Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Kristan

Voronoijevi diagrami

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Neža Mramor Kosta

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V delu predstavite pojem Voronoijevega diagrama. Podrobneje opišite lastnosti ravninskih Voronoijevih diagramov in zvezo med Voronoijevim diagramom in Delaunayevo triangulacijo. Predstavite najbolj znane algoritme za konstrukcijo Voronoijevih diagramov. Implementirajte Fortunov algoritem in preizkusite njegovo delovanje na umetnih podatkih in na podatkih o lokacijah javnih polnilnih mest za električne avtomobile v Sloveniji.

Kazalo

Slike

Povzetek

Naslov: Voronoijevi diagrami

Avtor: Matej Kristan

V diplomski nalogi najprej opišemo definicijo Voronoijevega diagrama in bolj podrobno lastnosti ravninskih Voronoijevih diagramov. Opišemo tudi triangulacijo ravnine, pojem Delaunayeve triangulacije in predstavimo povezavo med njima. Nato predstavimo tri različne algoritme za konstrukcijo ravninskih Voronoijevih diagramov in bolj podrobno pogledamo Fortunov algoritem, ki spada med algoritme s prebirno premico. Algoritmi s prebirno premico so posebej razširjeni v računski geometriji in z njimi rešujemo različne probleme v evklidskem prostoru. Za konstrukcijo Voronoijevega diagrama smo si izbrali Fortunov algoritem, ki smo ga implementirali v programskem jeziku Java. Pravilnost delovanja algoritma smo preverili na točkah, ki predstavljajo lokacije letališč v ZDA, lokacije javnih polnilnih mest za električne avtomobile v Sloveniji in na več naborih naključno generiranih točk.

Ključne besede: Voronoijev diagram, dvodimenzionalen Voronoijev diagram, prebirna premica, računalniška geometrija, Fortunov algoritem, evklidski prostor.

Abstract

Title: Voronoi diagrams

Author: Matej Kristan

In the thesis we first describe the definition of a Voronoi diagram and several properties of Voronoi diagrams in the plane. We also define triangulations of the plane and the concept of a Delaunay triangulation, and present the connection between Voronoi diagrams and Delaunay triangulations. We then present three different algorithms for constructing a Voronoi diagram in the plane, and provide a more detailed description of Fortune's algorithm which is an example of a sweep line algorithm. Sweep line algorithms are especially widespread in computational geometry and are used for solving various problems in Euclidean space. We selected Fortune's algorithm for constructing Voronoi diagrams, and implemented it in the Java programming language. The performance of our implementation of the algorithm was checked on several randomly generated datasets and on a dataset of geographic coordinates of public charging stations for electric cars in Slovenia.

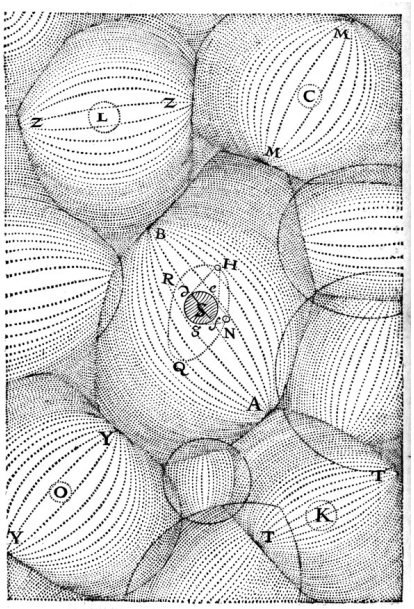
Keywords: Voronoi diagram, two-dimensional Voronoi diagram, sweep line, computational geometry, Fortune's algorithm, euclidean space.

Poglavje 1

Uvod

Voronoijeve diagrame zasledimo v literaturi že zelo zgodaj, precej prej kot večino drugih pojmov računske geometrije, in pod različnimi imeni. René Descartes je leta 1644 v knjigi [5] trdil, da je sončni sistem sestavljen iz konveksnih poligonov, ki jih je poimenoval vrtinci (glej sliko 1.1). Najdemo jih tudi pod imenom, Dirichletova teselacija, po matematiku Petru Gustavu Lejeune Dirichletu, ki je v svojem delu [10] definiral 2-dimenzionalne in 3-diomenzionalne Voronoijeve diangarme in jih uporabil pri preučevanju kvadratnih polinomov. Ime Voronoijevi diagrami pa so dobili po rusko-ukrainskem matematiku Georgyju Voronoyu, ki je leta 1908 v svojem delu [21] definiral pojem splošnega n-dimenzionalnega Voronoijevega diagrama. Voronoijevi diagrami se uporabljajo na različnih področjih. Na primer, v biologiji je George Stewart Brown [3] uporabil Voronoijeve diagrame za opis gostote dreves v gozdu, Roger Mead pa je za opis gostote rastlin uporabil isti koncept na rastlinah in je v svojem delu [12] Voronoijev diagram poimenoval rastlinski poligoni. V [20] je predstavljen širok spekter problemov, ki se lahko rešujejo tudi z Voronoijevimi diagrami.

V prvem delu diplomske naloge najprej predstavimo definicijo Voronoijevega diagrama in podrobno opišemo lastnosti ravninskih Voronoijevih diagramov. Nato predstavimo definicijo triangulacije ravnine in lastnosti Delaunayeve



Slika 1.1: Voronoijev diagram v knjigi [5].

triangulacije. Opišemo tudi povezavo med Voronoijevim diagramom in Delaunayevo triangulacijo, ker sta strukturi dualni, kar pomeni, da lahko preko Delaunayeve triangulacije konstruiramo Voronoijev diagram in obratno.

V drugem delu predstavimo dva pristopa konstrukcije ravninskih Voronoijevih diagramov. Prvi pristop je direktni, kjer direktno konstruiramo Voronoijev diagram, pri drugem pristopu pa najprej konstruiramo Delaunayevo triangulacijo in preko dualnosti zgradimo Voronoijev diagram.

Tretji del diplomske naloge je namenjen opisu algoritma, ki smo ga implementirali. Izbrali smo Fortunov algoritem konstrukcije ravninskih Voronoijevih diagramov, ki spada med algoritme s prebirno premico. Predstavljena sta tudi geometrijska problema, iskanje presečišča dveh premic in iskanje trikotniku očrtane krožnice, ki sta potrebna za delovanje algoritmov, ki smo jih opisali v drugem delu diplomske naloge. Predstavljene so tudi slike Voronoijevih diagramov zgrajenih z implementiranim algoritmom.

V zadnjem, četrtem delu diplomske naloge so sklepne ugotovitve.

Poglavje 2

Voronoijev diagram

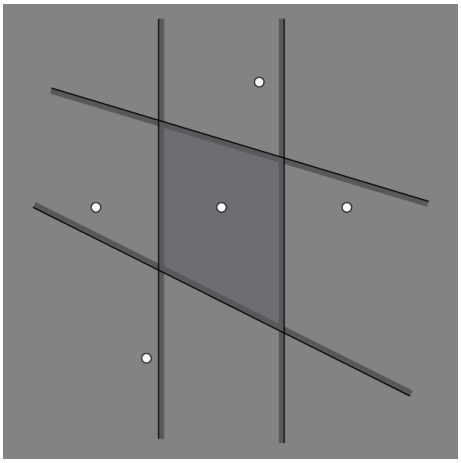
V tem poglavju bomo najprej predstavili Voronoijev diagram in opisali nekaj njegovih lastnosti, ki nam bodo pomagale pri razumevanju konstrukcije diagrama v nadaljevanju. Nato si bomo ogledali Delaunayevo triangulacijo ter opisali povezavo med Voronoijevim diagramom in Delaunayevo triangulacijo.

2.1 Ravninski Voronoijev diagram

Naj bo z $\mathcal{P} = \{p_1, p_2, ..., p_n\}$ označena množica n različnih točk v prostoru $\mathbb{R}^m$ in z $\mathrm{d}(p, q)$ evklidska razdalja med točkama p in q. Voronoijev diagram zgrajen nad točkami iz množice $\mathcal{P}$ označimo $Vor(\mathcal{P})$ in predstavlja razdelitev prostora na n Voronoijevih celic. Vsaki točki $p_i \in \mathcal{P}$ pripada Voronoijeva celica, ki jo označimo z oznako $\mathcal{V}(p_i)$. Točka $q \in \mathbb{R}^m$ leži znotraj Voronoijeve celice $\mathcal{V}(p_i)$ natanko takrat, kadar je

$$\mathrm{d}(q, p_i) \leq \mathrm{d}(q, p_j), \forall p_j \in \mathcal{P}.$$

Če med točkama p_i in p_j narišemo daljico $\overline{p_i p_j}$, hiperravnina, ki daljico razpolavlja in je nanjo pravokotna, prostor razdeli v dva polprostora. Polprostor, ki vsebuje točko p_i označimo s $h(p_i, p_j)$ in polprostor, ki vsebuje točko p_j označimo s $h(p_j, p_i)$. Točka $q \in \mathbb{R}^m$ leži znotraj polprostora $h(p_i, p_j)$ natanko



Slika 2.1: S temnejšo barvo je označen presek polravnin v $\mathbb{R}^2$, ki predstavlja eno Voronoijevo celico.

takrat, kadar je razdalja med točkama p_i in q manjša ali enaka razdalji med točkama p_j in q. Voronoijeva celica $\mathcal{V}(p_i)$ je presek $n-1$ polprostorov.

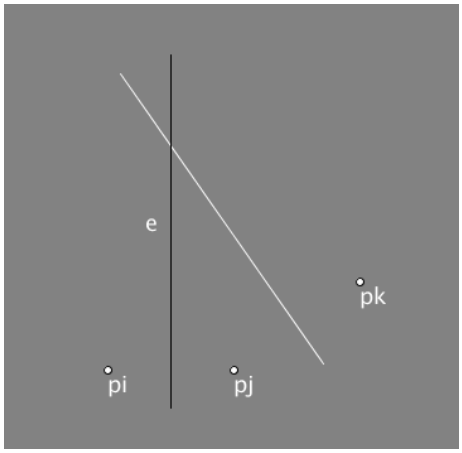
$$\mathcal{V}(p_i) = \bigcap_{j=1}^{n} h(p_i, p_j), i \neq j \tag{2.1}$$

Vsaka Voronoijeva celica ima obliko konveksnega poliedra, ki ni nujno omejen.

V tem delu si bomo podrobneje ogledali ravninske Voronoijeve diagrame. V tem primeru je vsaka Voronoijeva celica konveksen poligon z največ $n-1$ vozlišči in $n-1$ povezavami med vozlišči (glej sliko 2.1).

Izrek 1 (Theorem 7.2, [2])**.** *Naj bo $\mathcal{P}$ množica n točk. Če so vse točke kolinearne, potem povezave v $Vor(\mathcal{P})$ tvorijo množico $n-1$ vzporednih premic. Drugače tvorijo povezave $Vor(\mathcal{P})$ povezan graf sestavljen iz daljic ali poltrakov.*

Dokaz. Prvi del izreka je očiten. Da dokažemo drugi del, moramo pokazati, da med povezavami $Vor(\mathcal{P})$ ni nobene premice. Predpostavimo, da obstaja premica e, ki je simetrala na daljico $\overline{p_i p_j}$ in meji na Voronoijevi celici $\mathcal{V}(p_i)$



Slika 2.2: Premica e, ki je simetrala na daljico $\overline{p_i p_j}$ in simetrala na daljico $\overline{p_j p_k}$.

in $\mathcal{V}(p_j)$. Naj bo p_k točka, ki ni kolinearna s točkama p_i in p_j (glej sliko 2.2). Simetrala na daljico $\overline{p_j p_k}$ ni vzporedna s premico e zato jo mora sekati. Del premice e, ki leži znotraj $h(p_k, p_j)$ ne more mejiti na $\mathcal{V}(p_j)$, ker leži bližje točki p_k kot točki p_j. Protislovje. □

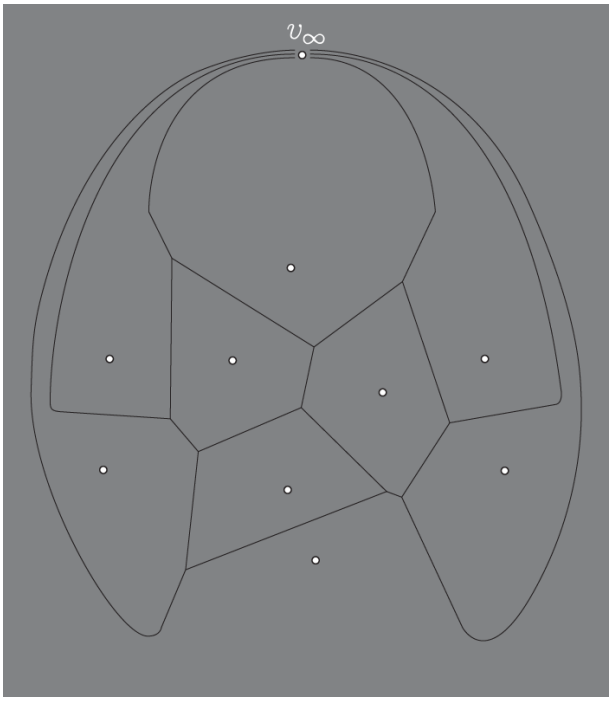
Ravninski Voronoijev diagram je unija vseh Voronoijevih celic, ki skupaj sestavljajo celotno ravnino.

Izrek 2 (Theorem 7.3, [2])**.** *Ravninski Voronoijev diagram zgrajen nad množico točk, ki vsebuje $n \geq 3$ točk, ima največ $2n-5$ vozlišč in največ $3n-6$ povezav.*

Dokaz. Izrek bomo pokazali s pomočjo Eulerjeve formule za povezane ravninske grafe [22]. Označimo z m_v število vozlišč, z m_e število povezav in z m_f število območij v ravnini, ki jih povezave med seboj ločujejo, in zapišimo Eulerjevo formulo.

$$m_v - m_e + m_f = 2 \tag{2.2}$$

Voronoijev diagram ni ravninski graf, saj ima poleg povezav tudi poltrakove. Zato med vozlišča dodamo vozlišče v_∞, ki se nahaja v neskončnosti in povežemo vse poltrake z vozliščem v_∞. Sedaj je Voronoijev diagram povezan



Slika 2.3: Voronoijev diagram z dodanim vozliščem v_∞.

ravninski graf in zanj velja Eulerjeva formula (glej sliko 2.3). Iz formule sledi relacija med številom vozlišč Voronoijevega diagrama n_v, številom povezav med vozlišči n_e in številom Voronoijevih celic n.

$$(n_v + 1) - n_e + n = 2 \tag{2.3}$$

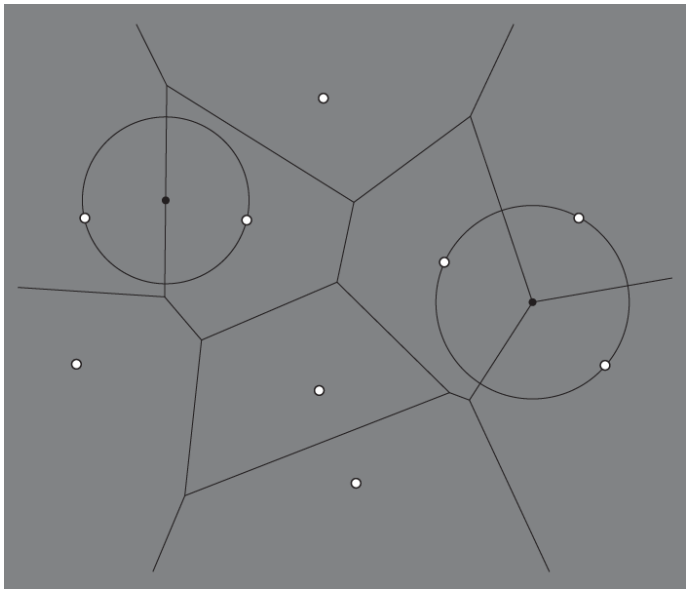
Vsaka povezava v planarnem grafu ima točno dve vozlišči. Če seštejemo vse stopnje vozlišč, dobimo dvakratno število povezav n_e in vsako vozlišče vključno z v_∞ ima stopnjo vozlišča vsaj 3. Od tod sledi naslednja neenačba

$$2n_e \geq 3(n_v + 1). \tag{2.4}$$

Iz neenačbe 2.4 izpostavimo n_e in ga vstavimo v enačbo 2.3.

$$(n_v + 1) - \frac{3}{2}(n_v + 1) + n \geq 2$$

$$-n_v - 1 + 2n \geq 4$$



Slika 2.4: Primer krožnice $C_{\mathcal{P}}(q)$ s tremi točkami (na desni strani) na krožnici in $C_{\mathcal{P}}(q)$ z dvema točkama (na levi strani) na krožnici.

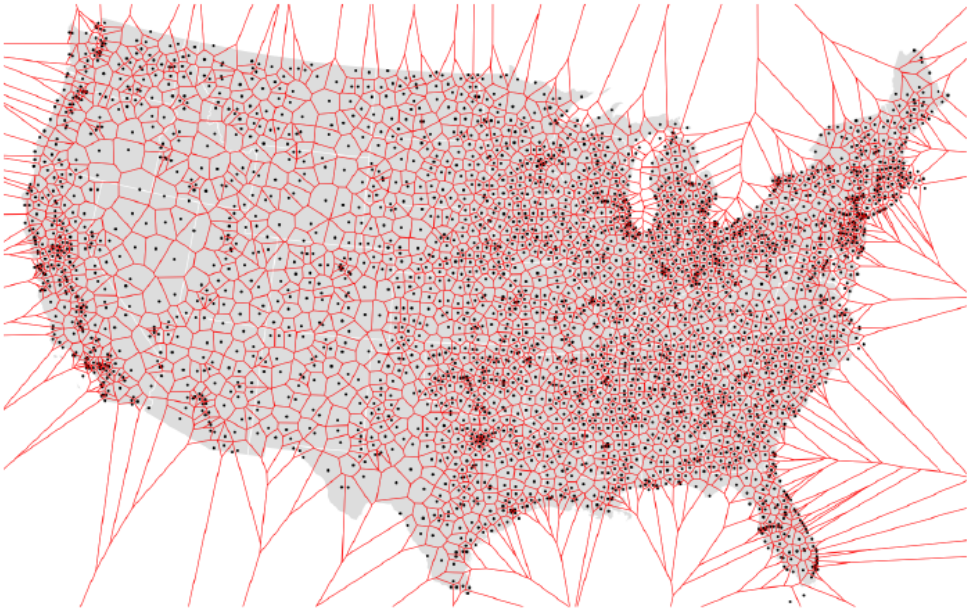
$$n_v \leq 2n - 5$$

Na podoben način lahko iz neenačbe 2.4 izpostavimo n_v in ga vstavimo v enačbo 2.3, da dokažemo drugo neenakost v izreku 2. □

Povezave $Vor(\mathcal{P})$ ležijo na simetralah med pari točk in vozlišča $Vor(\mathcal{P})$ so presečišča med simetralami. Če množica $\mathcal{P}$ vsebuje n točk, je vseh različnih parov točk $\frac{n(n-1)}{2}$, zato je toliko tudi simetral. Vse simetrale in vsa presečišča pa niso del Voronoijevega diagrama. Da opišemo povezave in vozlišča $Vor(\mathcal{P})$ za točko q definiramo krožnico s središčem v q, znotraj katere ne leži nobena točka iz množice $\mathcal{P}$ in jo označimo z oznako $C_{\mathcal{P}}(q)$. Primer $C_{\mathcal{P}}(q)$ je prikazan na sliki 2.4.

Izrek 3 (Theorem 7.4, [2])**.** *Za povezave in vozlišča Voronoijevega diagrama $Vor(\mathcal{P})$ veljata naslednji dve trditvi.*

1. *Točka q leži na povezavi med Voronoijevima celicama $\mathcal{V}(p_i)$ in $\mathcal{V}(p_j)$ natanko takrat, kadar na krožnici $C_{\mathcal{P}}(q)$ ležita točki p_i in p_j.*

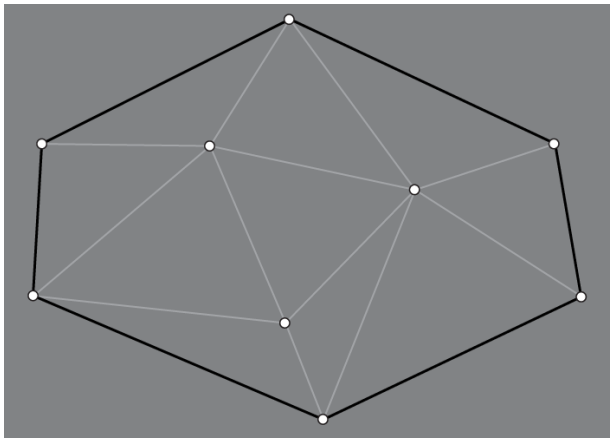


Slika 2.5: Voronoijev diagram zgrajen nad točkami, ki predstavljajo lokacije letališč v Združenih državah Amerike. Slika je iz [16].

2. *Točka q je vozlišče Voronoijevega diagrama natanko takrat, kadar na krožnici $C_{\mathcal{P}}(q)$ ležijo tri ali več točk iz množice $\mathcal{P}$.*

Dokaza izreka ne bomo navajali, saj presega obseg tega dela. Najdemo ga v [2].

Na sliki 2.5 je prikazan ravninski Voronoijev diagram zgrajen nad točkami, ki predstavljajo lokacije letališč v Združenih državah Amerike. Za približno oceno števila potencialnih obiskovalcev posameznega letališča predpostavimo, da so cene letalskih vozovnic na vseh letališčih enake. Potem lahko predvidimo, da bo posamezna stranka obiskala letališče, ki je njej najbližje. S takšnim modelom, za letališče, ki je predstavljeno s točko p, ocenimo približno število obiskovalcev s pomočjo Voronoijeve celice $\mathcal{V}(p)$. Vsi potencialni obiskovalci naj bi živeli znotraj Voronoijeve celice $\mathcal{V}(p)$.



Slika 2.6: S črno črto je označena konveksna ogrinjača točk, s svetlejšo sivo pa stranice triangulacije.

2.2 Delaunayeva triangulacija ravnine

V tem razdelku bomo predstavili pojem triangulacije ravnine in bolj podrobno opisali Delaunayevo triangulacijo, ki je tesno povezana z Voronoijevim diagramom.

Triangulacijo točk množice $\mathcal{P}$ označimo s $\mathcal{T}(\mathcal{P})$. Če točke v množici $\mathcal{P}$ ne ležijo na isti premici, je triangulacija $\mathcal{T}(\mathcal{P})$ množica trikotnikov, ki se med seboj ne prekrivajo in zapolnijo celotno površino konveksne ogrinjače točk $\mathcal{P}$ (glej sliko 2.6). Konveksna ogrinjača množice točk $\mathcal{P}$ je najmanjša podmnožica točk $\mathcal{P}$, ki tvorijo konveksen poligon, znotraj katerega ležijo vse točke iz množice $\mathcal{P}$. Vsak trikotnik t_{ijk} iz triangulacije je sestavljen iz treh nekolinearnih točk $p_i, p_j, p_k \in \mathcal{P}$ in vsaka triangulacija na istih točkah je sestavljena iz enakega števila trikotnikov in povezav. Število trikotnikov in povezav v triangulaciji je odvisno od števila točk v množici $\mathcal{P}$ in od števila točk, ki ležijo na konveksni ogrinjači točk $\mathcal{P}$.

Izrek 4 (Theorem 9.1, [2])**.** *Naj bo $\mathcal{P}$ množica n različnih točk, ki niso kolinearne in naj bo k število točk, ki ležijo na konveksni ogrinjači točk $\mathcal{P}$.*

Potem je triangulacija $\mathcal{T}(\mathcal{P})$ sestavljena iz $2n-2-k$ trikotnikov in $3n-3-k$ povezav.

Dokaz. Naj bo $\mathcal{T}(\mathcal{P})$ triangulacija sestavljena iz m trikotnikov. Vseh območij v ravnini, ki jih povezave med seboj ločujejo je $m+1$ in vseh vozlišč v triangulaciji je n. Če to vstavimo v Eulerjevo enačbo 2.2 dobimo

$$n - m_e + (m+1) = 2. \tag{2.5}$$

Vsa območja v ravnini so trikotniki in imajo tri povezave, razen neomejenega območja, ki ga omejuje k povezav. Vsaka povezava meji na dve območji v ravnini, iz tega sledi naslednja enačba

$$3m + k = 2m_e. \tag{2.6}$$

Iz enačbe 2.6 izpostavimo m_e in ga vstavimo v enačbo 2.5.

$$n - \frac{3m+k}{2} + m = 1$$

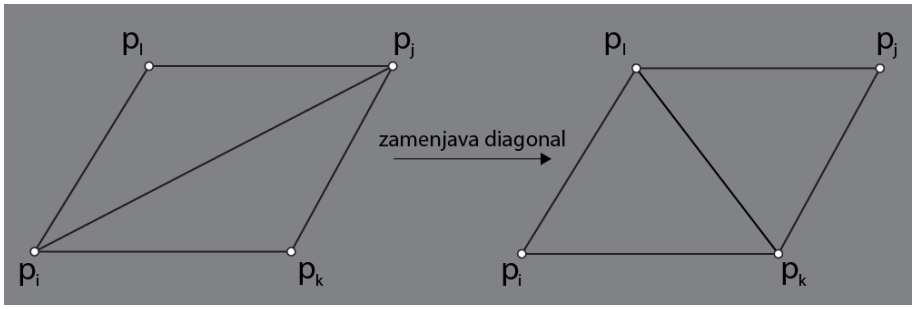
$$2n - 3m - k + 2m = 2$$

$$m = 2n - 2 - k$$

Na podoben način lahko iz enačbe 2.6 izpostavimo m in ga vstavimo v enačbo 2.5, da dokažemo drugo enakost v izreku 4.

□

Delaunayeva triangulacija je poseben primer triangulacije in jo označimo $\mathcal{DT}(\mathcal{P})$. Poimenovana je po ruskem matematiku Borisu Nikolajeviču Delaunayu, ki jo je definiral leta 1934. Naj bo triangulacija $\mathcal{T}(\mathcal{P})$ sestavljena iz m trikotnikov. Iz vsakega trikotnika v triangulaciji vzamemo samo najmanjši notranji kot in jih razvrstimo v naraščajočem vrstnem redu v seznam kotov $\alpha_1 \leq \alpha_2 \leq ... \leq \alpha_m$. Urejen seznam kotov triangulacije $\mathcal{T}(\mathcal{P})$ poimenujemo vektor kotov in ga označimo $\mathcal{A}(\mathcal{T}) = (\alpha_1, \alpha_2, ..., \alpha_m)$. Naj bo $\mathcal{T}'(\mathcal{P})$ drugačna triangulacija na istih točkah in pripadajoč vektor kotov $\mathcal{A}(\mathcal{T}') = (\alpha'_1, \alpha'_2, ..., \alpha'_m)$. Vektor $\mathcal{A}(\mathcal{T})$ je večji kot vektor $\mathcal{A}(\mathcal{T}')$, če je leksikografsko večji od vektorja $\mathcal{A}(\mathcal{T}')$, kar označimo kot $\mathcal{A}(\mathcal{T}) > \mathcal{A}(\mathcal{T}')$. Za



Slika 2.7: Prikazuje zamenjavo diagonal dveh sosednjih trikotnikov, ki skupaj tvorita konveksen štirikotnik.

triangulacijo $\mathcal{T}(\mathcal{P})$ rečemo, da je Delaunayeva triangulacija $\mathcal{DT}(\mathcal{P})$ natanko takrat, kadar je vektor kotov triangulacije $\mathcal{A}(\mathcal{T})$ večji ali enak od vektorja kotov poljubne druge triangulacije na $\mathcal{P}$.

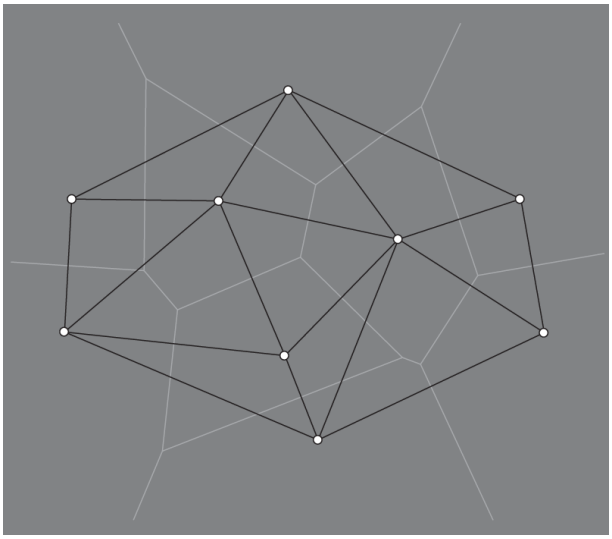
Delaunayeva triangulacija $\mathcal{DT}(\mathcal{P})$ je tista triangulacija točk $\mathcal{P}$, ki med vsemi možnimi triangulacijami $\mathcal{T}(\mathcal{P})$ vsebuje trikotnike z največjimi možnimi najmanjšimi notranjimi koti. Zato rečemo, da Delaunayeva triangulacija maksimizira najmanjše notranje kote trikotnikov v triangulaciji [9].

Kadar si trikotnika t_{ijk} in t_{ijl} iz triangulacije $\mathcal{T}(\mathcal{P})$ delita skupno povezavo $\overline{p_i p_j}$ in skupaj tvorita konveksen štirikotnik, lahko z zamenjavo diagonale $\overline{p_i p_j}$ z diagonalo $\overline{p_k p_l}$ dosežemo, da se bosta najmanjša notranja kota obeh trikotnikov povečala (glej sliko 2.7). Če za vse pare trikotnikov, ki skupaj predstavljajo konveksne štirikotnike na ta način maksimiziramo najmanjše notranje kote, lahko poljubno triangulacijo z zamenjevanjem diagonal preoblikujemo v Delaunayevo triangulacijo.

Opišimo trikotnike, ki tvorijo Delaunayevo triangulacijo še drugače.

Izrek 5 (Theorem 9.7, [2])**.** *Tri točke $p_i, p_j, p_k \in \mathcal{P}$ predstavljajo trikotnik $\mathcal{DT}(\mathcal{P})$ natanko takrat, kadar krožnica skozi točke p_i, p_j, p_k ne vsebuje nobene točke iz množice $\mathcal{P}$ v svoji notranjosti.*

Dokaz izreka si lahko pogledamo v [2].



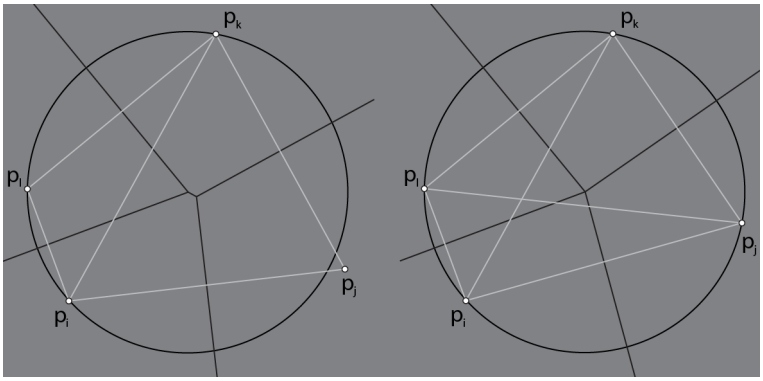
Slika 2.8: Prikazuje s črno Delaunayevo triangulacijo $\mathcal{DT}(\mathcal{P})$ in s svetlo sivo Voronoijev diagram $Vor(\mathcal{P})$.

Opišimo še povezavo med Voronoijevim diagramom $Vor(\mathcal{P})$ in Delaunayevo triangulacijo $\mathcal{DT}(\mathcal{P})$. Stranice Voronoijevega diagrama in Delaunayeve triangulacije predstavljajo dualna grafa [18], kar pomeni, da lahko iz Voronoijevega diagrama konstruiramo Delaunayevo triangulacijo in obratno.

Točki $p_i, p_j \in \mathcal{P}$ sta Voronoijevi sosedi, če si pripadajoči Voronoijevi celici $\mathcal{V}(p_i)$ in $\mathcal{V}(p_j)$ v Voronoijevem diagramu $Vor(\mathcal{P})$ delita skupno povezavo.

Če v množici točk $\mathcal{P}$ povežemo vse Voronoijeve sosede, dobimo Delaunayevo triangulacijo konveksne ogrinjače množice točk $\mathcal{P}$. Povezave med Voronoijevimi sosedami, ki si delijo poltrakove v Voronoijevem diagramu, skupaj tvorijo konveksno ogrinjačo točk $\mathcal{P}$. Ostale povezave med Voronoijevimi sosedami pa so povezave med sosednjimi trikotniki v Delaunayevi triangulaciji (glej sliko 2.8).

Težave nastanejo, ko štiri točke, ki predstavljajo vozlišča sosednjih trikotnikov v triangulaciji $\mathcal{T}(\mathcal{P})$ ležijo na isti krožnici. V tem primeru se štiri Voronoijeve celice sekajo v središču krožnice, pripadajoče točke pa tvorijo tetivni štirikotnik. Ne glede na izbiro diagonal oba para trikotnikov izpol-



Slika 2.9: Levo so prikazane štiri točke, ki skoraj ležijo na isti krožnici in nad njimi zgrajen Voronoijev diagram in Delaunayeva triangulacija. Desno pa so prikazane štiri točke, ki ležijo na isti krožnici. Prikazan je tudi Voronoijev diagram katerega povezava se je skrčila v točko in dve različni Delaunayevi triangulaciji teh štirih točk.

njujeta pogoje izreka 5 (glej sliko 2.9). Delaunayeva triangulacija v tem primeru ni natanko določena, ampak obstajata dve (ali več) različni Delaunayevi triangulaci na točkah množice $\mathcal{P}$. Zaradi te težave večina implementacij algoritmov za konstrukcijo Delaunayeve triangulacije zahteva, da so točke v „splošni legi“ oziroma, da nobene štiri točke ne ležijo na isti krožnici.

Poglavje 3

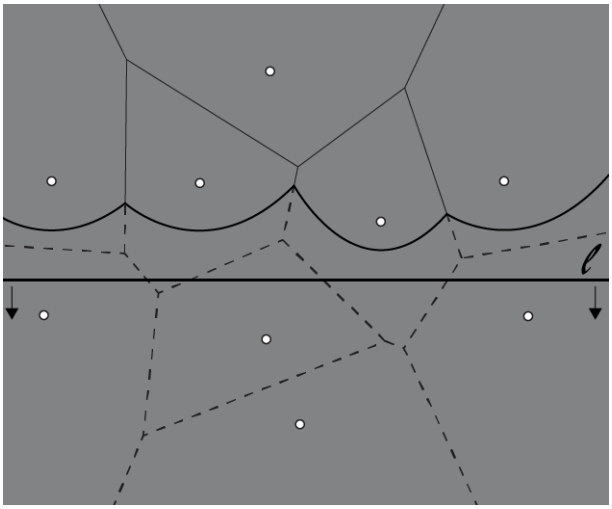
Konstrukcija ravninskega Voronoijevega diagrama

V tem poglavju si bomo ogledali različne algoritme konstrukcije ravninskih Voronoijevih diagramov. Vse opisane konstrukcije so povzete po knjigi [2].

Poznamo dva pristopa h konstrukciji Voronoijevega diagrama. Prvi je direktni pristop, kjer iz množice točk $\mathcal{P}$ direktno konstruiramo Voronoijev diagram $Vor(\mathcal{P})$, pri drugem pa najprej konstruiramo Delaunayevo triangulacijo $\mathcal{DT}(\mathcal{P})$, preko katere zgradimo dualni Voronoijev diagram $Vor(\mathcal{P})$. Časovna zahtevnost je v obeh primerih $\mathcal{O}(n \log n)$.

3.1 Direktni pristop

Med direktnimi pristopi konstrukcije Voronoijevega diagrama $Vor(\mathcal{P})$ je najbolj znan Fortunov algoritem, ki je dobil ime po Stevenu Fortuneju, ki ga je objavil leta 1986 v [8] in ima časovno zahtevnost $\mathcal{O}(n \log n)$. Ideja drugega algoritma pa je, da za vsako točko $p_i \in \mathcal{P}$, Voronoijevi celici $\mathcal{V}(p_i)$ poiščemo presek $n - 1$ polravnin. Najhitrejši algoritmi iskanja preseka n polravnin imajo časovne zahtevnosti $\mathcal{O}(n \log n)$ [2] in ker imamo v Voronoijevem diagramu n Voronoijevih celic sledi, da imajo takšni algoritmi časovne zahtevnosti $\mathcal{O}(n^2 \log n)$.

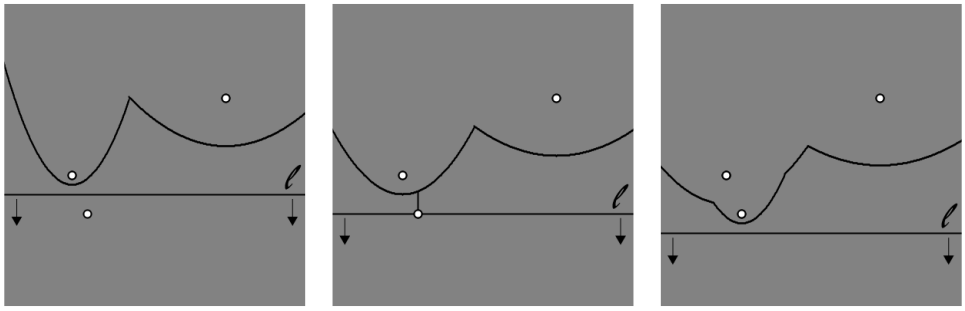


Slika 3.1: Prebirna premica je označena s črko l in nad njo je mejna krivulja, ki je sestavljena iz odsekov parabol. Sosednji paraboli se stikata točno na povezavi Voronoijevega diagrama in sčasoma izrišeta celotno povezavo med Voronoijevima celicama.

Poznamo tudi Lloydov algoritem, ki je dobil ime po Stuartu Lloydu [11]. Algoritem je iterativen, kar pomeni, da z vsako iteracijo dobimo boljši približek končne strukture. Najprej zgradimo $Vor(\mathcal{P})$, potem za vsako celico $\mathcal{V}(p_i)$ izračunamo težišče in na koncu premaknemo točko p_i v težišče celice $\mathcal{V}(p_i)$. Algoritma ne bomo opisali bolj podrobno, ker ni namenjen konstrukciji splošnega Voronoijevega diagrama ampak konstrukciji $Vor(\mathcal{P})$, kjer vsaka točka $p_i \in \mathcal{P}$ predstavlja težišče Voronoijeve celice $\mathcal{V}(p_i)$ [7].

3.1.1 Fortunov algoritem

Fortunov algoritem je algoritem s prebirno premico (angl. sweep line algorithm). Te vrste algoritmi so posebej razširjeni v računski geometriji pri reševanju različnih problemov v evklidskem prostoru. V ravnini se uporabljajo prebirne premice, v prostoru pa prebirne ravnine. Ideja takšnih algoritmov je, da od vrha ravnine proti dnu premikamo vodoravno prebirno



Slika 3.2: Prikazuje dogodek točke, ko prebirna premica doseže novo točko in se v mejni krivulji pojavi nova parabola.

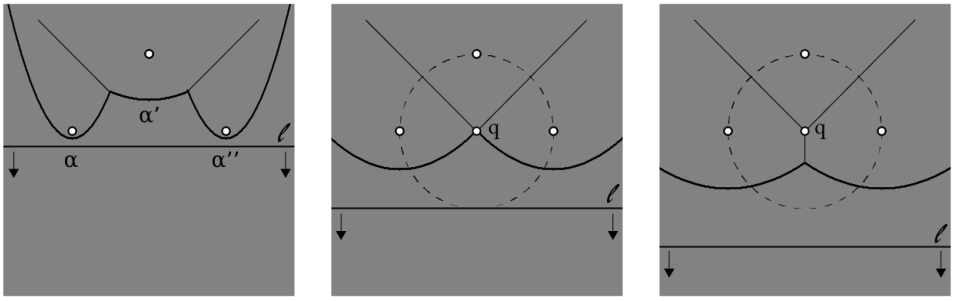
premico in sledimo presečiščem strukture, ki jo trenutno konstruiramo, s prebirno premico. Med premikanjem prebirne premice se podatkovna struktura ne spreminja. Podatkovna struktura se spremeni le, ko višina prebirne premice doseže točno določene točke, tako imenovane točke dogodkov.

V primeru Fortunovega algoritma imamo opravka še z mejno krivuljo (angl. beach line), ki jo označimo s črko $\mathcal{L}$. Mejna krivulja je sestavljena iz odsekov parabol, ki skupaj tvorijo območje nad katerim je Voronoijev diagram dokončno določen. Mejna krivulja $\mathcal{L}$ je graf funkcije, kar pomeni, da vsaka navpična črta seka mejno krivuljo samo v eni točki. Slika 3.1 prikazuje prebirno premico in mejno krivuljo $\mathcal{L}$, ki je sestavljena iz odsekov parabol. Oblika mejne krivulje se spremeni v dveh različnih dogodkih.

Prvi dogodek imenujemo dogodek točke, ki se zgodi, ko višina prebirne premice doseže točko $p \in \mathcal{P}$ (glej sliko 3.2). Za lažje razumevanje opišimo prva dva taka dogodka in sicer, ko višina prebirne premice doseže prvo točko dogodka $p_1 \in \mathcal{P}$ se v prazno mejno krivuljo $\mathcal{L}$ doda točka p_1. Dodana točka predstavlja parabolo, na kateri ležijo vse takšne točke, ki so enako oddaljene od prebirne premice na višini l_y in od točke $p_i = (p_{i,x}, p_{i,y})$, in ima obliko

$$y = \frac{1}{2(p_{i,y} - l_y)}(x^2 - 2p_{i,x}x + p_{i,x}^2 + p_{i,y}^2 - l_y^2). \tag{3.1}$$

Ko višina prebirne premice doseže drugo točko $p_2 \in \mathcal{P}$, se v mejno krivuljo $\mathcal{L}$ doda točka p_2 in to tako, da mejna krivulja, ki je vsebovala samo eno parabolo



Slika 3.3: Prikazuje dogodek kroga, ko prebirna premica doseže najnižjo točko krožnice in se parabola α' skrči v točko.

$\mathcal{L} = \{p_1\}$, sedaj vsebuje tri odseke parabol $\mathcal{L} = \{p_1, p_2, p_1\}$. Nova parabola razdeli staro parabolo na dva odseka, ki sta predstavljena z isto parabolo, k mejni krivulji pa prispeva odsek nove parabole med presečiščema s sosednjima parabolama. Torej vsakič, ko prebirna premica doseže novo točko $p \in \mathcal{P}$ se v mejno krivuljo doda nova parabola, predstavljena s točko p, stara parabola, ki leži navpično nad točko p pa se razdeli v dva odseka parabol med katerima leži nova parabola.

Drugi primer, kjer se oblika mejne krivulje spremini, imenujemo dogodek kroga. Naj bodo α, α' in α'' sosednje parabole, ki so predstavljene s tremi različnimi točkami $p_i, p_j, p_k \in \mathcal{P}$. Če se presečišče med parabolama α in α' s premikanjem prebirne premice bliža presečišču med parabolama α' in α'', se lahko odsek parabole α' sčasoma skrči v točko in izpade iz mejne krivulje. Tak dogodek imenujemo potencialni dogodek kroga. Če skozi točke p_i, p_j, p_k narišemo krožnico $\mathcal{C}_\mathcal{P}(q)$, ko prebirna premica doseže najnižjo točko na krožnici $\mathcal{C}_\mathcal{P}(q)$ in se v mejni krivulji med odseke parabol $\alpha, \alpha', \alpha''$ ne doda nova parabola, ki bi jo predstavljala točka $p_l \in \mathcal{P}$, se sredinski odsek parabole α' skrči v točko in izgine iz mejne krivulje. Središče krožnice $\mathcal{C}_\mathcal{P}(q)$ predstavlja vozlišče Voronoijevega diagrama (glej sliko 3.3).

Algoritem sledi presečiščem med sosednjimi parabolami v mejni krivulji, ki s premikanjem prebirne premice sčasoma izrišejo povezave, katere tvorijo

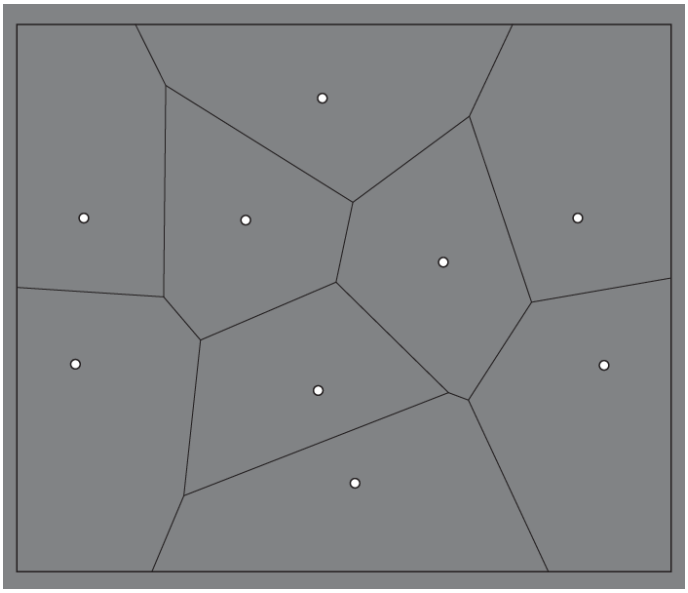
Voronoijev diagram (glej sliko 3.1).

Oglejmo si še v kakšnem vrstnem redu algoritem obdeluje dogodke. Algoritem vsebuje prioritetno vrsto dogodkov, ki jo označimo z oznako $\mathcal{Q}$. Prioritetna vrsta mora ohranjati pravilen vrstni red dogodkov, ne glede na čas, ko je bil dogodek dodan v prioritetno vrsto. Vrstni red dogodkov je urejen po naraščajočih y koordinatah točk dogodkov. Če imata dogodka enaki y koordinati sta urejena še po naraščajoči x koordinati. Koordinatno izhodišče je v zgornjem levem kotu, koordinata x narašča v smeri vodoravno od leve proti desni, koordinata y pa narašča v smeri navpično navzdol. Dogodki točk so znani vnaprej, zato na začetku prioritetna vrsta vsebuje vse dogodke točk. Dogodki krogov pa se zaznavajo in dodajajo v prioritetno vrsto $\mathcal{Q}$ sproti med izvajanjem algoritma. Algoritem se v glavni zanki izvaja toliko časa dokler v prioritetni vrsti $\mathcal{Q}$ obstaja kakšen od dogodkov. Opišimo algoritem s sledečo psevdo kodo.

```
vhod: množica točk v ravnini P
izhod: Voronoijev diagram podan kot množica daljic
voronoijevDiagram(P):
1.    v prioritetno vrsto Q dodamo točke iz množice P, ki
      predstavljajo vse dogodke točk
2.    while (prioritetna vrsta Q ni prazna)
3.        do (odstrani dogodek z najmanjšo y koordinato iz Q)
4.        if (dogodek je dogodek točke)
5.            then obdelajDogodekTočke(p_i)
6.            else obdelajDogodekKroga(e)
7.    obdelajVsePolTrake()
```

Funkcija *obdelajDogodekTočke(p_i)* poskrbi za dogodek točke, ki ga predstavlja točka p_i in naredi sledeče korake.

- Če mejna krivulja $\mathcal{L}$ ni prazna preskoči ta korak, sicer v prazno mejno krivuljo doda novo parabolo, predstavljeno s točko p_i in zaključi s klicem funkcije *obdelajDogodekTočke(p_i)*.



Slika 3.4: Slika prikazuje Voronoijev diagram znotraj pravokotnika.

- V mejni krivulji $\mathcal{L}$ poišče parabolo α, ki leži navpično nad točko p_i. Če parabola α predstavlja že obstoječ dogodek kroga v vrsti dogodkov $\mathcal{Q}$, ga izbriše iz $\mathcal{Q}$. V mejno krivuljo doda točko p_i, ki predstavlja odsek nove parabole tako, da razdeli parabolo α na dva odseka parabol. Če je parabola α v mejni krivulji predstavljena s točko p_j, doda novo povezavo v Voronoijev diagram, ki jo bosta presečišči z novo parabolo in sosednjima odsekoma parabolame sčasoma izrisali med $\mathcal{V}(p_i)$ in $\mathcal{V}(p_j)$.

- Če levi odsek parabole α predstavlja potencialen dogodek kroga, doda nov dogodek v vrsto $\mathcal{Q}$. Med nov dogodek in levi odsek parabole α doda kazalca med njima. Prav tako preveri desni odsek parabole α in po potrebi doda nov dogodek v $\mathcal{Q}$.

Funkcija *obdelajDogodekKroga(e)* poskrbi za dogodek kroga e in naredi sledeča koraka.

- Izbriše parabolo α, ki predstavlja dogodek kroga e iz mejne krivulje $\mathcal{L}$ in zaključi dve povezavi v Voronoijevem diagramu, ki se končata v

središču kroga.

- Če leva sosednja parabola α predstavlja potencialen dogodek kroga, doda nov dogodek v vrsto $\mathcal{Q}$. Med nov dogodek in levi odsek parabole α doda kazalca med njima. Prav tako preveri desno sosedo parabole α in po potrebi doda nov dogodek v $\mathcal{Q}$.

Ko algoritem obdela vse dogodke, je zgradil vse povezave v Voronoijevem diagramu, ki so daljice. Povezave, ki so poltrakovi, zgradi tako, da celoten Voronoijev diagram postavi v pravokotnik in skrajša poltrakove v daljice, da se zaključijo ravno na robu pravokotnika (glej sliko 3.4).

3.1.2 Presek polravnin

V tem razdelku bomo predstavili algoritem, ki Voronoijev diagram konstruira tako, da poišče preseke n polravnin. V nadaljevanju bomo opisali, kako s programerskim pristopom „deli in vladaj“ učinkovito poiščemo presek n polravnin.

Naj bo $\mathcal{H} = \{h_1, h_2, ..., h_n\}$ množica n različnih polravnin. Polravnino h_i lahko predstavimo z linearno neenačbo

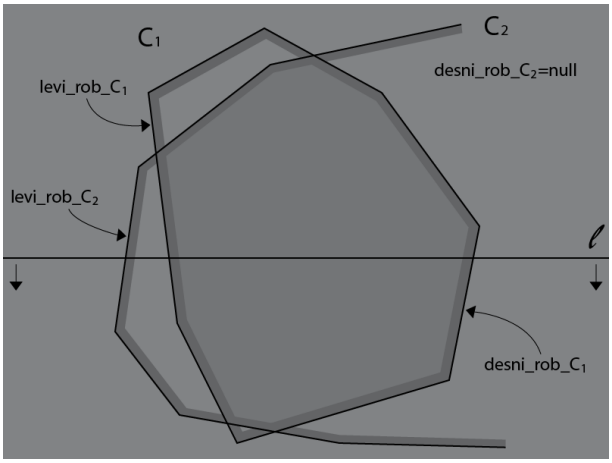
$$a_i x + b_i y \leq c_i,$$

kjer so $a_i, b_i, c_i \in \mathbb{R}$ konstante. Konstanti a_i in b_i ne smeta biti hkrati enaki nič. Tako predstavljena polravnina je omejena s premico $a_i x + b_i y = c_i$. Presek n polravnin je nabor vseh točk, ki ustrezajo n neenačbam hkrati.

Presek polravnin je konveksen poligon, ki ni nujno omejen in ima največ n stranic. Vse stranice poligona so deli premic, ki omejujejo polravnine. Oglišča poligona so presečišča med premicami, ki omejujejo polravnine.

S psevdo kodo algoritma opišimo iskanje preseka n polravnin.

```
vhod: množica polravnin H
izhod: konveksen poligon C, ki predstavlja presek polravnin
presekPolravnin(H):
```



Slika 3.5: Presečišča prebirne premice s povezavami $levi\_rob\_\mathcal{C}_1$, $desni\_rob\_\mathcal{C}_1$, $levi\_rob\_\mathcal{C}_2$, $desni\_rob\_\mathcal{C}_2$ in s svetlo sivo pobarvan presek poligona $\mathcal{C}_1$ in poligona $\mathcal{C}_2$.

```
1.    if (številoPolravnin(ℋ) == 1)
2.        then 𝒞 = edina polravnina iz množice ℋ
3.        else razdeli množico ℋ na dve podmnožici ℋ1 in ℋ2
4.            𝒞1 = presekPolravnin(ℋ1)
5.            𝒞2 = presekPolravnin(ℋ2)
6.            𝒞 = presekKonveksnihPoligonov(𝒞1, 𝒞2)
7.    return 𝒞
```

Opišimo še funkcijo $presekKonveksnihPoligonov(\mathcal{C}_1, \mathcal{C}_2)$, ki vrne konveksen poligon $\mathcal{C}$, ki predstavlja presek dveh konveksnih poligonov $\mathcal{C}_1$ in $\mathcal{C}_2$. Tako kot Fortunov algoritem, tudi ta spada med algoritme s prebirno premico. Za razliko od Fortunovega algoritma so v tem primeru vsi dogodki znani vnaprej. Točke dogodkov so oglišča povezav, ki jih prebirna premica trenutno seka v vsakem konveksnem poligonu. Dogodki se obdelujejo po naraščajočih y koordinatah in, če imata dva dogodka enaki y koordinati, po naraščajočih x koordinatah. Izhodišče koordinatnega sistema je ravno tako kot pri Fortunovem algoritmu v zgornjem levem kotu. Ker prebirna premica seka konveksen poligon v največ dveh točkam bo algoritem potre-

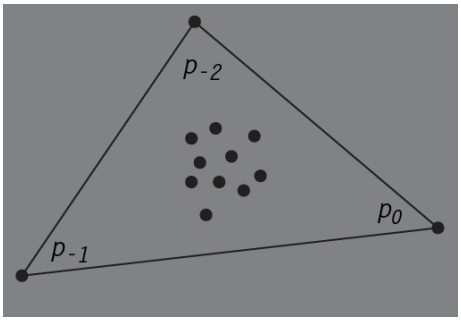
boval štiri kazalce na povezave konveksnih poligonov, ki trenutno sekajo prebirno premico in sicer po dva kazalca za vsak poligon, ki jih bomo imenovali $levi\_rob\_\mathcal{C}_1$, $desni\_rob\_\mathcal{C}_1$, $levi\_rob\_\mathcal{C}_2$ in $desni\_rob\_\mathcal{C}_2$. Če prebirna premica ne seka levega ali desnega roba poligona ga inicializiramo *null* (glej sliko 3.5).

Z y_1 označimo ordinato zgornjega oglišča poligona $\mathcal{C}_1$, če ima poligon $\mathcal{C}_1$ navzgor neomejeno povezavo inicializiramo $y_1 = -\infty$. Enako definiramo y_2 za poligon $\mathcal{C}_2$. Ob klicu funkcije se prebirna premica nahaja na višini $l_y = \max(y_1, y_2)$ in inicializira kazalce na povezave $levi\_rob\_\mathcal{C}_1$, $desni\_rob\_\mathcal{C}_1$, $levi\_rob\_\mathcal{C}_2$ in $desni\_rob\_\mathcal{C}_2$, ki jih prebirna premica l_y seka.

Z vsakim dogodkom ima algoritem opravka z novo povezavo e, ki se začne v točki p in predstavlja oglišče poligona. Najprej preveri kateremu poligonu pripada povezava e in ali leži na levi ali na desni strani poligona od najvišjega oglišča. Opisali bomo primer, ko povezava e pripada poligonu $\mathcal{C}_1$ in leži na levi strani poligona. V ostalih treh primerih je obdelava dogodka podobna.

Če je točka p na prebirni premici med povezavo $levi\_rob\_\mathcal{C}_2$ in $desni\_rob\_\mathcal{C}_2$, povezava e leži na robu poligona $\mathcal{C}$ od točke p naprej. Potem preveri ali se povezavi e in $desni\_rob\_\mathcal{C}_2$ sekata. Če se sekata, oba dela povezav ležita na robu poligona $\mathcal{C}$, bodisi presečišče predstavlja najnižje oglišče poligona $\mathcal{C}$, če točka p leži levo od $desni\_rob\_\mathcal{C}_2$ na prebirni premici ali pa presečišče predstavlja najvišje oglišče poligona $\mathcal{C}$, če točka p leži desno od $desni\_rob\_\mathcal{C}_2$ na prebirni premici. Na koncu preveri še ali se povezavi e in $levi\_rob\_\mathcal{C}_2$ sekata. Če se sekata presečišče predstavlja oglišče poligona $\mathcal{C}$ in povezava, ki poteka iz presečišča je del povezave e, če točka p leži levo od $levi\_rob\_\mathcal{C}_2$ na prebirni premici, drugače je del povezave $levi\_rob\_\mathcal{C}_2$.

Ker funkcija $presekKonveksnihPoligonov(\mathcal{C}_1, \mathcal{C}_2)$ ne potrebuje vrste dogodkov lahko naslednji dogodek najde v konstantnem času s pomočjo štirih kazalcev na povezave, ki trenutno sekajo prebirno premico. Če algoritem išče presečišče konveksnih poligonov sestavljenih iz n polravnin, je število vseh dogodkov največ $2(n+1)$. Vsak dogodek obdela v konstantnem času, iz tega sledi, da je časovna zahtevnost algoritma $\mathcal{O}(n)$.



Slika 3.6: Prikazuje trikotnik iz točk $\Omega = \{p_{-2}, p_{-1}, p_0\}$.

3.2 Dualni graf Delaunayevege triangulacije

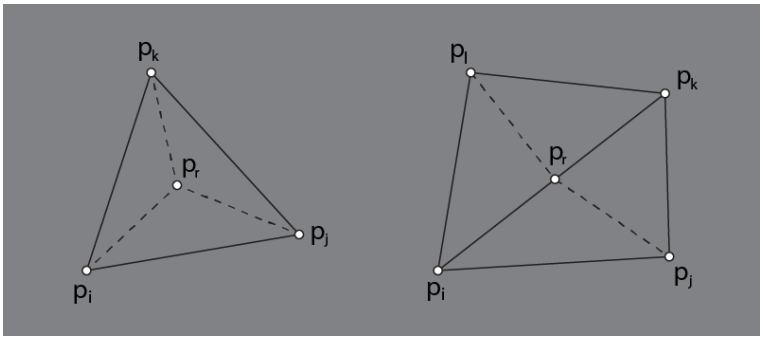
Opisali bomo konstrukcijo ravnine Delaunayeve triangulacije. Konstrukcija Voronoijevega diagrama sledi iz dualnosti.

Poznamo vrsto različnih algoritmov za konstrukcijo Delaunayeve triangulacije. Med njimi so algoritmi „deli in vladaj“, algoritmi s prebirno premico, med njimi ustrezno prilagojen Fortunov algoritem, in naključni inkrementalni algoritmi. V nadaljevanju bomo bolj podrobno opisali naključni inkrementalni algoritem, ki ima časovno zahtevnost $\mathcal{O}(n \log n)$.

3.2.1 Naključni inkrementalni algoritem

Na začetku algoritem izbere tri točke $\Omega = \{p_{-2}, p_{-1}, p_0\}$, ki tvorijo trikotnik, znotraj katerega ležijo vse točke množice $\mathcal{P} = \{p_1, p_2, ..., p_n\}$ (glej sliko 3.6) in gradi Delaunayevo triangulacijo nad unijo $\Omega \cup \mathcal{P}$. Na koncu konstrukcije iz Delaunayeve triangulacije odstrani vse trikotnike, ki imajo eno ali več vozlišč v točkah množice $\Omega = \{p_{-2}, p_{-1}, p_0\}$.

Inkrementalni algoritem dodaja točke iz množice $\mathcal{P}$ eno za drugo v Delaunayevo triangulacijo $\mathcal{DT}(\mathcal{P})$ v naključnem vrstnem redu in ohranja $\mathcal{DT}(\mathcal{P})$ s trenutno dodanimi točkami. Predpostavimo, da v obstoječo Delaunayevo triangulacijo dodamo točko p_r. Če točka p_r leži v notranjosti obstoječega trikotnika jo povežemo z vsemi oglišči. Če leži na povezavi med dvema triko-



Slika 3.7: Predstavlja dva možna primera kje lahko točka p_r leži.

tnikoma jo povežemo z obema krajiščema povezave in nasprotnim ogliščem v obeh trikotnikih. Oba možna scenarija sta prikazana na sliki 3.7. S takšnim dodajanjem točk algoritem ves čas ohranja triangulacijo, vendar ne nujno Delaunayevo, saj lahko dodana točka p_r tvori nov trikotnik, ki ne izpolnjuje pogojev izreka 5. Da dobimo Delaunayevo triangulacijo, pregledamo vse konveksne štirikotnike z novimi povezavami in, če je potrebno, zamenjamo diagonali. Predstavimo algoritem s psevdo kodo.

```
vhod: množica točk v ravnini P
izhod: Delaunayeva triangulacija DT(P)
delaunayevaTriangulacija(P):
1.    v DT(P) dodaj trikotnik iz točk p_{-2}, p_{-1}, p_0
2.    poišči naključno permutacijo točk iz množice P
3.    for (r = 1 to n)
4.        poišči trikotnik t_{ijk} v DT(P), znotraj katerega
          leži točka p_r
5.        if (p_r leži v notranjosti trikotnika t_{ijk})
6.            then dodaj povezave med p_r in tremi oglišči
                  trikotnika t_{ijk})
7.                 legalizirajPovezavo(p_r, p_i p_j, DT(P))
8.                 legalizirajPovezavo(p_r, p_j p_k, DT(P))
9.                 legalizirajPovezavo(p_r, p_k p_i, DT(P))
```

```
10.          else dodaj povezavo med p_r in štirimi vozlišči
                 sosednjih trikotnikov
11.              legalizirajPovezavo(p_r, p_ip_j, DT(P))
12.              legalizirajPovezavo(p_r, p_jp_k, DT(P))
13.              legalizirajPovezavo(p_r, p_kp_l, DT(P))
14.              legalizirajPovezavo(p_r, p_lp_i, DT(P))
15.   odstrani točke p_{-2}, p_{-1}, p_0 in vse povezave iz njih
```

Funkcija *legalizirajPovezavo(p_r, $\overline{p_ip_j}$, $\mathcal{DT}$)*, z zamenjevanjem diagonal konveksnih štirikotnikov, ki jih tvorijo novi trikotniki s sosednjimi, triangulacijo preoblikuje v Delaunayevo triangulacijo. Opišimo omenjeno funkcijo s psevdo kodo.

```
vhod: točka p_r, povezava p_ip_j, triangulacija T(P)
izhod: Delaunayeva triangulacija DT(P)
legalizirajPovezavo(p_r, p_ip_j, T(P)):
1.    if (p_ip_j ni povezava Delaunayeve triangulacije)
2.        then naj bo t_ijk sosednji trikotnik preko povezave p_ip_j
3.            zamenjaj diagonalo p_ip_j z diagonalo p_rp_k
4.            legalizirajPovezavo(p_r, p_ip_k, T(P))
5.            legalizirajPovezavo(p_r, p_kp_j, T(P))
```

Razložimo prvo vrstico psevdo kode, ki opisuje funkcijo *legalizirajPovezavo(p_r, $\overline{p_ip_j}$, $\mathcal{DT}$)*. Kadar si dva trikotnika t_{ijk} in t_{ijr}, ki skupaj tvorita konveksen štirikotnik, delita diagonalo $\overline{p_ip_j}$ in ne izpolnjujeta pogojev izreka 5 jo zamenjamo z diagonalo $\overline{p_kp_r}$.

Tako zgradimo Delaunayevo triangulacijo $\mathcal{DT}(\mathcal{P})$ z inkrementalnim algoritmom. Da dobimo Voronoijev diagram $Vor(\mathcal{P})$, za vsak trikotnik t_{ijk} v Delaunayevi triangulaciji povežemo središče očrtane krožnice s središči očrtanih krožnic sosednjih trikotnikov. Če trikotnik t_{ijk} nima enega (ali več) sosednjega trikotnika oziroma, če leži na robu triangulacije, povezava predstavlja poltrak v Voronoijevem diagramu, ki poteka iz središča očrtane krožnice tri-

kotnika t_{ijk} preko povezave na robu triangulacija tako, da razpolavlja povezavo. Opišimo postopek še s psevdo kodo.

```
vhod: Delaunayeva triangulacija DT(P), ki vsebuje m trikotnikov
izhod: Voronoijev diagram podan kot množica daljic
voronoijevDiagram(DT(P)):
1.    for (r = 1 to m)
2.        t_r = trikotnik v DT(P)
3.        for (s = 1 to 3)
              t_s = sosednji trikotnik trikotnika t_r
4.            if (t_s obstaja)
5.                then poveži središče očrtane krožnice trikotnika
                      t_r s središčem očrtane krožnice trikotnika
                      t_s
6.                else povezava je poltrak iz središča očrtane krožnice
                      trikotnika t_r in razpolavlja stranico na robu
                      triangulacije
```

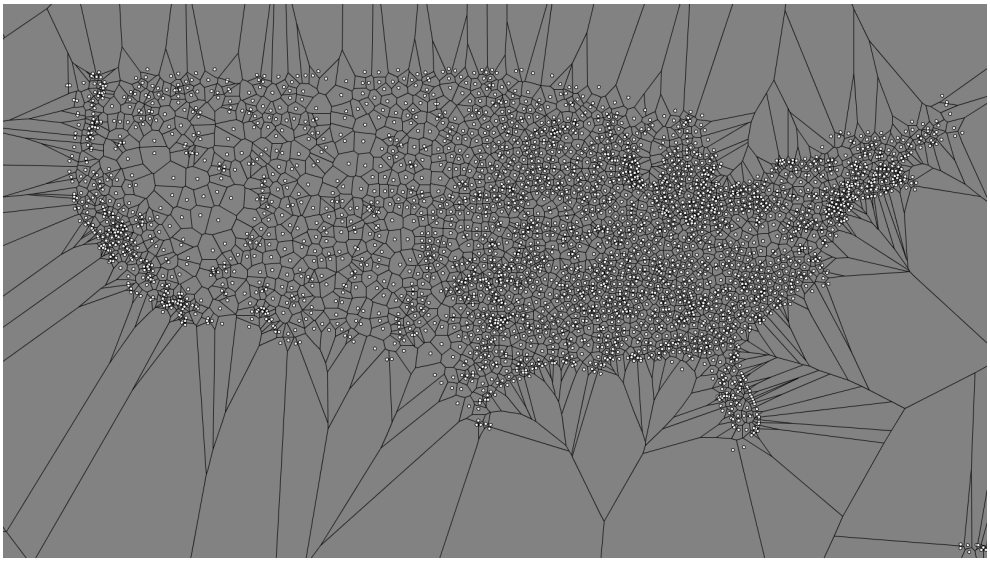
Poglavje 4

Implementacija konstrukcije Voronoijevega diagrama

V tem poglavju bomo opisali našo implementacijo algoritma za konstrukcijo Voronoijevega diagrama. Odločili smo se za Fortunov algoritem, ki smo ga programirali v razvijalskem okolju NetBeans [14] v programskem jeziku Java [13].

4.1 Fortunov algoritem

Za grafični prikaz na računalniškem zaslonu smo namesto standardne javanske knjižnice uporabili tako imenovani Processing [19], ki je sam zase ravno tako kot NetBeans razvijalsko okolje in hkrati tudi programski jezik, posebej namenjen vizualizaciji podatkov. Processing je napisan v programskem jeziku Java, zato obstaja Processing knjižnica, ki jo lahko uporabljamo direktno iz javanskega programa. Ker je v omenjeni knjižnici že implementirano vse potrebno za izrisovanje na zaslon, enega izmed razredov razširimo v „PApplet“ in dodamo metodo „public void draw()“, ki se izvede za vsako sličico, ki je prikazana na zaslonu. Znotraj te metode lahko že rišemo poljubne primitive kot so črte in krogi s preprostim klicem vnaprej definiranih metod, ki pripadajo knjižnici. Omenjena primitiva bosta zadoščala našim potrebam,



Slika 4.1: Voronoijev diagram zgrajen nad 3367 letališči v ZDA [15].

saj bomo točke risali kot kroge, Voronoijev diagram pa je skupek ravnih črt.

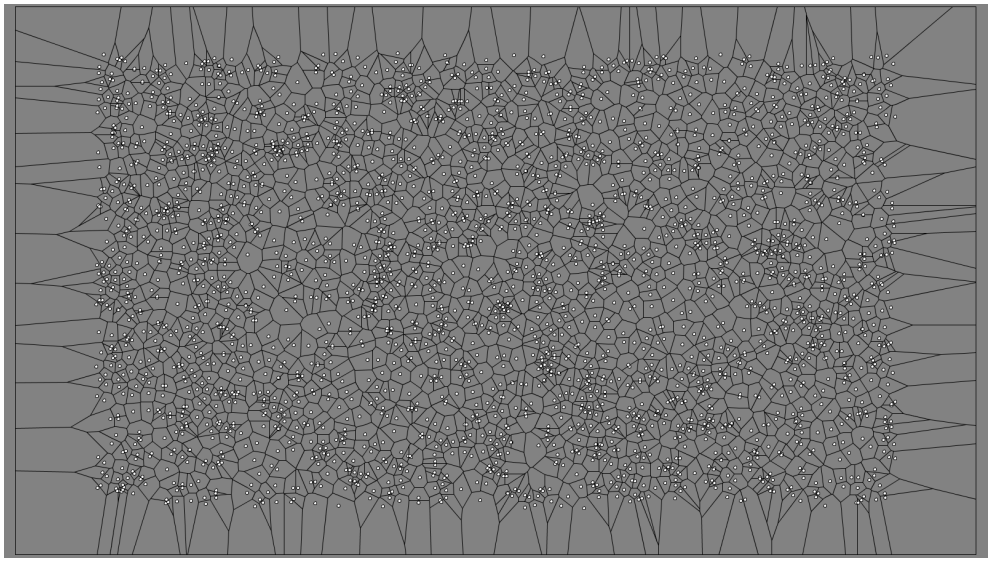
Vhodni podatki Fortunovega algoritma so točke. Zato smo napisali razred „**Input.java**“, ki točke prebere iz datoteke ali pa jih pred začetkom izvajanja algoritma z levimi kliki na računalniško miško označimo preko zaslona.

V razredu „**Voronoi.java**“ je glavna zanka algoritma in vsebuje javno dostopne statične kazalce na

- vrsto dogodokov točk $\mathcal{Q}_t$,
- vrsto dogodokov kroga $\mathcal{Q}_k$,
- mejno krivuljo $\mathcal{L}$ in
- seznam povezav Voronoijevega diagrama $\mathcal{S}$

in javno dostopne statične spremenljivke

- minimalna in maksimalna vrednost abscisne in ordinatne koordinate točk množice $\mathcal{P}$ označenih z x_{min}, x_{max} in y_{min}, y_{max}.



Slika 4.2: Voronoijev diagram zgrajen nad množico 2500 različnih naključnih točk.

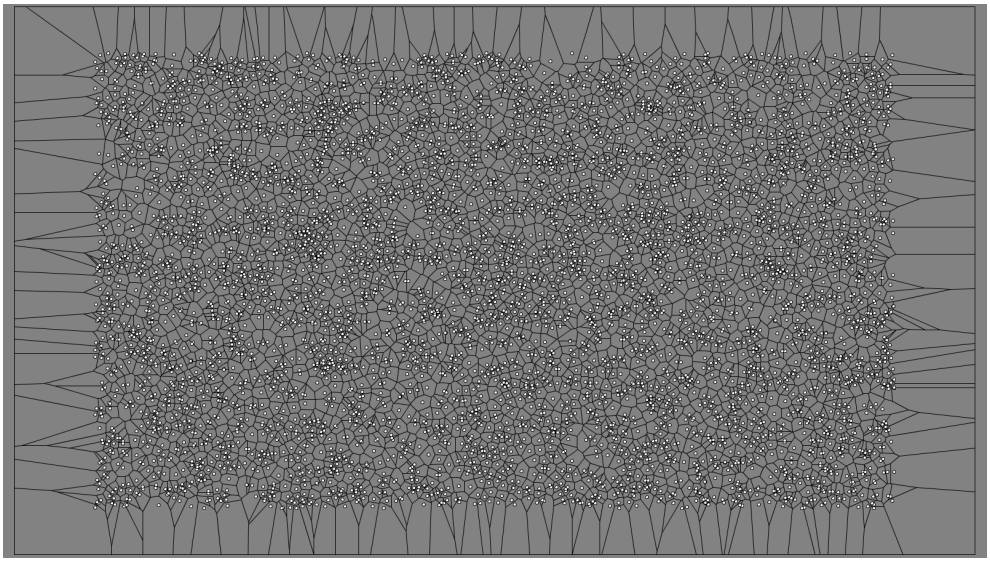
V razredu so tudi štiri javne statične metode namenjene konstrukciji Voronoijevega diagrama. Vrstni red klicev metod je pomemben in je tak kot v spodnjem opisu.

init() Inicializira prazno mejno krivuljo $\mathcal{L}$ in prazen seznam povezav v Voronoijevem diagramu $\mathcal{S}$.

fillPoints($\mathcal{P}$) Inicializira prazno vrsto dogodkov $\mathcal{Q}_t$ in $\mathcal{Q}_k$. Doda vse točke iz množice $\mathcal{P}$ v vrsto točk $\mathcal{Q}_t$ in nastavi vrednosti spremenljivkam x_{min}, x_{max} in y_{min}, y_{max}.

construct() Zabeleži si čas ob klicu metode, da na koncu izračuna čas konstrukcije diagrama in implementira glavno zanko Fortunovega algoritma.

draw() Metoda izriše vse povezave Voronoijevega diagrama iz seznama $\mathcal{S}$ in pravokotnik znotraj katerega ležijo vse povezave na zaslon računalnika.



Slika 4.3: Voronoijev diagram zgrajen nad množico 5000 različnih naključnih točk.

Parabola, ki predstavlja vozlišče seznama mejne krivulje $\mathcal{L}$, je implementirana v razredu „**Arc.java**“ in vsebuje kazalce na

- točko p, ki predstavlja parabolo,
- dve sosednji paraboli, ki sta označeni kot *prev* in *next*,
- dogodek kroga e tipa „Event“ in
- dve povezavi s_l in s_r tipa „Segment“, ki predstavljata Voronoijevi povezavi, ki ju presečišči s parabolama prev in next sčasoma izrišeta.

Parabola vsebuje tri bolj pomembne metode, ki jih bomo opisali.

intersection(p, l_y) Metoda prejme dva argumenta, točko p, ki predstavlja parabolo α, in višino prebirne premice l_y ter vrne točko presečišča parabole „this“ s parabolo α. Točko presečišča poišče tako, da izračuna razliko kvadratnih funkcij, ki predstavljata paraboli in s formulo $x = \frac{-b-\sqrt{b^2-4ac}}{2a}$ izračuna eno ničlo, ki jo nato vstavi v kvadratno funkcijo

parabole α. Da preprečimo deljenja z nič, ko je $a = 0$, metoda posebej obravnava primere, ko točki, ki predstavljata parabolo „this“ in p ležita na isti višini, ali ko katera izmed točk leži na višini prebirne premice l_y.

above(p) Metoda kot argument prejme točko p in vrne točko, ki leži na paraboli „this“ v mejni krivulji $\mathcal{L}$, če parabola leži navpično nad točko p, sicer vrne *null*.

chechAndAddEvent(l_y) Metoda prejme en argument, višino na kateri se nahaja prebirna premica l_y. Nato preveri ali tri točke $prev.p$, p in $next.p$ tvorijo trikotnik, ki je orientiran v smeri urinega kazalca. To pomeni, da parabola „this“ predstavlja potencialen dogodek kroga. V tem primeru doda nov dogodek kroga s kazalcem na parabolo „this“ v vrsto $\mathcal{Q}_k$. Trikotnik je orientiran v smeri urinega kazalca natanko takrat, kadar je vektorki produkt med dvema vektorjema, ki predstavljata stranici trikotnika, pozitiven. Če je vektorski produkt enak nič, pomeni, da sta povezavi v $Vor(\mathcal{P})$ vzporedni in v takšnem primeru nimamo potencialnega dogodka kroga.

Mejna krivulja je implementirana v razredu „ **FrontList.java**“ kot seznam parabol $\mathcal{L}$. V razredu je en kazalec na

- parabolo, ki predstavlja začetno vozlišče v seznamu.

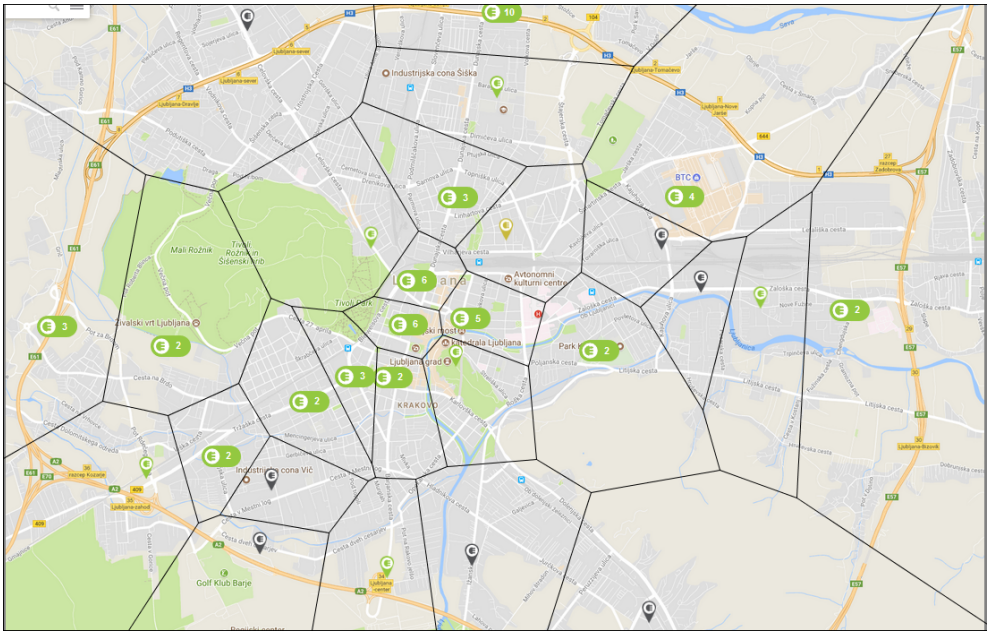
Implementira tudi dve metodi, ki obdelata posamezen dogodek.

handlePointEvent(p) Metoda obdela dogodek točke p.

handleCircleEvent(e) Metoda obdela dogodek kroga e.

Dogodek kroga je implementiran z razredom „**Event.java**“ in vsebuje dva kazalca na

- parabolo α, ki predstavlja izginjajočo parabolo v mejni krivulji $\mathcal{L}$ in
- točko q, ki predstavlja središče krožnice.



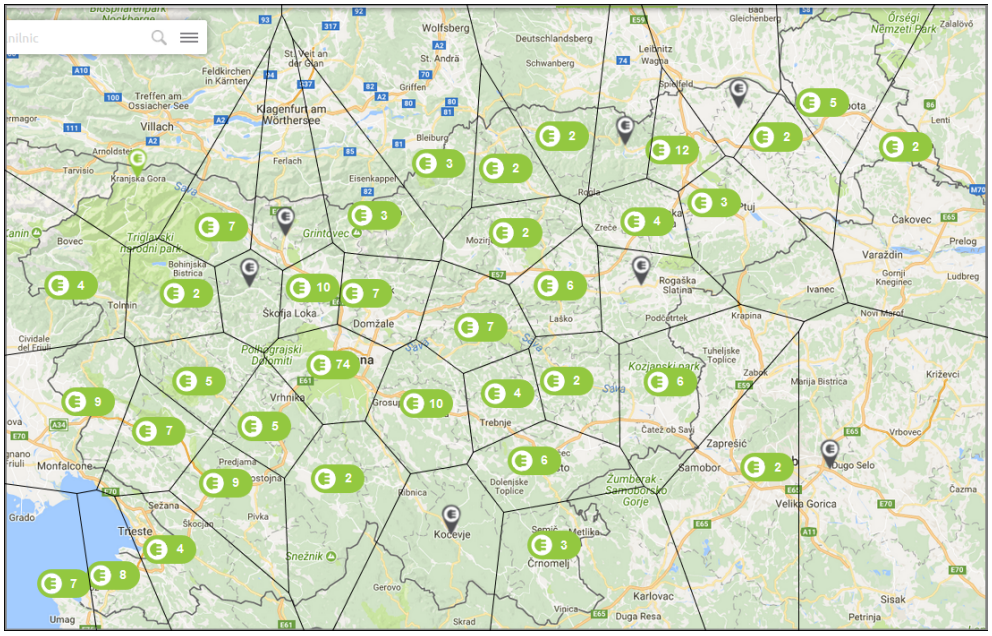
Slika 4.4: Voronoijev diagram zgrajen nad lokacijami javnih polnilnih postaj za električne avtomobile v Ljubljani [4].

in dve spremenljivki

- realno število y, ki predstavlja višino prebirne premice, kdaj se bo dogodek zgodil in
- spremenljivko *valid*, ki je na začetku „true“ in se po potrebi nastavi na „false“, da se dogodka v glavni zanki Fortunovega algoritma ne upošteva.

Prioritetni vrsti $\mathcal{Q}_t$ in $\mathcal{Q}_k$ sta implementirani kot naraščajoče urejeni kopici v seznamu. Takšna podatkovna struktura ima časovno zahtevnost dodajanja in brisanja elementa $\mathcal{O}(\log n)$. Najmanjši element v naraščajoči kopici pa lahko najdemo s časovno zahtevnostjo $\mathcal{O}(1)$.

V naši implementaciji mejne krivulje $\mathcal{L}$, parabolo poiščemo s časovno zahtevnostjo $\mathcal{O}(n)$. Običajna implementacija Fortunovega algoritma predposta-



Slika 4.5: Voronoijev diagram zgrajen nad lokacijami javnih polnilnih postaj za električne avtomobile v Sloveniji [4].

vlja, da je mejna krivulja implementirana kot uravnoteženo iskalno drevo. Časovna zahtevnost iskanja parabole v uravnoteženem iskalnem drevesu je $\mathcal{O}(\log n)$. Zato ima naša implementacija slabšo časovno zahtevnost $\mathcal{O}(n^2)$.

Svojo implementacijo Fortunovega algoritma smo testirali najprej na enakih podatkih kot na sliki 2.5, nad točkami, ki predstavljajo lokacije letališč v Združenih državah Amerike (glej sliko 4.1), na nekaj naborih naključnih točk v ravnini (glej sliko 4.2 in 4.3) in nad lokacijami javnih polnilnic za električne avtomobile v Sloveniji, ki so v današnjih časih vse bolj priljubljeni (glej sliko 4.4 in 4.5).

Implementirani algoritem pa ne deluje pravilno, če množica $\mathcal{P}$ vsebuje dve (ali več) enaki točki. Pred klicem algoritma je zato potrebno vnaprej odstraniti ponavljajoče točke.

4.2 Presečišče premic

Za zaključek poglavja opišimo še implementacijo iskanja presečišča dveh premic, ki jo uporabljamo pri iskanju presečišča konveksnih poligonov in, da Voronoijev diagram postavimo znotraj pravokotnika. Pravokotnik določimo z minimalnimi in maksimalnimi koordinatami točk, ki smo jih shranili v x_{min}, x_{max} in y_{min}, y_{max}. Nato poiščemo presečišča med pravokotnikom in Voronoijevimi povezavami, da porežemo vse povezave, ki so poltrakovi in se zaključijo na robu pravokotnika.

Premico v ravnini lahko predstavimo z vektorsko enačbo $\vec{F}(x)$, ki je določena s točko $\vec{r}$ in vektorjem $\vec{n}$ v smeri premice.

$$\vec{F}(x) = x\vec{n} + \vec{r} = x \begin{pmatrix} n_x \\ n_y \end{pmatrix} + \begin{pmatrix} r_x \\ r_y \end{pmatrix},$$

kjer parameter x določa lego točke na premici. Presečišče dveh premic dobimo tako, da enačimo vektorski enačbi, kjer t označuje parameter na prvi, s pa na drugi premici.

$$\vec{F}_1(t) = \vec{F}_2(s)$$

$$t\vec{n}_1 + \vec{r}_1 = s\vec{n}_2 + \vec{r}_2$$

$$t \begin{pmatrix} n_{1x} \\ n_{1y} \end{pmatrix} + \begin{pmatrix} r_{1x} \\ r_{1y} \end{pmatrix} = s \begin{pmatrix} n_{2x} \\ n_{2y} \end{pmatrix} + \begin{pmatrix} r_{2x} \\ r_{2y} \end{pmatrix}$$

$$t \begin{pmatrix} n_{1x} \\ n_{1y} \end{pmatrix} - s \begin{pmatrix} n_{2x} \\ n_{2y} \end{pmatrix} = \begin{pmatrix} r_{2x} \\ r_{2y} \end{pmatrix} - \begin{pmatrix} r_{1x} \\ r_{1y} \end{pmatrix}$$

$$\begin{pmatrix} n_{1x}, & -n_{2x} \\ n_{1y}, & -n_{2y} \end{pmatrix} \begin{pmatrix} t \\ s \end{pmatrix} = \begin{pmatrix} r_{2x} - r_{1x} \\ r_{2y} - r_{1y} \end{pmatrix}$$

Če premici nista vzporedni se sekata in presečišče dobimo tako, da rešimo dobljeni linearni sistem za neznanki t in s in ju vstavimo v pravi enačbi. Če pa sta premici vzporedni je determinanta matrike enaka nič. V tem primeru sta premici lahko mimobežnici, kar pomeni, da se ne sekata ali pa v celoti ležita ena na drugi in predstavljata isto premico. V obeh primerih, ko sta

premici vzporedni, v naši implementaciji ne naredimo ničesar in namesto točke presečišča vrnemo *null*.

Iskanje presečišča dveh premic smo uporabili tudi pri iskanju krožnice skozi tri nekolinearne točke p, q, r. Krožnico predstavimo s polmerom krožnice in s središčem. Središče krožnice dobimo kot presečišče simetral na daljici $\overline{pq}$ in $\overline{pr}$. Polmer krožnice pa izračunamo kot razdaljo med presečiščem simetral in ene izmed točk p, q, r.

Poglavje 5

Sklepne ugotovitve

V diplomski nalogi smo predstavili Voronoijev diagram in triangulacijo ravnine. Prikazali smo največje število oglišč in povezav v Voronoijevem diagramu, zgrajenim nad $n \geq 3$ točkami in s pomočjo Eulerjeve formule za planarne grafe ugotovili, da je število oglišč in povezav linearno v primerjavi s številom točk n. Predstavili smo tudi Delaunayevo triangulacijo, ki je dualna struktura Voronoijevega diagrama in opisali povezavo med njima ter ugotovili, da lahko preko Delaunayeve triangulacije v linearnem času zgradimo Voronoijev diagram. Implementirali smo Fortunov algoritem konstrukcije ravninskih Voronoijevih diagramov in podrobno opisali našo implementacijo.

Pri implementaciji Fortunovega algoritma smo najprej brez težav implementirali iskanje presečišča premic in iskanje očrtanega kroga trikotniku. Potem smo tudi brez zapletov implementirali kopico, ki poskrbi za pravilen vrstni red obdelovanja dogodkov. Največ preglavic nam je povzročala implementacija mejne krivulje in pravilna obdelava dogodkov. Na koncu smo imeli tudi kar nekaj težav s sestavljanjem algoritma, da je deloval kot celota. Zaradi lažje realizacije naša implementacija Fortunovega algoritma ni idealna, saj smo mejno krivuljo implementirali kot iskalni seznam in ne kot uravnoteženo iskalno drevo, zato ima slabšo časovno zahtevnost kot jo narekuje Fortunov algoritem.

Pri konstrukciji Voronoijevega diagrama nad točkami, ki predstavljajo lokacije javnih polnilnih postaj za električne avtomobile v Sloveniji (glej sliko 4.5), bi bilo smiselno, da bi točki, ki predstavlja 74 polnilnic v Ljubljani dodelili večje območje kot točki, ki predstavlja eno polnilnico v Kočevju. Takšnim Voronoijevim diagramom rečemo uteženi Voronoijevi diagrami [1]. Vsaki točki iz množice nad katero gradimo Voronoijev diagram, pripada tudi utež, ki predstavlja pomembnost točke.

Obstaja tudi ravninski Voronoijev diagram zgrajen nad daljicami. Za izgradnjo takšnega Voronoijevega diagrama bi morali nadgraditi obstoječ Fortunov algoritem in bi imeli opravka z več različnimi dogodki točk kot tudi več različnimi dogodki krogov [2].

Pri konstrukciji Voronoijevega diagrama bi lahko uporabili drugačno metriko razdalje, na primer manhattansko razdaljo. Prav tako pa poznamo tudi drugačne variante Voronoijevih diagramov [6, 7, 17], ki jih tukaj nismo omenili.

Literatura

[1] Franz Aurenhammer and Herbert Edelsbrunner. An optimal algorithm for constructing the weighted Voronoi diagram in the plane. *Pattern Recognition*, 17(2):251–257, 1984.

[2] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd edition, 2008.

[3] G. Brown. Point density in stems per acre. *New Zealand Forestry Service Research Notes*, 38:1–11, 1965.

[4] Elektro Ljubljana d.d. Gremo na elektriko. Dosegljivo: `https://www.gremonaelektriko.si/`. [Dostopano 8. 11. 2017].

[5] R. Descartes. *Principia philosophiae*. Ludovicum Elzevirium, 1644.

[6] ADAM DOBRIN. A review of properties and variations of voronoi diagrams. Dosegljivo: `https://www.whitman.edu/Documents/Academics/Mathematics/dobrinat.pdf`, 2005. [Dostopano 24. 11. 2017].

[7] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal Voronoi tessellations: Applications and Algorithms. *SIAM Rev.*, 41(4):637–676, 1999.

[8] S. Fortune. A sweepline algorithm for Voronoi diagrams. In *Proceedings of the Second Annual Symposium on Computational Geometry*, SCG, pages 313–322, New York, NY, USA, 1986.

[9] Oyvind Hjelle and Morten Dæhlen. *Triangulations and Applications (Mathematics and Visualization)*. Springer-Verlag New York, Inc., Secaucus NJ, USA, 2006.

[10] G. Lejeune Dirichlet. Über die Reduction der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen. *Journal für die reine und Angewandte Mathematik*, 40:209–227, 1850.

[11] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[12] R. Mead. A relationship between individual plant-spacing and yield. *Annals of Botany*, 30:301–309, 1966.

[13] Sun Microsystems. Java. Dosegljivo: `https://www.java.com/en/`. [Dostopano 25. 10. 2017].

[14] Sun Microsystems. Netbeans. Dosegljivo: `https://netbeans.org/`. [Dostopano 22. 10. 2017].

[15] Mike Bostock. Us airports positions. Dosegljivo: `https://gist.github.com/mbostock/4360892`. [Dostopano 8. 11. 2017].

[16] Mike Bostock. Us airports voronoi. Dosegljivo: `https://bl.ocks.org/mbostock/4360892`. [Dostopano 22. 10. 2017].

[17] Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Inc., New York, NY, USA, 1992.

[18] Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[19] Casey Reas and Benjamin Fry. Processing. Dosegljivo: `https://processing.org/`. [Dostopano 23. 10. 2017].

[20] Michael Ian Shamos and Dan Hoey. Closest-point problems. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, pages 151–162, Washington, DC, USA, 1975.

[21] Georges Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième mémoire. Recherches sur les parallélloèdres primitifs. *Journal für die Reine und Angewandte Mathematik*, 134:198–287, 1908.

[22] Robin J Wilson. *Introduction to Graph Theory*. John Wiley & Sons, Inc., New York, NY, USA, 1986.