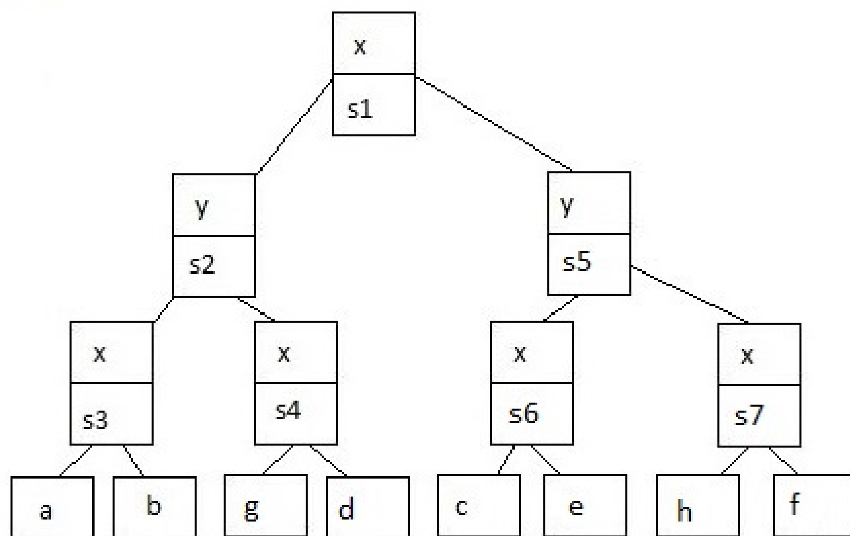
točka b je list drevesa.





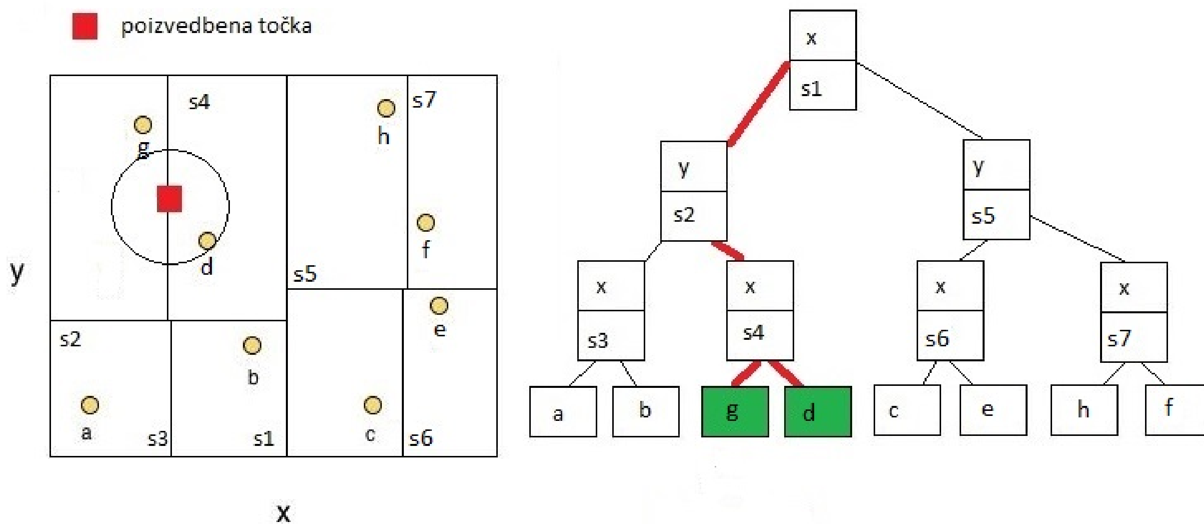
Po koncu konstrukcije dobimo končno kd-drevo:





Slika 8: Konstrukcija kd-drevesa (5.korak in zadnji korak)

Prikažimo iskanje ϵ-najbližjih sosedov z uporabo kd-drevesa na primeru. Natančen algoritem je opisan v [12].



Slika 9: Iskanje najbližjih sosedov v kd-drevesu

Slika 9 kaže poizvedbeno točko v prostoru z 8 točkami in prikaz sprehoda po kd-drevesu za iskanje sosedov. Rdeča povezava v kd-drevesu pomeni sprehod. Obarvano končno vozlišče pomeni, da se to vozlišče primerja s poizvedbeno točko, ali sta si oddaljeni manj kot ϵ. Od korena kd-drevesa sledi sprehod v levo poddrevo, ker krog z radijem ϵ s središčem poizvedbene točke seka s prostorom, ki se nahaja levo od razdelilne premice s1. Ko pride do končnega vozlišča d, se nato vrne nazaj ter preveri drugo opcijo, ali krog seka prostor, v katerem se nahaja h. Če seka, potem se sprehaja do končnega vozlišča h. Sprehod po kd-drevesu se konča, ko se ponovno vrne na začetek korena ter sta oba poddrevesa že preverjena od korena. V tem primeru je vozlišče d edini sosed za poizvedbeno točko.

Obstaja tudi alternativni pristop - t.i. aproksimacijski. δ-aproksimacijsko iskanje najbližjega soseda je poseben primer iskanja. Naj bo napaka $\delta > 0$, točka $p \in S$ je $(1+\delta)$-aproksimacijski sosed poizvedbe q, če $\mathrm{d}(p,q) \leq (1+\delta) * \mathrm{d}(p',q)$, kjer je p' pravi najbližji sosed za q.

Za δ-aproksimacijsko iskanje najbližjega soseda lahko uporabimo kd-drevo ali naivni pristop.

Če uporabljamo δ-aproksimacijsko kd-drevo grafa sosednosti, ne bomo mogli razširiti do pravega Vietors-Ripsovega kompleksa, temveč le do aproksimacijskega Vietoris-Ripsovega kompleksa.
Uporaba aproksimacijskega pristopa je primerna, ko točni algoritmi ne morejo časovno učinkovito iskati rešitev poizvedbe najbližjega soseda.

Za implementacijo računanja grafa sosednosti v C++ uporabljamo knjižnico ANN[14], ki pozna vse naštete pristope. Za ustvaritev grafa sosednosti uporabljamo knjižnico Boost Graph[19] za ustvaritev grafa.

Algoritem za izračun grafa sosednosti:

VHOD: množica točk S, razdalja ϵ
IZHOD: graf sosednosti
za vsako poizvedbeno točko $q \in S$:

1. uporabi algoritem za reševanje problema ϵ-najbližjih sosedov s poizvedbeno točko q in razdaljo ϵ

2. rezultat algoritma so pari točk $(q, a_1), ..., (q, a_l)$ za $a_1, ...a_l \in S$, pri čemer je indeks l enak številu točk, ki so znotraj kroga z radijem ϵ in je $\mathrm{d}(q,a_i) < \epsilon$ za vse i

3. parom točk (q, a_i) za vse i priredimo povezave v grafu

Vhod algoritma za izračun grafa sosednosti je množica točk S. Za poizvedbeno točko vzamemo vse točke iz S. Za vsako izbrano poizvedbeno točko iz S uporabljamo en izmed algoritmov za iskanje ϵ najbližjega soseda iz knjižnice ANN. Ko algoritem najde vse sosede za izbrano poizvedbeno točko, priredimo povezave med poizvedbenimi točkami in vsemi sosedi v graf z uporabo knjižnice Boost Graph. Dobimo graf sosednosti. Ta graf je priprava za vhod v algoritem Vietoris-Ripsove razširitve.

5 Algoritmi Vietoris-Ripsove razširitve

V tem poglavju bomo predstavili tri algoritme za razširitev iz 1-skeleta Vietoris-Ripsovega kompleksa (grafa sosednosti) v k-skelet Vietoris-Ripsovega kompleksa. Ko imamo dokončen k-skelet tega kompleksa, izračunamo še utežno funkcijo za k-skelet Vietoris-Ripsovega kompleksa.

Obravnavali bomo induktivni, inkrementalni in maksimalni algoritem. [21]

Vsak izmed teh algoritmov ima:

- VHOD: graf sosednosti G z utežno funkcijo w in maksimalno dimenzijo k (k>1)
- IZHOD: Vietoris-Ripsov kompleks

5.1 Induktivni algoritem

Spodaj je psevdokoda induktivnega algoritma. Funkcija LOWER-NBRS najde vse sosede oglišča u v grafu G z manjšo utežjo kot u. Oznaka $G.V$ v psevdokodi označuje vozlišča v grafu G. Oznaka $G.E$ označuje povezave v grafu G. Množica ν je rešitev Vietoris-Ripsovega kompleksa.

Psevdokoda induktivnega algoritma

LOWER-NBRS(G,u)
1 **return** $\{v \in G.V \mid u > v, \{u, v\} \in G.E\}$

INDUKTIVNI-VR(G,k)
1 $\nu \leftarrow G.V \cup G.E$
2 **for** $i \leftarrow 1$ **to** k **do**
3   **foreach** i-simpleks $\tau \in \nu$ **do**
4         $N \leftarrow \bigcap_{u\in\tau}$LOWER-NBRS($G$,$u$)
5   **foreach** $v \in N$ **do**
6             $\nu \leftarrow \nu \cup \{\tau \cup \{v\}\}$
7 **return** ν

Izrek 3.8: INDUKTIVNI-VR(G,k) izračunava k-skelet $\nu(G)$

Dokaz:

Izrek dokazujemo z indukcijo na dimenzijo k skeleta simplicialnega kompleksa.

V prvi vrstici je dodan celoten Vieoris-Ripsov graf, ki ustreza 1-skeletu Vietoris-Ripsovega kompleksa, torej izrek v primeru $r = 1$ velja.

Za indukcijski korak privzemimo, da na začetku i-te iteracije zanke, ki se začne v 2. vrstici, simplicialni kompleks vsebuje celoten i-skelet Vietoris-Ripsovega kompleksa. Dokazati moramo, da bodo v i-tem koraku iteracije pravilno dodana vsa $(i+1)$-kolica. Vsak $(i+1)$-simpleks Vietoris-Ripsovega kompleksa je dobljen tako, da nekemu i-simpleksu τ dodamo oglišče, ki je sosedno vsem njegovim ogliščem. Algoritem v 3. vrstici pregleda vsak i-simpleks in v 4. vrstici poišče vsa oglišča, ki so glede na dano ureditev pred oglišči simpleksa τ in so sosedi vseh njegovih oglišč. V 5. in 6. vrstici za vsako tako oglišče v v simplicialni kompleks doda simpleks $\sigma = \tau \cup \{v\}$. Preveriti moramo, da je σ res simpleks, torej, da vsebuje vsa svoja lica. Zaradi indukcijske predpostavke vsebuje vsa lica iz τ, ker je v sosed vseh oglišč simpleksa τ, pa vsebuje tudi vsa lica, ki imajo za eno oglišče novi simpleks v. Torej je σ res simpleks. Ker je v vedno minimalno oglišče (glede na dano ureditev) novega simpleksa σ in je minimalno oglišče v vsakem simpleksu eno samo, algoritem vsak simpleks upošteva samo enkrat in izrek je dokazan.

■

5.2 Inkrementalni algoritem

Spodaj je psevdokoda inkrementalnega algoritma.

Psevdokoda inkrementalnega algoritma

INKREMENTALNI-VR(G,k)
1 $\nu \leftarrow \emptyset$
2 **foreach** $u \in G.V$ **do**
3 $N \leftarrow$ LOWER-NBRS(G,u)
4 ADD-COFACES(G,k,$\{u\}$,N,ν)
5 **return** ν

ADD-COFACES(G,k,τ,N,ν)
1 $\nu \leftarrow \nu \cup \{\tau\}$
2 **if** $\dim(\tau) \geq k$ **then return**
3 **else foreach** $v \in N$ **do**
4 $\sigma \leftarrow \tau \cup \{v\}$
5 $M \leftarrow N \cap$ LOWER-NBRS(G,v)
6 ADD-COFACES(G,k,σ,M,ν)

Izrek 3.8: INKREMENTALNI-VR(G,k) izračunava k-skelet $\nu(G)$

Dokaz:

Rekurzivna funkcija ADD-COFACES ima 2 pomembna argumenta:

- Prvi je τ, ki je simpleks k-skeleta
- Drugi je N, ki je množica nižjih sosedov oglišč simpleksa τ

Na začetku vsakega koraka zanke v drugi vrstici algoritma, kompleks ν že vsebuje vse simplekse k-skeleta, ki imajo maksimalno oglišče glede na izbrano ureditev manjše od u. Rekurzivna funkcija ADD-COFACES na tem koraku v k-skelet doda vse simplekse z maksimalnim ogliščem u. Ta lastnost algoritma zagotavlja, da bo vsak simpleks v k-skelet dodan na enem samem koraku.

Ob prvem klicu funkcije je τ oglišče in sta obe zahtevi izpolnjeni, saj mora skelet vsebovati vsa oglišča simplicialnega kompleksa, N pa je ravno množica spodnjih sosedov tega oglišča. Pokazati moramo, da sta obe zahtevi izpolnjeni tudi ob vsakem naslednjem klicu funkcije. Ukaz v 2. vrstici poskrbi za to, da v k-skelet ni dodan noben simpleks dimenzije več kot k, simpleks σ, ki nastane v 4. vrstici pa je spet simpleks k-skeleta, saj je dobljen kot unija

simpkesa τ, ki je v k-skeletu in oglišča v, ki je spodnji sosed oglišč simpleksa τ. V 5. vrstici so pravilno izračunani spodnji sosedi novega simpleksa in klic algoritma v 6. vrstici ima pravilne vhodne podatke.

■

5.3 Maksimalni algoritem

Oba prejšnja algoritma delujeta po principu od spodaj navzgor, se pravi, pri konstrukciji začne z nižjimi dimenzijami lic in jim dodaja kolica. V nadaljevanju predstavimo od zgoraj navzdol algoritem, ki temelji na tem, da je Vietoris-Ripsov kompleks kompleks klik ("clique complex") Vietoris-Ripsovega grafa.

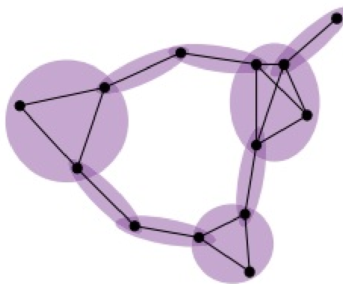
Definicija 3.9: Klika

Klika v neusmerjenem grafu $G = (V, E)$ je podmnožica množic vozlišč $C \subseteq V$ tako, da za vsaki dve vozlišči v C obstaja povezava teh dveh. To je enako, kot bi rekli, da je podgraf, induciran z C, poln.

Definicija 3.10: Maksimalna klika

Klika je *maksimalna*, če jo ne moremo povečati.

Kompleks klik vsebuje maksimalne klike grafa za svoje maksimalne simplekse.



Slika 10: Maksimalna klika. Vsako ovalno območje označuje eno od 9 maksimalnih klik, ki predstavljajo maksimalne simplekse v Vietoris-Ripsovem kompleksu.

Algoritem je preprost: označimo vse maksimalne klike C, potem pa generiramo vsa lica simpleksov, ki pripadajo maksimalnim klikam, torej vse podmnožice množice oglišč posamezne klike.

Psevdokoda maksimalnega algoritma:

MAKSIMALNI-VR(G,k)
1 $C \leftarrow$ BRON-KERBOSCH(G)
2 $\nu \leftarrow$ GENERATE-FACES(C,$k+1$)
3 **return** ν

Prvi korak algoritma MAKSIMALNI-VR: z algoritmom BRON-KERBOSCH iščemo maksimalno kliko našega neusmerjenega grafa. Bron-Kerboschov algoritem [1] za iskanje maksimalnih klik v povezanem grafu je rekurzivni algoritem, ki množicam oglišč R, P in X priredi vse maksimalne klike, ki vsebujejo vsa oglišča iz R, nekaj oglišč iz P in nobenega oglišča iz X. Opis algoritma je povzet po [4].

Psevdokoda Bron-Kerbosch brez pivotiranja

Bron-Kerbosch1(R,P,X):
1 če sta P in X obe prazni:
2 R je maksimalna klika
3 za vsako točko $v \in P$:
4 Bron-Kerbosch1($R \cup \{v\}$,$P\cap$ N(v),$X\cap$N(v))
5 $P := P \setminus \{v\}$
6 $X := X \cup \{v\}$

kjer je N(v) množica točk, ki so sosedi točke v.
Da bi dobili seznam maksimalnih klik, kličemo program BronKerbosch1($\emptyset$, $V(G)$,$\emptyset$), kjer je $V(G)$ množica vozlišč v grafu G.

Algoritem pospešimo z uporabo pivotiranja za iskanje maksimalnih klik. Pivotiranje zahteva manj rekurzivnih klicev kakor pri osnovni obliki Bron-Kerboschevega algoritma.

Psevdokoda Bron-Kerbosch s pivotiranjem

Bron-Kerbosch2(R,P,X):
1 če sta P in X obe prazni:
2 R je maksimalna klika
3 za pivot u izberimo eno od oglišč z najvišjo stopnjo v grafu G
4 za vsako točko $v \in P \setminus$ N(u):
5 Bron-Kerbosch2($R \cup \{v\}$,$P\cap$ N(v),$X\cap$N(v))
6 $P := P \setminus \{v\}$
7 $X := X \cup \{v\}$

kjer je N(v) množica točk, ki so sosedi točke v.
Da bi dobili seznam maksimalnih klik, kličemo program BronKerbosch2($\emptyset$, $V(G)$,$\emptyset$), kjer je $V(G)$ množica vozlišč v grafu G.

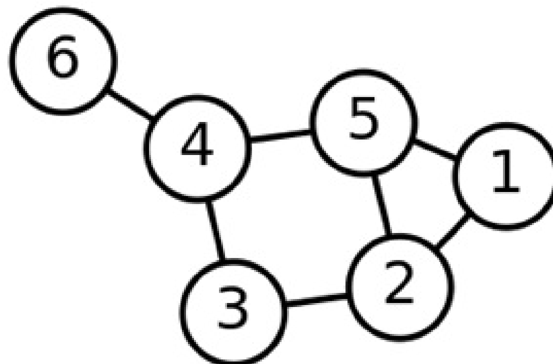
V drugem koraku MAKSIMALNI-VR(G, k) generiramo vse podmnožice vseh maksimalnih klik. Vsaka podmnožica maksimalne klike σ z r elementi je $(r-1)$-lice σ kot simpleks. Generiranje kombinacij je klasičen problem v diskretni matematiki.

Izrek 3.11: MAKSIMALNI-VR(G, k) izračuna k-skelet $\nu(G)$.

Dokaz izreka sledi neposredno iz pravilnosti Bron-Kerboschovega algoritma.

PRIMER:

Vzemimo primer majhnega vhodnega grafa za ta algoritem. [2]



Slika 11: Graf s petimi maksimalnimi klikami: štiri povezave in en trikotnik.

V prvem koraku algoritma najprej kličemo Bron-Kerboschov algoritem z vrednostmi $R = \emptyset$, $P = \{1, 2, 3, 4, 5, 6\}$ in $X = \emptyset$. Za pivot u izberimo eno od oglišč z najvišjo stopnjo, tako bo čim manj rekurzivnih klicev. V grafu imajo vozlišča 2, 4 in 5 stopnjo 3, izberimo za pivot vozlišče 2. Potem so $P\backslash$N(2)= $\{2, 4, 6\}$, kjer je N(2)=$\{1, 3, 5\}$. Iteracija notranje zanke algoritma za $v = 2$ naredi rekurziven klic za algoritem z $R = \{2\}$, $P = \{1, 3, 5\}$ in $X = \emptyset$.

Spodaj je celotno zaporedje klicev algoritma Bron-Kerbosch s pivotiranjem. Z zamikom v desno pomeni rekurziven klic, z zamikom v levo pa

ne naredi rekurzivnega klica, temveč sporoči rezultat množice R ter se vrne nazaj.

Bron-Kerbosch2($\emptyset$, $\{1, 2, 3, 4, 5, 6\}$, $\emptyset$)
    Bron-Kerbosch2($\{2\}$,$\{1, 3, 5\}$, $\emptyset$)
        Bron-Kerbosch2($\{2, 3\}$,$\emptyset$, $\emptyset$):output$\{2, 3\}$
        Bron-Kerbosch2($\{2, 5\}$,$\{1\}$, $\emptyset$)
            Bron-Kerbosch2($\{1, 2, 5\}$,$\emptyset$, $\emptyset$):output$\{1, 2, 5\}$
    Bron-Kerbosch2($\{4\}$,$\{3, 5, 6\}$, $\emptyset$)
        Bron-Kerbosch2($\{3, 4\}$,$\emptyset$, $\emptyset$):output$\{3, 4\}$
        Bron-Kerbosch2($\{4, 5\}$,$\emptyset$, $\emptyset$):output$\{4, 5\}$
        Bron-Kerbosch2($\{4, 6\}$,$\emptyset$, $\emptyset$):output$\{4, 6\}$
    Bron-Kerbosch2($\{6\}$,$\emptyset$, $\{4\}$):no output

Množica maksimalnih klik je: $\{2, 3\}$ $\{1, 2, 5\}$ $\{3, 4\}$ $\{4, 5\}$ in $\{4, 6\}$.

Maksimalne klike ustrezajo maksimalnim simpleksom Vietoris-Ripsovega kompleksa. V drugem koraku algoritma generiramo še vsa lica maksimalnih simpleksov, da dobimo cel Vietoris-Ripsov kompleks:
($\{2\}$, $\{3\}$, $\{2,3\}$, $\{1\}$, $\{2\}$, $\{5\}$, $\{1,2\}$, $\{1,5\}$, $\{2,5\}$, $\{1,2,5\}$, $\{3\}$, $\{4\}$, $\{3,4\}$, $\{4\}$, $\{5\}$, $\{4,5\}$, $\{4\}$, $\{6\}$, $\{4,6\}$)

5.4 Utežna funkcija

Na koncu še napišemo algoritem za izračun utežne funkcije za simplekse Vietoris-Ripsov kompleksa. Naj bo Vietoris-Ripsov kompleks ν, algoritem spodaj vrne razširitev utežne funkcije $\omega : \nu \to \mathbb{R}$.

COMPUTE-WEIGHTS(ν,ω)
1 **foreach** vertex $v \in \nu$ **do**
2 $\omega(v) \leftarrow 0$
3 **foreach** edge $e \in \nu$ **do**
4 $\omega(e) \leftarrow \omega(e)$
5 **foreach** simplex $\sigma \in \nu$ **do**
6 WEIGHT(σ,ω)
7 **return** ω

WEIGHT(σ,ω)
1 **if** $\omega(\sigma)$ is defined **then**
2 **return** $\omega(\sigma)$
3 **else** **return** $\omega(\sigma) \leftarrow \max_{\tau \subset \sigma}$ WEIGHT(τ,ω)

6 Implementacija kode

Za testiranje opisanih algoritmov za konstrukcijo Vietoris-Ripsovega kompleksa smo uporabili programski jezik C++ v okolju Linux.

Za konstrukcijo grafa sosednosti uporabljamo knjižnico ANN in Boost Graph.

ANN (kratica Approximate Nearest Neighbor) je knjižnica, napisana v programskem jeziku C++ za podporo točnega in aproksimacijskega iskanja najbližjega soseda v prostorih različnih dimenzij. ANN sta prva implementirala David M.Mount in Sunil Arya.

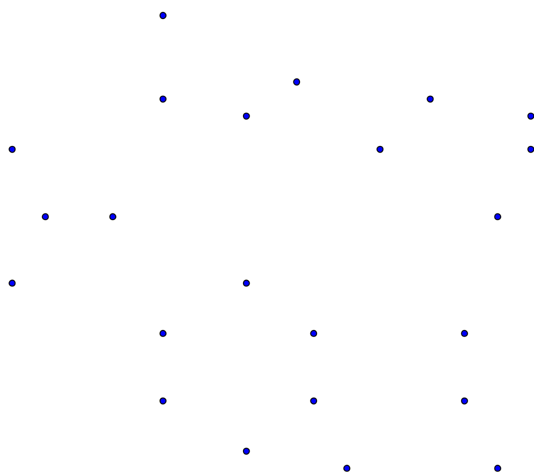
Knjižnico uporabljamo v skadu z navodili [14].

Boost Graph Library je namenjena za delo s grafi kot je shranjevanje grafa sosednosti.

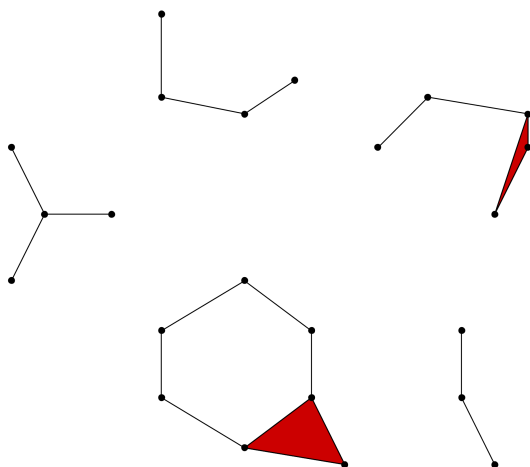
7 Rezultat

Na množici točk v ravnini, ki je prikazana na sliki 12, smo zgradili Vietoris-Ripsove komplekse pri treh različnih merilih: $\sqrt{10}$, $\sqrt{20}$ in $\sqrt{40}$. Koordinate točk so v tabeli 1, zgrajeni Vietoris-Ripsovi kompleksi pa na slikah 13,14 in 15.

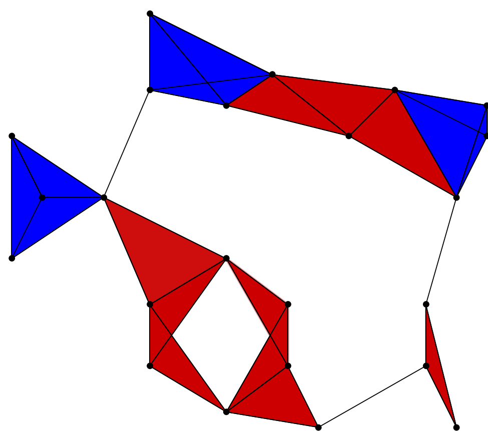
Poglejmo, kako izgleda Vietoris-Ripsov kompleks pri merilu $\epsilon = \sqrt{10}$ pri sliki 13, $\epsilon = \sqrt{20}$ pri sliki 14 in $\epsilon = \sqrt{40}$ pri sliki 15. Rdeče obarvani liki so trikotniki, modro tetraedri. S sivo barvo so ponazorjeni 4-simpleks(pentakron), s temnosivo pa 5-simpleks.



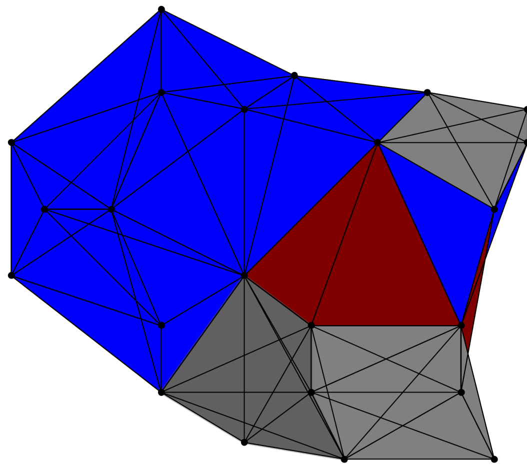
Slika 12: Množica 23 točk v Evklidski ravnini



Slika 13: Vietoris-Ripsov kompleks pri $\epsilon = \sqrt{10}$



Slika 14: Vietoris-Ripsov kompleks pri $\epsilon = \sqrt{20}$



Slika 15: Vietoris-Ripsov kompleks pri $\epsilon = \sqrt{40}$

Spodaj določimo koordinate točk, ki so razporejene kot na sliki 12.

(-8,0)	(-8,4)	(-7,2)
(-5,2)	(-3.5,5.5)	(-3.5,8)
(-1,5)	(0.5,6)	(3,4)
(4.5,5.5)	(7.5,5)	(7.5,4)
(6.5,2)	(5.5,-1.5)	(5.5, -3.5)
(6.5,-5.5)	(2, -5.5)	(1, -3.5)
(-1, -5)	(-3.5,-3.5)	(-3.5,-1.5)
(-1,0)	(1,-1.5)	

Tabela 1: koordinate točk, ki so razporejene kot na sliki 12.

Izmerili bomo izvajanja posameznih treh različnih algoritmov, ki smo jih implementirali (Inkrementalni, induktivni in maksimalni algoritem). Vhodni podatki so koordinate točk in pripadajoča metrika po sliki 12 in tabeli 1.

Izmerili smo čas 100 ponovitev izvajanja algoritma in izračunali povprečen čas.

Rezultati:

Časovni rezultat pri merilu $\epsilon= \sqrt{10}$ za vse tri algoritme:

- Inkrementalni algoritem ima povprečni čas 9,6 milisekund (ms)
- Induktivni algoritem ima povprečni čas 8,7 ms
- Maksimalni algoritem ima povprečni čas 12,2 ms

Vidimo, da je pri merilu $\epsilon=\sqrt{10}$ najhitrejši algoritem induktivni, za njim je inkrementalni, najbolj počasen je maksimalni.

Časovni rezultat pri merilu $\epsilon= \sqrt{20}$ za vse tri algoritme:

- Inkrementalni algoritem ima povprečni čas 11,4 ms
- Induktivni algoritem ima povprečni čas 11,2 ms
- Maksimalni algoritem ima povprečni čas 18,2 ms

V tem primeru, pri merilu $\epsilon=\sqrt{20}$ sta inkrementalni in induktivni približno enaka hitra, najbolj počasen je še vedno maksimalni.

Časovni rezultat pri merilu $\epsilon= \sqrt{40}$ za vse tri algoritme:

- Inkrementalni algoritem ima povprečni čas 18,2 ms
- Induktivni algoritem ima povprečni čas 21,5 ms
- Maksimalni algoritem ima povprečni čas 72,8 ms

V zadnjem primeru, ko je merilo $\epsilon=\sqrt{40}$, je inkrementalni algoritem hitrejši od induktivnega, najpočasnejši pa je še vedno maksimalni.

Pri manjšem merilu je torej induktivni algoritem najhitrejši. Z večanjem razdalje je inkrementalni hitrejši od induktivnega.

8 Zaključek

Opisali smo dvostopenjski algoritem za izgradnjo Vietoris-Ripsovega kompleksa. Implementirani so bili trije različni algoritmi: induktivni, inkrementalni in maksimalni, ki so podrobneje predstavljeni v članku [21]. Na izbranem vzorcu točk je bil zgrajen Vietoris-Ripsov kompleks pri različnih merilih in narejena je bila primerjava algoritmov glede na čas, potreben za izgradnjo kompleksov. Izkazalo se je, da je maksimalni algoritem pri vseh merilih najpočasnejši, kar ni presentljivo. Pri manjših merilih je induktivni algoritem najhitrejši, pri največjem merilu, kjer je število nastalih simpleksov največje, pa je inkrementalni algoritem hitrejši od induktivnega. Podrobnejše analize časovne kompleksnosti algoritmov nismo opravili. Druga temeljna ugotovitev je, da je za hitrost izgradnje Vietoris-Ripsovega kompleksa pomembna izbira merila. Veliko merilo vpliva na potek reševanja predvsem zaradi velikega števila simpleksov in njihove velike dimenzije, kjer na čas izračuna vplivajo omejitve računalniške zmogljivosti.

A Priloga: Implementacija programske kode

Za implementacijo kode uporabljamo programski jezik *C++*.

Na začetku najprej moramo dodati zaglavje datoteke v *C++*:

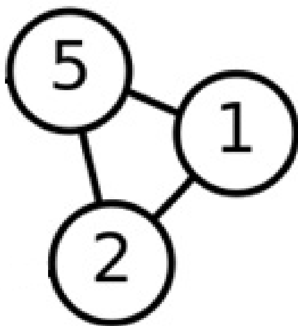
```
#include <iostream>
#include <boost/graph/adjacency_list.hpp> (knjižnica Boost za graf)
#include <ANN/ANN.h>                      (knjižnica ANN)
#include <fstream>                        (za branje/pisanje datoteke)
#include <set>                            (za delo z množicami)
#include <map>                            (za delo s preslikavami)
using namespace std;
```

Tu so deklaracije in definicije funkcij in spremenljivk.

Pa glavni program `int main(int argc, char **argv){`

```
...

}
```

Kako shranjujemo Vietoris-Ripsov kompleks?
To lahko naredimo s pomočjo uporabe funkcije `set` iz standardne knjižnice *C++* (*std*).
Pokažimo na primeru abstraktnega simplicialnega kompleksa, ki vsebuje trikotnik:



Slika 16: Abstraktni simplicialni kompleks, ki vsebuje trikotnik

V *C++*:

```
set<set<int> > VR;                              //Vietoris-Rips

//{1},{2},{5} dodamo v Vietoris-Ripsov kompleks VR
set.insert(1); VR.insert(set); set.clear();
set.insert(2); VR.insert(set); set.clear();
set.insert(5); VR.insert(set); set.clear();

//{1,2},{1,5},{2,5} dodamo v Vietoris-Ripsov kompleks VR
set.insert(1); set.insert(2); VR.insert(set); set.clear();
set.insert(1); set.insert(5); VR.insert(set); set.clear();
set.insert(2); set.insert(5); VR.insert(set); set.clear();

//{1,2,5} dodamo v Vietoris-Ripsov kompleks VR
set.insert(1); set.insert(2);set.insert(5); VR.insert(set);
```

Tako *VR* ima množico $\{\{1\},\{2\},\{5\},\{1,2\},\{1,5\},\{2,5\},\{1,2,5\}\}$ $\{1\},\{2\},\{5\}$ so oglišča tega kompleksa. $\{1,2\},\{1,5\},\{2,5\}$ so povezave tega kompleksa. $\{1,2,5\}$ je trikotnik tega kompleksa.

Funkcijo `map` iz standardne knjižnice *C++* uporabljamo kot utežno funkcijo.

Primer povezave 1,2; ki sta z razdaljo 0.65 dodamo v utežno funkcijo:

```
map<set<int>,double> weight;                    //utežna funkcija

set.insert(1);set.insert(2);            //množica {1,2} je povezava
//utežna funkcija množice {1,2} priredimo razdaljo 0.65
weight[set]=0.65;
```

A.1 Implementacija kode za računanje prve faze

V ukazni vrstici poženemo program z enim argumentom datoteke, v katerem je zapisano zaporedje realnih števil.

Po zagonu programa na zaslon izpiše zahtevek vnosa podatkov za spremenljivke, ki jih potrebujemo kot podatke za računanje prve faze.

V funkciji `main()` v *C++*:

```
int dim; double distance;
cout<<"Dimenzija prostora:"
cin>>dim;
cout<< "Maksimum dimenzije:"
cin>>k;
cout<<"Razdalja:"
cin>>distance;
```

Spremenljivka `dim` je celo število, ki kaže na velikosti dimenzije prostora. Spremenljivka `k` je celo število, ki kaže na maksimum dimenzije za Vietoris-Ripsov kompleks. Spremenljivka `distance` je merilo za razdaljo med dvema točkama. Ukaz `cin>>dim` pomeni vnos podatkov v spremenljivko `dim`. Vnos je izveden s tipkovnico.

Branje podatkov iz datotek:

V kodi *C++*:

```
int main(argc,char **argv) {

...

getArgs(argc,argv);                //kličemo funkcijo getArgs

...

 }
```

Funkcija `getArgs` je funkcija, kjer iz ukazne vrstice dobi argument, v katerem je naslovljena datoteka ter jo odpre:

```
istream* dataIn=NULL;

void getArgs(int argc, char**argv){
static ifstream dataStream;

if (argc≤1) cout«"Ni datoteke";  //argc je število argumentov, ki
                                  //smo jih podali v ukazni vrstici
if (argc==2) {                    //v ukazni vrstici
                                  //$VRComplex.exe datoteka sta dve
                                  //argumenta
    dataStream.open()argv[1],ios::in);
                                  //argv je vrednost argumenta. Ta
                                  //ukaz odpre datoteko argv[1]
    if(!dataStream){              //če je argument napačen
                                  //(primer: ni prave datoteke)
        cerr<<"Ne more odpreti datoteke"; }
                                  //se izpiše na zaslon napake
    dataIn=&dataStream; }         //globalna spremenljivka dataIn
                                  //shrani vse podatke iz datoteke
else cerr<<"Samo en argument"; }
                                  //na zaslon bo izpisana napaka, če
                                  //smo dali preveč argumentov
                                  //v ukazni vrstici
```

Funkcija `readPt` je funkcija, ki zaporedje realnih števil $a_1, ..., a_{dim}$ iz datoteke vnaša v koordinato `p[0],...,p[dim-1]`. `p[i]` je koordinata in `p` je točka. `dim` je dimenzija prostora.

```
bool readPt(istream &in, ANNpoint p) {
    for(int i=0; i<dim;i++){
        if(!(in>>p[i])) return false;
                                  //če neuspešno bere iz datoteke, vrne
                                  //false (neuspešen vnosa podatkov v
                                  //točki)
return true; }
```

Zdaj pa poglejmo, kako implementiramo kodo za računanje prve faze.

V glavni funkciji `int main`:

```
int main(int argc, char **argv) {

getArgs(argc,argv);           //kličemo funkcijo getArgs, da
                              //dobimo podatke iz datoteke
int nPts=0;                   //spremenljivka nPts je število točk
int maxPts=1000;              //maxPts je maksimalno število točk

ANNpointArray dataPts;     //deklaracija spremenljivke za polja točk
ANNpoint queryPt;          //deklaracija spremenljivke za poizvedbeno
                           //točko
ANNidxArray nnIdx=NULL;    //definicija spremenljivke za polja
                           //indeksnih točk
ANNdistArray dists;        //deklaracija spremenljivke za polja razdalj
ANNkd_tree* kdTree;        //deklaracija kd-drevesa


queryPt=annAllocPt(dim);              //definicija queryPt
dataPts=annAllocPts(maxPts,dim);      //definicija dataPts

while(nPts<maxPts && readPt(*dataIn,dataPts[nPts])) {
nPts++; }                  //za vsako iteracijo se poveča nPts za eno

//Zgoraj: iz datoteke realnih števil vnašamo v polje točk dataPts:
//dataPts[0] je prva točka, dataPts[1] je druga točka in tako dalje.
//dataPts[0][0] je prva koordinata prve točke, dataPts[0][1] je druga
//koordinata prve točke.

nnIdx=new ANNIdx[nPts];         //število točk nPts je že znano, zato
                                //definiramo nnIdx
dists=new ANNdists[nPts];       //definicija dists

Graph g(nPts);                  //ustvarimo graf z nPts točkami
```

(izven glavne funkcije `int main` smo definirali
`typedef boost::adjacency_list<boost::vecS,boost::vecS,boost::directedS> Graph;`. Od tod zato lahko zgoraj uporabimo `Graph g(nPts);` za ustvaritev grafa)

RAČUNANJE GRAFA SOSEDNOSTI

```
kdTree=new ANNkd_tree(dataPts,nPts,dim);    //konstruktor kd-drevesa

for(int i=0;i<nPts;i++) {
     queryPt=dataPts[i];
     kdTree->annkFRSearch(queryPt,distance,nPts,nnIdx,dists,0);

     for(int j=0;i<nPts && nnIdx[j]!=-1; j++) {
                                  //for zanka za dodajanje povezav med
//poizvedbeno točko in vsemi bližnjimi sosedi v graf. Pogoj nnIdx[j]!=-1
//pomeni, da so bližnji sosedi (vrednost -1 pomeni, da ni bližnjih sosedov)

           if(0!=dists[j]){     //pogoj 0!=dists[j] prepreči, da je
//najbližji sosed za poizvedbeno točko sam sebi najbližji.

                 if(i<nnIdx[j]) {  //pogoj i<nnIdx[j] omogoča, da
//ne dodamo dve povezavi z dvema istima točkama, temveč le eno
//povezavo (na primer dodamo samo povezavo (1,2) in (2,1) ne dovolimo)

                    boost::add_edge(i,nnIdx[j],g); //dodamo
//povezavo (i,nnIdx[j]) v graf

                    set<int> set1;
                    set1.insert(i);
                    set1.insert(nnIdx[j]);
                    weight[set1]=sqrt(dists[j]); //za povezavo med
//dvema točkama izračunamo razdaljo in jo dodamo v utežno funkcijo }
}
```

Opis zgornjega algoritma: za vsako točko `dataPts[0],...,dataPts[nPts-1]` uporabimo `dataPts[i]` kot poizvedbeno točko `queryPt=dataPts[i]`. Izvedi funkcijo `annkFRSearch` za poizvedbeno točko `queryPt`. Po izvedbi `annkFRSearch` dobimo novo informacijo za `nnIdx` in `dists`. `nnIdx[0]` je najbližji sosed za poizvedbeno točko `queryPt`, `dists[0]` pa je razdalja med `nnIdx[0]` in `queryPt`. Indeks poizvedbene točke in vse indekse bližnjih sosedov za poizvedbeno točko shranimo kot povezavo v graf. Prav tako razdaljo med njima shranimo v utežno funkcijo.

Po koncu računanja prve faze še zapremo ANN:

```
delete[]nnIdx;
delete[]dists;
```

```
kdTree;
annClose();
```

Po tem dobimo že graf sosednosti, ki je rešitev prve faze. Naslednji korak je konstrukcija Vietoris-Ripsovega kompleksa. Implementirali bomo posebej 3 različnih metod: induktivna, inkrementalna in maksimalna.

A.2 Implementacija kode za računanje druge faze

Induktivni algoritem

Psevdokoda induktivnega algoritma:

LOWER-NBRS(G,u)
1 **return** $\{v \in G.V | u > v, \{u, v\} \in G.E\}$

INDUKTIVNI-VR(G,k)
1 $\nu \leftarrow G.V \cup G.E$
2 **for** $i \leftarrow 1$ **to** k **do**
3 **foreach** i-simpleks $\tau \in \nu$ **do**
4 $N \leftarrow \bigcap_{u \in \tau}$LOWER-NBRS($G$,$u$)
5 **foreach** $v \in N$ **do**
6 $\nu \leftarrow \nu \cup \{\tau \cup \{v\}\}$
7 **return** ν

Funkcija LOWER-NBRS(G, u) z argumentom grafa G in indeksne točke u kot rezultat vrne množico indeksov tistih vozlišč, ki izpolnjujejo zahtevan pogoj: indeks sosednjih vozlišč je manjši od indeksa u - "nižji" sosedi.

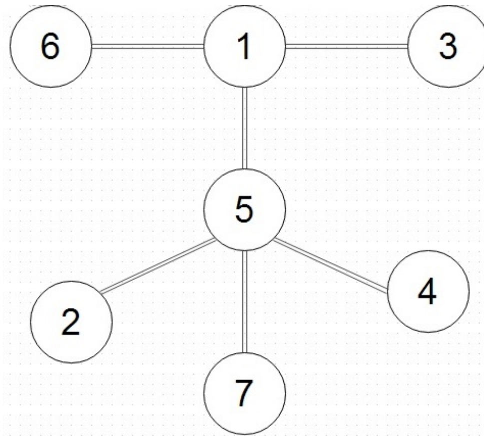
V C++ implementiramo funkcijo LOWER-NBRS(G, u):

```
map<int,set<int> > neighbors;         //deklaracija preslikav neighbors.
//Ključ neighbors je indeksna točka in vrednost neighbors pa je množica
//indeksnih točk, ki so sosedi za ključ. neighbors[3]={1,2}  pomeni,
//da sta indeksni točki 1 in 2 soseda za indeksno točko 3.

//Funkcija LOWER-NBRS(G, u):

template<typename Graph>
set<int> LOWER-NBRS(const Graph & g, int u){
```

```
VItr vitr, vend;               //iterator za graf. Iterator omogoča
//sprehajanje po grafu.
set<int> sosedi;               //množica sosedov

if(neighbors.count(u)==0){  //neighbors.count(u)==0 pomeni, da
//je neighbors s ključem u brez vrednosti, zato po grafu iščemo nove sosede
//za ključ

//Za vsako vozlišče v ∈ G primerja z u in preveri, če je u povezan z v.
//Če je povezan, dodamo ga v množico sosedi. V tej množici
//so shranjeni vsi sosedi vozlišča u, katerih indeks je manjši od u.

for(boost::tie(vitr,vend)=boost::vertices(g); vitr!=vend; vitr++){
    if(u>*vitr)
        if(boost::edge(*vitr,u,g).second) {
            sosedi.insert(*vitr); } }


neighbors[u]=sosedi;           //množico sosedi dodamo v neighbors[u]
//zato, da ni potrebno ponovno preiskovanje sosedov po grafu.
return sosedi; }               //vrne rezultat

return neighbors[u]; }         //vrne rezultat brez sprehajanja po grafu
//Konec funkcije
```

PRIMER nižji sosed:

Naj bo graf G podan:



Slika 17: Graf s 7 indeksnimi točkami

Funkcija LOWER-NBRS($G, 5$) vrne rezultat {1,2,4}. Sosedi, katerih indeks je manjši od 5, zato točko 7 ne sodi v množico.

Implementacija induktivnega algoritma INDUKTIVNI-VR(G,k):

```
template<typename Graph>
set<set<int> >INDUCTIVE_VR(const Graph & g, int k){
set<set<int> >V;              //množica simpleksov za Vietoris-Ripsov
                              //kompleks
set<set<int> >V1[k];          //množica i-simpleksov. (V1[1] je množica
                              //1-simpleksov, V1[2] je množica 2-simpleksov
                              //itd.)
VItr vitr, vend;              //iterator za vozlišča v grafu
EItr eitr, eend;              //iterator za povezave v grafu


set<int> N,N1,N2,v;
set<int>::iterator it1,it2;
set<set<int> >::iterator it;

int index=2;
```

Prvo vrstico pri psevdokodi induktivnega algoritma prve vrstice 1 $\nu \leftarrow$
$G.V \cup G.E$
implementiramo:

```
for (boost::tie(vitr,vend)=boost::vertices(g); vitr!= vend; vitr++){

    set<int> set2;
    set2.insert(*vitr);
    V.insert(set2);                    //ν ← G.V
    V1[0].insert(set2); }


for (boost::tie(eitr,eend)=boost::edges(g); eitr!= eend; eitr++){

    set<int> set2;
    set2.insert(source(*eitr,g));
    set2.insert(target(*eitr,g));
    V.insert(set2);                    //ν ← ν ∪ G.E
    V1[1].insert(set2); }
```

2. in 3. vrstico pri psevdokodi induktivnega algoritma
2 **for** $i \leftarrow 1$ **to** k **do**
3 **foreach** i-simpleks $\tau \in \nu$ **do**
implementiramo:

```
for (int i=1; i<=k; i++) {
    if(V[i].size()!=0) {
        for(it=V1[i].begin(); it!=V1[i].end(); it++) {
```

4.vrstico pri psevdokodi induktivnega algoritma $N \leftarrow \bigcap_{u\in\tau}$LOWER-NBRS($G$,$u$)
implementiramo:

```
        N.clear();

        it1=(*it).begin();
        N1=LOWER-NBRS(g,*(it1));
        it1++;
        N2=LOWER-NBRS(g,*(it1));
```

```
set_intersection(N1.begin(),N1.end(),N2.begin(),N2.end(),
inserter(N,N.end()));     //set_intersection vrne rezultat
//N, ki je presek med N1 in N2 (N=N1∩N2).
N1.clear(); N2.clear();
```

//če je $\bigcap_{u\in\tau}$LOWER-NBRS(G,u) presek več kot 2 množic, potem
//izračunamo:

```
if(N.size()!=0) {
  while(index<(*it).size()) {
    it1++;
    N1=LOWER-NBRS(g,*(it1));

    set_intersection(N1.begin(),N1.end(),N.begin(),N.end(),
    inserter(N2,N2.end()));

    N=N2;
    N2.clear(); N1.clear();

    index++; }

  index=2;
```

Pri psevdokodi 5, 6 in 7 vrstice

5 **foreach** $v \in N$ **do**
6 $\nu \leftarrow \nu \cup \{\tau \cup \{v\}\}$
7 **return** ν

implementiramo:

```
    for (it2=N.begin(); it2!=N.end();it2++) {

    v=*it;
    v.insert(*it2);
    V.insert(v);
    V1[i+1].insert(v); } } } } }


return V; }
```

Inkrementalni algoritem

Psevdokodo inkrementalnega algoritma

INKREMENTALNI-VR(G,k)
1 $\nu \leftarrow \emptyset$
2 **foreach** $u \in G.V$ **do**
3 $N \leftarrow$ LOWER-NBRS(G,u)
4 ADD-COFACES(G,k,$\{u\}$,N,ν)
5 **return** ν

implementiramo:

```
template<typename Graph>
void INCREMENTAL_VR(const Graph & g,int k) {
VItr vitr,vend;
set<int> N;

for (boost::tie(vitr,vend)=boost::vertices(g);vitr!=vend; vitr++){
// za vsako vozlišče v grafu

    N=LOWER-NBRS(g,*vitr);
    set<int> u;
    u.insert(*vitr);
    ADD_COFACES(g,k,u,N); } }
```

Funkcijo LOWER-NBRS smo že implementirali pri induktivnem algoritmu.

Funkcija ADD-COFACES je rekurzivna funkcija, v katerem doda simplekse k-skeleta.

Psevdokodo funkcije ADD-COFACES

ADD-COFACES(G,k,τ,N,ν)
1 $\nu \leftarrow \nu \cup \{\tau\}$
2 **if** $\dim(\tau) \geq k$ **then return**
3 **else foreach** $\nu \in N$ **do**
4 $\sigma \leftarrow \tau \cup \{v\}$
5 $M \leftarrow N\cap$ LOWER-NBRS(G,v)
6 ADD-COFACES(G,k,σ,M,ν)

implementiramo:

```
template<typename Graph>
void ADD_COFACES(const Graph & g,int k,set<int> T, set<int> N){


set<int> sigma;
set<int> N1,N2;
set<int>::iterator it;
V.insert(T);                    // ν ← ν ∪ {τ}

if(T.size()>=k) return;                    //if dim(τ) ≥ k
else


//foreach ν ∈ N do
for (it=N.begin(); it!=N.end(); it++) {

  sigma=T;                    //σ ← τ ∪ {v}
  sigma.insert(*it);


//M ← N∩ LOWER-NBRS(G,v)
  N1=LOWER(g,*it);
  set_intersection(N.begin(),N.end(),N1.begin(),N1.end(),
  inserter(N2,N2.end()));

//ADD-COFACES(G,k,σ,M,ν)
  ADD_COFACES(g,k,sigma,N2);
  N2.clear(); N1.clear(); }
  N.clear(); }
```

Maksimalni algoritem

Psevdokodo funkcije Bron-Kerbosch(R,P,X):

1 če sta P in X obe prazni:
2   R je maksimalna klika
3 za pivot u izberimo eno od oglišč z najvišjo stopnjo v grafu G
4 za vsako točko $v \in P\setminus$ N(u):
5   Bron-Kerbosch2($R \cup \{v\}$,$P\cap$ N(v),$X\cap$N(v))
6   $P := P\setminus \{v\}$
7   $X := X \cup \{v\}$

implementiramo:

```
typedef boost::graph_traits<Graph>::adjacency_iterator adj_it;

templatet<typename Graph>
void BRON_KERBOSCH(const Graph & g, set<set<int> > & Klika, set<int>
R, set<int> P, set <int> X) {
set<int> sosedi;
set<int> R1,P1,X1;
set<int>::iterator it,it1;


int pivot;
int degree=0; //degree je stopnja vozlišča v grafu
```

//če sta P in X prazna, je R klika ter jo dodamo v množico $Klika$ (množica vseh klik)

```
if (P.empty() && X.empty()) {
Klika.insert(R); return;
else {
```

//iščemo točko v P, ki ima največjo stopnjo v grafu in jo priredimo v *pivot*

```
for (it=P.begin(); it!=P.end(); it++) {
if (degree<boost::degree(*it,g)) {
pivot=*it;
```

```
degree=boost:degree(*it,g); } }
```

//za vsako točko $v \in P \backslash N(u)$

```
for (it=P.begin();it!=P.end();it++) {
if(!((boost::edge(*it,pivot,g).second))) {
```

// $R1 = R \cup \{v\}$

```
R1=R; R.insert(*it);
```

//iščemo sosede za točko v v grafu in jih priredimo množici `sosedi`

```
pair<adj_it,adj_it> neighbors=boost::adjacent_vertices(vertex(*it,g),g);

   for(;neighbors.first!=neighbors.second; neighbors.first++) {
sosedi.insert(*neighbors.first); }
```

//$P1 = P \cap N(v); \quad X1 = X \cap N(v)$

```
set_intersection(P.begin(),P.end();sosedi.begin(),sosedi.end(),
inserter(P1,P1.end()));
set_intersection(X.begin(),X.end();sosedi.begin(),sosedi.end(),
inserter(X1,X1.end()));
```

//rekurzivni klic Bron-Kerbosch z novim argumentom $R1, P1, X1$

```
BRON_KERBOSCH(g,Klika,R1,P1,X1);
P1.clear(); X1.clear();
X.insert(*it);  //X = X ∪ {v}
P.erase(*it); } } }  //P = P\{v}
```

Po algoritmu BRON_KERBOSCH dobimo množico vseh maksimalnih klik in za vsako maksimalno kliko generiramo novo množico vseh podmnožic maksimalne klike. Vsaka maksimalna klika in njihove vse podmnožice dodamo v Vietoris-Ripsov kompleks.

Utežna funkcija za k-skelet Vietoris-Ripsovega kompleksa

COMPUTE-WEIGHTS(ν,ω)
1 **foreach** simplex $\sigma \in \nu$ **do**
2 WEIGHT(σ,ω)
3 **return** ω

WEIGHT(σ,ω)
1 **if** $\omega(\sigma)$ is defined **then**
2 **return** $\omega(\sigma)$
3 **else** **return** $\omega(\sigma) \leftarrow \max_{\tau \subset \sigma}$ WEIGHT(τ,ω)

implementiramo:

```
void COMPUTE_WEIGHT(set<set<int> > & V, map<set<int>, double>
                    & weight) {

set<set<int> >::iterator it;
for (it=V.begin(); it!=V.end(); it++) {
if((*(it).size()>2) {
WEIGHT((*it),weight); } } }


double WEIGHT(set<int> sigma, map<set<int>,double> & weight) {

map<set<int>,double>::iterator it;
it=weight.find(sigma);
if (it!=weight.end()) return weight[sigma];
else {
double max=0;
double t=0;
set<set<int> > sets,sets1;
subsets(sets,sigma); //funkcija subsets priredi množico sets kot
                  //množica vseh podmnožic množice sigma
sets1=sets;
set<set<int> >::iterator it1;
sets1.erase(sigma); //množica sets ne vsebuje sigma zaradi prave
                  //podmnožice. (τ ⊂ σ)
```

```
for(it1=sets.begin(); it1!=sets1.end();it1++) {

  t=WEIGHT(*it1,weight);
  if (max<t) max=t; }

weight[sigma]=max;
return weight[sigma]; } }
```

// Funkcija subsets

```
void subsets(set<set<int> > & sets, set<int> initial) {

if (initial.empty()) return;
sets.insert(initial);
set<int>::iterator it=initial.begin();

for(;it!=initial.end(); it++)
    set<int> new_set(initial);
    new_set.erase(new_set.find(*it));
    subsets(sets,new_sets); } }
```

Literatura:

[1] Bron, C., Kerbosch, J., 1973. ”Algorithm 457: finding all cliques of an undirected graph”, Commun. ACM (ACM) 16 (9): 575–577.

[2] Bron-Kerbosch algorithm. Wikipedia. 2013.
URL: http://en.wikipedia.org/wiki/Bron-Kerbosch_algorithm

[3] Carlsson, E., Carlsson, G., de Silva, V., 2006, ”An algebraic topological method for feature identification”, Int. J. Comput. Geom. Appl. 16 (4): 291–314.

[4] Cazals, F., Karande, C., 2008. A note on the problem of reporting maximal cliques.

[5] Chazal, F., Oudot, S., 2008. ”Towards Persistence-Based Reconstruction in Euclidean Spaces”, ACM Symposium on Computational Geometry: 232–241.

[6] de Berg, M., van Kreveld M., Overmars, M., Schwarzkopf, O., 2000. Computational Geometry: Algorithms and Applications. Springer-Verlag, New York, second edition.

[7] de Silva, V., Carlsson, G., 2004. Topological estimation using witness complexes. IEEE/Eurographics Symposium on Point-Based Graphics, 157-166.

[8] de Silva, V. Ghrist, R., 2006. ”Coordinate-free coverage in sensor networks with controlled boundaries via homology”, The International Journal of Robotics Research 25 (12): 1205–1222.

[9] Edelsbrunner, P., Mücke, E.P., 1994. Three-dimensional alpha shapes. ACM Transactions on Graphics, 13:43-72.

[10] Giesen, J., John, M., 2003. The flow complex: A data structure for geometric modeling. Symposium on Discrete Algorithms, 285-294.

[11] Kunaver, T., 2010. Izločanje nevidnih ploskev v 3D-pogonih. Diplomsko delo. Ljubljana, Fakulteta za računalništvo in informatiko.

[12] Ladner, R., 2002. K-D Trees. CSE 373, Data Structures and Algorithms, Lecture 22. University of Washington.

[13] Morozov, D., 2012. Dionysus.
URL: `http://www.mrzv.org/software/dionysus`.

[14] Mount, D.M., Arya, S., 2010. ANN:A library for approximate nearest neighbor searching.
URL: `http://www.cs.umd.edu/∼mount/ANN/`

[15] Mrčun, J., 2008. Topologija.

[16] Muhammad, A., Jadbabaie, A., 2007, ”Dynamic coverage verification in mobile sensor networks via switched higher order Laplacians”, in Broch, Oliver, Robotics: Science and Systems, MIT Press.

[17] Perry, P., de Silva, V., 2006. PLEX 2.5: Simplicial complexes in MATLAB.
URL: `http://math.stanford.edu/comptop/programs/plex.html`.

[18] Sexton, H., Johansson, M.V., 2009. JPlex.
URL: `http://comptop.stanford.edu/programs/jplex/`.

[19] Siek, J., Lee, L., Lumsdaine, A., 2002. The Boost Graph Library: User Guide and Reference Manual.

[20] Vietoris-Rips complex. Wikipedia. 2013.
URL: `http://en.wikipedia.org/wiki/Vietoris-Rips_complex`

[21] Zomorodian, A., 2010. Fast Construction of the Vietoris-Rips Complex. Computers & Graphics 34(3):263-271.

[22] Zomorodian, A., 2005. Topology for Computing.