ŠIAULIŲ UNIVERSITETAS
Fizikos – matematikos fakultetas
Informatikos katedra

Laura Žvikaitė
Informatikos specialybė

Voronojaus diagramos ir jų taikymai

Magistro darbas

Vadovė – doc. R.Steuding
Recenz. – doc.V.Sirius

Šiauliai
2005

Turinys:

I. Įvadas

Voronojaus diagramos yra labai naudingos struktūros. Pavyzdžiui, jos puikiai reprezentuoja augimo fenomeną. Todėl nenuostabu, kad Voronojaus diagramos vaidina svarbų vaidmenį modeliuojant kristalo ar didelių visatos objektų augimo procesą. Voronojaus diagramos taip pat padeda išspręsti tokias problemas, kaip artimiausio kaimyno paieška arba judėjimo planavimas, naudojamos miestų planavime, kartografijoje, planuojant mobiliųjų telefonų retransliacijos stočių išdėstymą ir pan. Voronojaus diagramų bei jų apibendrinimų, jų savybių, jų kūrimo algoritmų nagrinėjimas iki šiol išlieka pagrindinis skaičiuojamosios geometrijos objektas, o gauti rezultatai vis labiau randa praktinį pritaikymą.

Mano pasirinkta tema – Voronojaus diagramų taikymas tinklų optimizavime.

Tikslai: Išsiaiškinti Voronojaus diagramų vaidmenį įvairiose mokslinėse srityse, pateikti konkrečių modelių pavyzdžių ir realizuoti praktiškai konkretų modelį pasirinktoje srityje.

Pateikiami šie priedai:

1. Vartotojo vadovas;
2. Techninė užduotis;
3. CD.

II. Teorinė dalis

1) Temos analizė

Voronojaus diagrama – klasikinis skaičiuojamosios geometrijos uždavinys.

Pasaulyje ši teorija yra pakankamai gerai žinoma. Voronojaus diagrama ir Tinklinė Voronojaus diagrama – tai puikiai išnagrinėtos temos. Yra žinomi ne vienas garsus ir efektyvus algoritmas. Pagrindinis mokslininkų tikslas – rasti dar efektyvesnį algoritmą. Voronojaus diagramos ne tik skaičiuojamosios geometrijos uždavinys. Jos pritaikytos ir daugelyje kitų mokslo sričių. Dar vienas aspektas – pritaikymų galimybės. Realizuotas pritaikymas labai skirtingose sferos. Ieškoma vis įvairesnių sričių, kuriose galėtų pritaikyti Voronojaus diagramas.

Lietuvoje ši tema beveik nenagrinėjama. Nepavyko rasti jokių publikacijų, literatūros ar pan. Lietuvos mokslininkams ir programuotojams – tai naujos perspektyvos.

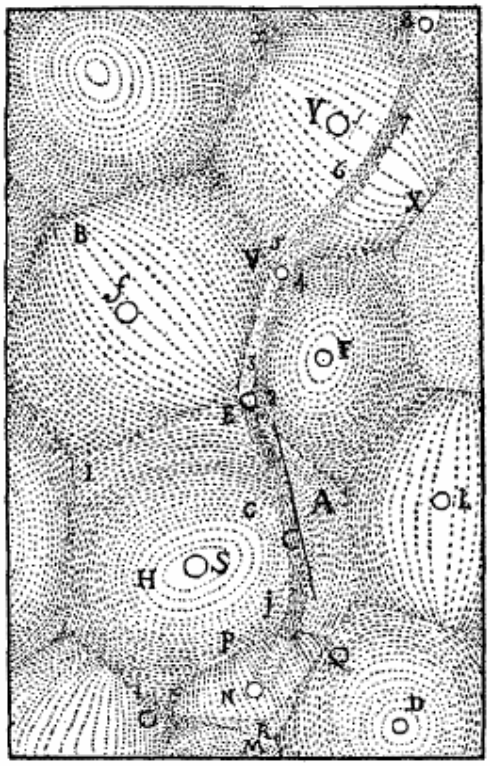
2) Darbo srities analizė

1. Voronojaus diagramos apibrėžimas ir paprasčiausios savybės

VD – Voronojaus (Voronoi) diagrama. Plokštumos taškų M_i Voronojaus diagrama vadiname plokštumos suskaidymą į sritis, pasižyminčias savybe, kad sričiai priklausantis taškas M_i yra artimiausias visiems tos srities taškams.

Voronojaus diagrama – klasikinis skaičiuojamosios geometrijos uždavinys. Voronojaus diagrama labai susijusi ir su kitomis skaičiuojamosios geometrijos sritimis, kurios pradėtos nagrinėti jau XVII amžiuje. R.Dekartas (Descartes) [1] pareiškė, kad saulės sistema sudaryta iš sūkurių. Jo iliustracijos rodo erdvės skaldymąsi į išgaubtas sritis. Kiekviena sritis sudaryta iš materijos besisukančios apie nejudančią žvaigždę (1 pav.).

1 pav. Dekarto erdvės skaidymas į viršūnes



Nors Dekartas dar nebuvo iki galo išnagrinėjęs šių sričių, jau buvo aišku, kad pagrindinė idėja bus plėtojama toliau. Tarkime, duota erdvė *M* ir *p* sričių aibė *S* toje erdvėje *M*. Tarkime, sritis *p* įtakoja erdvės *M* tašką *x*. Tada regioną *p* sudarys visi *x* taškai, kuriems *p* srities įtaka yra stipriausia, $p \epsilon S$. Atsiradus šiai koncepcijai, ji buvo sėkmingai pritaikyta įvairiose mokslo srityse. Atitinkamose srityse buvo naudojami skirtingi pavadinimai. Biologijoje ir psichologijoje buvo naudojamas terminas transformuotos vidurinės ašys, chemijoje ir fizikoje – Wigner-Seitz sritys, kristalografijoje – veiksmo sritys. Matematikai Dirichlė (Dirichlet) ir Voronojus buvo pirmieji, kurie formaliai pristatė šią koncepciją. Dirichlė apibrėžė šį uždavinį 1850 metais. 1908m. tiksliai matematiškai aprašė Voronojus. Šią idėją jie panaudojo tirdami kvadratines figūras, kur sritys yra taškų tinkleliai priklausantys nuo Euklido atstumo. Galutinė struktūra buvo pavadinta Voronojaus diagrama, šis pavadinimas naudojamas ir šiandien. Voronojus buvo pirmasis apibrėžęs šios struktūros dualumą - bet kurios dvi taškų aibės yra sujungtos, jų sritys turi bendrą briauną. Vėliau Delaunė (Delaunay) nustatė, kad dvi taškų aibės yra sujungtos tada jei (tai yra, tada ir tik tada jei) jos yra apskritime, kurio viduje nėra aibės *S* taškų. Tada Voronojaus diagramos dualumas žymimas Delaunė trianguliacija.

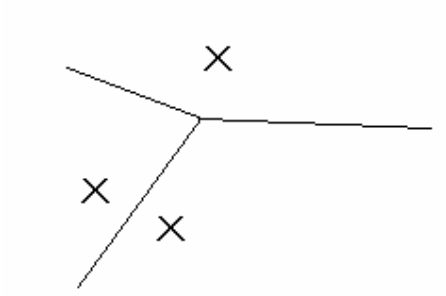
Voronojaus diagramą ir jos dualumą galima pritaikyti ne tik kitose mokslo srityse, bet ir daugelyje geometrinių uždavinių.

Taikymo sritys:

- Kartografijoje, miestų planavime;
- Planuojant mobiliųjų telefonų retransliacijos stočių išdėstymą;
- Efektyviai artimiausio kaimyno paieškai (atpažinimas, kompiuterinė rega);
- Aproksimuojant paviršius (kompiuterinė geometrija, atpažinimas);
- Formuojant gardeles, baigtinių elementų metode (diferencialinės lygtys);
- Vizualizacijoje (kompiuterinė rega, geometrija);
- Daugiamačių duomenų struktūrose (kompiuterija).

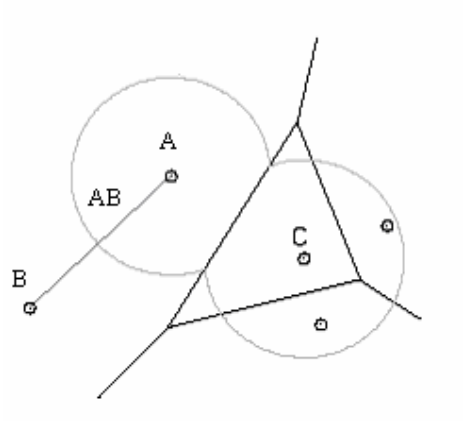
Voronojaus diagramos dažniausiai aprašomos kalbant apie taip vadinamąjį Pašto uždavinį. Plokštumoje yra keli taškai: pašto skyriai. Kiekvienas pašto skyrius priklauso sričiai, kurios visi taškai yra artimesni būtent tam pašto skyriui nei bet kuriam kitam. Pašto skyrius yra Voronojaus generatorius, o nustatytų sričių ribos vaizduoja Voronojaus diagramą (2 pav.).

2 pav. Trys pašto skyriai ir jų sritys pagal Voronojaus diagramas



Voronojaus diagramos forma priklauso nuo apibrėžtos sąlygos. Dažniausiai naudojami Voronojaus diagramos konstravime matai: Euklido atstumas (L) arba Manheteno atstumas. Paprasčiausia Voronojaus diagramos forma pavaizduota 3 paveikslėlyje.

3 pav. Euklido atstumas *AB* dviems Voronojaus generatoriams *A* ir *C*



Šiuo atveju *L* pritaikyta plokštumoje. Atstumas tarp dviejų taškų *A* ir *B* apskaičiuojamas pagal formulę 2.1

$$L(A,B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} \qquad (2.1)$$

Dėl *L* pasirinkimo sudaryta erdvė tampa izotropinė: kiekviena kryptimi įmanoma keliauti tuo pačiu greičiu. Tuo pačiu atstumu nuo generatoriaus nutolę taškai suformuoja apskritimą. Apskritimo dydis priklauso nuo keliavimo greičio (*a*) ir kelionės trukmės (*b*). Dėl šios priežasties apskritimas vadinamas izochrone: izochronės sujungia taškus, kurie nutolę nuo vieno ar kelių generuojančių taškų vienodu atstumu.

Toks Voronojaus diagramos konstravimo metodas patogus programinėms realizacijoms ir pritaikytas GIS sistemoje, nors negalima teigti, kad tai visiškai atspindi realius keliavimo kaštus, kurie nagrinėjami transporto uždaviniuose.

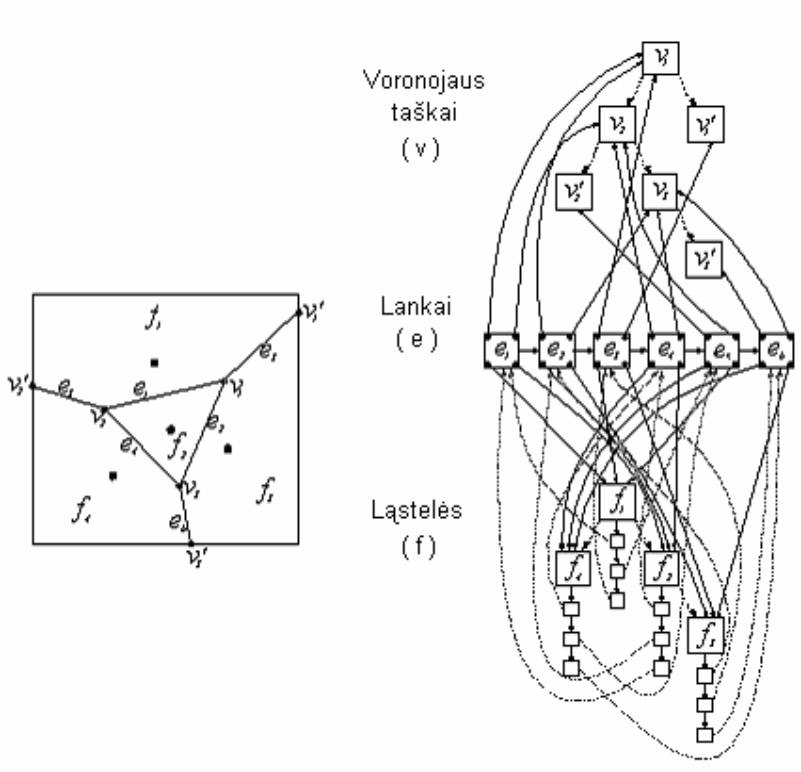
Plokštumos suskaidymas pagal Voronojaus diagramą nevienarūšiuose transporto tinkluose yra netgi mažiau realistinis. Tokie tinklai dažniausiai nėra suplanuoti ir skirtingos keliavimo rūšys leidžia įvairiu greičiu keliauti pagal tinklo lankus. Be to, patekimas į tinklą yra ribotas: gatvių tinklas gali prasidėti tik prie įėjimo į namą, o viešasis transporto tinklas gali prasidėti tik stotyse ar stotelėse. Dėl šios priežasties nagrinėsime Tinklinę Voronojaus diagramą.

2. Vorornojaus diagramų konstravimas

2.1 Duomenų struktūra, reprezentuojanti Voronojaus diagramas

Duomenų struktūra, reprezentuojanti VD sudaryta iš trijų duomenų struktūrų. Voronojaus diagramą sudaro Voronojaus ląstelės, Voronojaus lankai ir Voronojaus mazgai [2]. Kiekviena grupė yra apibrėžta atitinkamoje duomenų struktūroje, kurios, be abejo, yra susijusios. 4 paveikslėlis demonstruoja grafinį Voronojaus diagramos vaizdą.

4 pav. Voronojaus diagramos duomenų struktūra



Voronojaus diagramą suformuota iš lankų (sričių susikirtimo briaunų) ir mazgų (lankų viršūnių). Voronojaus diagramos pavaizdavimui kompiuteryje tinka bet kokia duomenų struktūra, kurioje apibrėžta grafų teorija. Nepaisant to, Voronojaus diagrama vaizduojama keliais būdais. Populiariausias - taip vadinamasis keturių lankų duomenų struktūra. Remiantis paprasčiausia grafų teorija, galima nustatyti, kad daugiausia gali būti *3n – 6* lankų ir *2n – 4* viršūnių, kai turime *n* sričių. Kitaip tariant, reikalaujama tik O(n) atminties. Tai dar viena priežastis padidinanti praktinį Voronojaus diagramos pritaikymą.

Panagrinėkime standartinį Pašto uždavinį. Tarkime, turime šešis pašto skyrius. Atsižvelgiant į kliento buvimo vietą bus nustatytas Voronojaus regionas, kuriame yra klientui artimiausias pašto skyrius. Duomenų struktūroje "Voronojaus diagrama" nustačius taško buvimo vietą, galima labai greitai nustatyti artimiausio kaimyno sritį (per O(log n) laiką). Sritys gali reprezentuoti ir universalines parduotuves. Voronojaus apylinkę apibrėš diagramos lankai. Neskaitant, kad ši koncepcija plačiai taikome ekonomikoje, egzistuoja plačios pritaikymo galimybės ir biologijos, fizikos, chemijos sistemose. Voronojaus viršūnės yra tos vietos, kurias mažiausiai įtakoja srities pasiekiamumas, pagrindinis dėmesys – lengvas tam tikros vietos pasiekiamumas.

Kalbant apie Voronojaus diagramą būtina paminėti du labai svarbius dalykus. Pirmiausia, sričių trumpiausio sujungimo tinklas (pavyzdys galėtų būti elektros energijos tiekimo tinklas). Šis tinklas sujungia sritis, kurios yra Voronojaus kaimynės. Antra, kaimyninių sričių sujungimas į trianguliarų tinklą, kitaip vadinamoji Delaunė trianguliacija. Delaunė trianguliacija vadinamas toks plokštumos taškų M_i trejetų sujungimas trikampiais, kad nubrėžtame per bet kurio trikampio viršūnes apskritimo viduje nėra nei vieno taško M_i.

Voronojaus diagramų konstravimo metodai yra tokie seni, kaip seniai pradėti studijuoti gamtos mokslai. Be abejo, pirmosios diagramos buvo braižomos su pieštuku ir liniuote. Jau tais ankstyvaisiais laikais žmonės skundėsi nevienareikšmiškumais. Šiais laikais, kai egzistuoja efektyvus ir praktiškas algoritmų konstravimo būdas, svarbiausia svarstymų dalimi tampa specialieji (netipiniai) atvejai. Algoritmų esmė yra tipiška metodika, o problema tampa algoritmų veiklos sritis, kurioje modeliuojama Voronojaus diagrama.

Yra žinomi šie algoritmai:

- Keitimo algoritmas;
- Įterpimo algoritmas (Incremental insertion);
- "Skaidyk ir valdyk" algoritmas (Divide and Conqure);
- Fortune algoritmas.

Plačiau aptarsime kelis algoritmus.

2.2 “Įterpimo algoritmas”

Grynas (Green) ir Sibsonas (Sibson) pirmieji konstravo Voronojaus diagramą “Įterpimo” algoritmu. Pagrindinė idėja – išgauti *V(S)* iš *V(S\{p})* įterpiant sritį *p*. Kadangi *p* regionas gali būti sudarytas iš *n – 1* lankų, kai *n* = |*S*|, tai algoritmas yra $O(n^2)$ sudėtingumo. Tinkamai padalijus sričių aibes, galima tikėtis O(n).

Geriausiai „Įterpimo“ [3] algoritmą galima apibrėžti Delaunė trianguliacijai. „Įterpimo“ algoritmas trianguliaciją pradeda nuo trijų apjungtų trikampiu taškų pasirenkant atsitiktinai sekantį tašką, kuris prijungiamas prie trianguliacijos. Ši procedūra kartojama kol išsemiami visi taškai. Trianguliacijos papildymas atliekamas panaudojant ketvirtainį, aštuntainį arba Delaunė medį.

2.3 “Skaidyk ir valdyk” algoritmas

Šamos (Shamos) ir Hoji (Hoey) [6] pasiūlė pirmąją algoritmo “Skaidyk ir valdyk” versiją. “Skaidyk ir valdyk” yra klasikinė uždavinio sprendimo technika ir įrodyta, kad ši technika tinka ir geometriniams uždaviniams spręsti. Šis algoritmas pasižymi tuo, kad dalina pradinį uždavinį į kelis dalinius uždavinius, kuriuos išsprendus atskirai ieškomas bendro uždavinio sprendimas sujungiant dalinių uždavinių sprendimus.

“Skaidyk ir valdyk” algoritmo pagrindinė forma atrodo taip: taškų sričių aibė *S* dalinama į maždaug vienodo dydžio poaibius *L* ir *R*. Tada rekursiškai skaičiuojamos Voronojaus diagramos *V(L)* ir *V(R)*. Svarbiausia dalis: surasti padalijimo liniją, po to *V(L)* ir *V(R)* sujungti į *V(S)*. Ši dalis reikalauja O(n) operacijų, o visam algoritmui atlikti reikia O(n log n).

Algoritmas “SKAIDYK IR VALDYK”

Duota: *n* dydžio uždavinys *P*

Gauti: *P* uždavinio sprendimą

BEGIN

0. IF $n = 1$ THEN

   Sprendžiamas uždavinys *P*;

1. *P* uždavinys dalinamas į *k* dalinių uždavinių;
2. rekursiškai sprendžiamas kiekvienas dalinis uždavinys;
3. sujungiami dalinių uždavinių sprendimai uždavinio *P* išsprendimui;

END.

Uždavinio *P* "dydis" yra įvesties duomenų kiekio daliklis. Pavyzdžiui, jei reikia taškų aibėje *S*, kuri yra plokštumoje, sukonstruoti iškiliuosius kevalus, tada uždavinio dydis galėtų būti taškų skaičius duotoje aibėje *S*.

Efektyviam algoritmui sudaryti, tikimasi, kad 1. ir 3. žingsniai atliekami be didesnių kliūčių.

2.4 Fortune algoritmas

1985 S.Fortune [4] išrado įdomų algoritmą Voronojaus diagramai konstruoti. Jis panaudojo taip vadinamąją "šluojančių tiesių" strategiją, kurią įgyvendino pasitelkdamas kompiuterinės grafikos ir skaičiuojamosios geometrijos algoritmų. Algoritmo idėja yra labai paprasta ir blogiausiu atveju algoritmui atlikti reikia tik $O(n \log_2 n)$ laiko, čia *n* – įvestų taškų skaičius.

Konstruoti Voronojaus diagramas šiandien Fortune algoritmas yra pats populiariausias. Tačiau programuotojai vis dar konstruoja labiausiai tinkamas duomenų struktūras ir jomis operuojančias funkcijas. Apskritai reikia trijų duomenų struktūrų: vienos, kuri palaikytų Voronojaus diagramą ir kitų dviejų, kurios įgyvendintų valymo ("šlavimo") procesus (taško ir apskritimo įvykiai ir priekinė parabolės sritis) [5].

Fortune algoritmui atlikti reikia $O(n \log n)$ operacijų.

3) Darbinės srities modelis

1. Tinklinės VD (TVD) apibrėžimas ir pagrindinės savybės

Tarkime, savo vaikui jūs ieškote mokyklos. Kriterijus, pagal kurį rinksitės mokyklą, yra kelio ilgis iki mokyklos. Jei suformuluosite tai kaip erdvinę problemą, iš visų miesto mokyklų ieškosite tos, kuri būtų arčiausiai jūsų namų. Voronojaus diagramos būtų klasikinis metodas išspręsti šį erdvinį uždavinį. Voronojaus diagrama atskirtų sritis, kurios yra arčiausios tam tikrai mokyklai. Tai pavaizduota 5 paveikslėlyje. Jūsų namai bus vienoje iš šių nustatytų sričių.

5 pav. Trys mokyklos ir joms priklausančios sritys pagal Voronojaus diagramą.

Kartais populiarus metodas turi ribotas reikšmes, ypač jei galimybės nukeliauti į vietą yra apribotos vienu ar keliais tinklais. Šiuo atveju anksčiau minėtas metodas teikia tik apytikrį vertinimą, kuris gali būti klaidingas. Kai kurios Voronojaus diagramos prielaidos netaikytinos mieste:

- Atstumas tarp dviejų adresų nėra Euklido atstumas; jie turi būti skaičiuojami pagal transporto tinklą ar kelis tinklus. Jeigu jūsų duktė turi apeiti pastatų kvartalą, kelias iki mokyklos gali būti ganėtinai ilgesnis nei Euklido atstumas.
- Atstumai gali būti asimetriški. Įsivaizduokite, kad jūsų duktė eina nuokalne viena kryptimi, bet grįždama namo turi kilti į kalną. Jei ji važiuoja autobusu, tai autobuso maršrutas gali labai priklausyti nuo vienkrypčių gatvių. Taigi kelionės laikas gali labai skirtis.
- Atstumai gali būti nevienarūšiai. Iš esmės, jūsų duktė gali keliauti pėsčiomis. Bet jeigu kažkurią kelio dalį ji važiuos metro, ji nukeliaus ilgas distancijas per sąlyginai trumpą laiką.

Tinklinė Voronojaus diagrama nagrinėja atstumus tik tinkluose, ne plokštumoje. Tai dalina tinklą, ne erdvę, į Voronojaus ląsteles. Tinkle Voronojaus ląstelė yra lankų ir susikirtimo taškų aibė artimesnė vienam Voronojaus generatoriui (šiuo atveju: mokykla) negu kitam (6 pav.).

6 pav. Tinklinė Voronojaus diagrama tiems patiems trim generatoriams, kaip ir 5 pav., apskaičiavus kelionės kaštus keliaujant gatvių tinklu

Naudodami Tinklinę Voronojaus diagramą, galime modeliuoti tinklo pakeitimus. Naujos vietos nustatymas modeliuojamas įvedant į Tinklinę Voronojaus diagramą naują generatorių (tarkime, jei įstaiga ar pan. bus uždaryta, tai reikės ištrinti generatorių). Generatoriaus pozicija gali

keistis, tokiu atveju pasikeis ir Tinklinės Voronojaus ląstelės struktūra. Susidariusi situacija suteiks naujas srities parinkimo galimybes. Kaip pasikeitė nustatytos sritys, jei tinkle lankai ar mazgai tapo užblokuoti, arba kelionės kaštai keliaujant jais labai pasikeitė?

Tinklinė Voronojaus diagrama gali dinamiškai reaguoti į pasikeitimus tinkle.

Tinklinės Voronojaus diagramos algoritmas remiasi Deikstros trumpiausio kelio algoritmu [7]. Deikstros algoritmas jungiamame tinkle suranda trumpiausią kelią nuo pasirinkto mazgo iki bet kurio kito to tinklo mazgo. Jis turėtų būti modifikuotas taip, kad ieškotų trumpiausio kelio lygiagrečiai nuo keleto generatorių. Kiekvienas tinklo mazgas susiejamas su artimiausiu generatoriumi.

Mazgų Tinklinės Voronojaus diagramos apibrėžimas

Mes apibrėžėme Mazgų Tinklinę Voronojaus diagramą tiems generatoriams, kurie yra tinklo mazgų poaibiai. Mūsų užduotis yra priskirti tiems generatoriams kitus tinklo mazgus. Dėl to privalome modifikuoti Deikstros algoritmą.

Modifikuota versija suskaičiuoja kelionės kaštus nuo kiekvieno Voronojaus generatoriaus iki jo kaimyninių mazgų. Šitie mazgai nustatomi pagal laikinai mažiausius kelionės kaštus. Kitose iteracijose parenkamas mazgas su pigiausiomis sąnaudomis; jo kelionės kaštai jau tampa galutiniais (nebėra trumpesnių kelių). Kita iteracija prasideda nuo šio mazgo. Lygiagretūs skaičiavimai nuo visų Voronojaus generatorių turi išspręsti atvejus, kai mazgų kaimynai jau buvo pasiekti ir laikinai apibrėžti kitu trumpiausiu keliu. Šiais atvejais lyginami dveji kelionės iki kaimyno kaštai, pasirenkamas tas mazgas, iki kurio kelionės kaštai mažesni. Algoritmas pasibaigia, kai galutinai apibrėžiami visi mazgai.

Kiekvienas generatorius apibrėžia trumpiausio kelio medį. Apibrėžiant mazgą, algoritmas registruoja, iš kurio laikino kaimyninio mazgo jis buvo pasiektas, kitaip sakant, kuriam trumpiausio kelio medžiui jis priklauso. Taigi, mes žinome pilną mazgų aibės padalijimą.

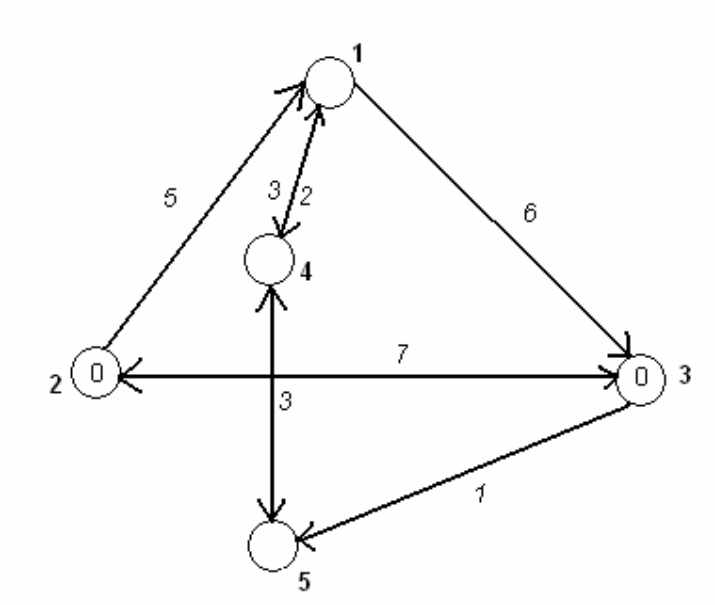
Lankų Tinklinės Voronojaus diagramos apibrėžimas

Kiekvienas tinklo lankas apibrėžiamas jo pradžios ir pabaigos mazgu (atsižvelgiant į tinklo generatorius). Galimi keturi skirtingi atvejai:

1. pradžios ir pabaigos mazgai priklauso tam pačiam Mazgo Tinklinės Voronojaus diagramos generatoriui. Šiuo atveju, lankas priskiriamas būtent tam trumpiausio kelio medžiui, ir žymimas generatoriaus ID.
2. pradžios ir pabaigos mazgai priklauso skirtingiems generatoriams ir lankas yra vienkryptis. Šiuo atveju, negalima keliauti nuo bet kurio lanko taško į pradžios mazgą, arba atvirkščiai, bet kuris lanko taškas gali būti pasiektas tiktai nuo pradžios mazgo. Visas lankas į trumpiausio kelio medį turi būti įtraukiamas nuo pradžios mazgo.
3. pradžios ir pabaigos mazgai priklauso skirtingiems generatoriams, lankas yra dvikryptis, bet simetrinis, lankas gali būti pradedamas tik pradžios arba pabaigos mazge. Tokie atvejai pasitaiko, pavyzdžiui, viešojo transporto tinkle, kur lanko pradžios ir pabaigos mazgai yra stotelės (sustojimo vietos). Šiuo atveju, visas lankas į trumpiausio kelio medį įtraukiamas nuo pradžios arba pabaigos mazgo, atsižvelgiant į tai, kuriuo atveju keliavimo kaštai mažesni.
4. pradžios ir pabaigos mazgai priklauso skirtingiems generatoriams, lankas yra dvikryptis ir nebūtinai simetrinis, lankas gali būti pradedamas bet kuriame taške. Šiuo atveju, lankas dalijamas. Padalijimo taškas yra tas taškas, kuriame kelionės kaštai į pradžios mazgo generatorių yra lygūs kelionės kaštams iki pabaigos mazgo generatoriaus. Pirmoji dalis x_A (nuo pradžios mazgo iki dalinimo taško) į trumpiausio kelio medį įtraukiama nuo pradžios mazgo, o antroji dalis, x_E, nuo dalinimo taško iki pabaigos mazgo, į trumpiausio kelio medį bus įtraukta nuo pabaigos mazgo. Padalijimo taške įvestas mazgas nepriklausys jokiam trumpiausio kelio medžiui. Taip susidaro riba tarp dviejų Tinklinių Voronojaus ląstelių.

2. TVD konstravimo algoritmas

Apžvelkime pavyzdį. Duotas tinklas su 5 mazgais ir 9 lankais. Mums reikia padalinti mazgus į dvi aibes su generatoriais 2 ir 3.



Tinklas taip pat gali būti apibrėžtas *gretimumo* matrica:

$$\text{Gretimumo matrica} = \begin{bmatrix} \infty & \infty & 6 & 3 & \infty \\ 5 & \infty & 7 & \infty & \infty \\ \infty & 7 & \infty & \infty & 1 \\ 2 & \infty & \infty & \infty & 3 \\ \infty & \infty & \infty & 3 & \infty \end{bmatrix}$$

Algoritme naudojami trys vektoriai:

a) vektorius *Kaštai*, sudarytas iš šiuo metu žinomų mažiausių kelionės kaštų nuo artimiausio generatoriaus, jo reikšmė 0 visiems generatoriams ir begalybė visiems kitiems mazgams.
b) vektorius *P*, binarinis Boolean tipo vektorius, kurio reikšmė lygi 1, jei priskyrimas yra laikinas.
c) Vektorius *AG*, jį sudaro kiekvienam taškui artimiausias generatorius, jo reikšmė mazgo ID generatoriui ir begalybė visiems kitiems mazgams. Mūsų pavyzdyje šie trys vektoriai yra:

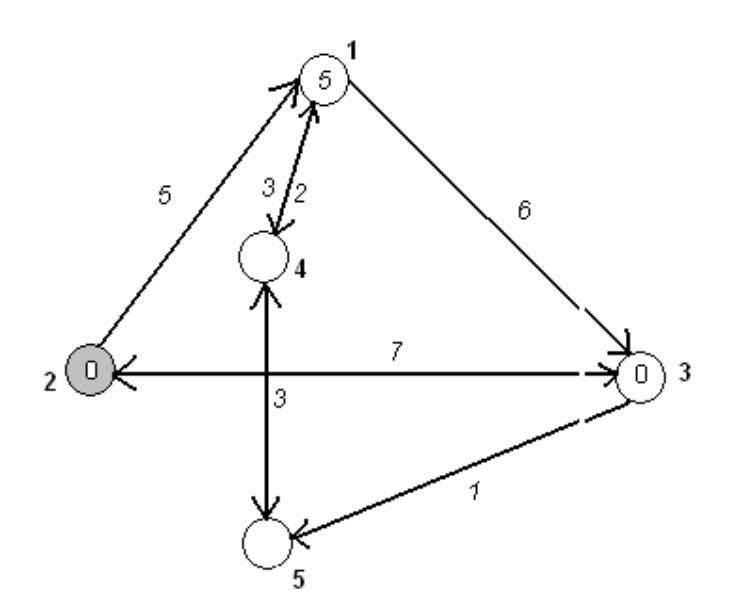
Kaštai = [∞ 0 0 ∞ ∞]

P = [1 1 1 1 1]

AG = [∞ 2 3 ∞ ∞]

Algoritmas pradedamas nuo šitų trijų vektorių kartoja aprašytas iteracijas tol, kol vektorius *Kaštai* įgyja baigtines reikšmes. Mūsų pavyzdyje apibrėžtam tinklui iš viso yra penkios iteracijos. Kiekvienoje iteracijoje *minindex* mazgas apibrėžia vektoriuje *Kaštai* minimalią reikšmę, kuri nėra galutinė, tai yra, jos reikšmė vektoriuje *P* vis dar yra 1. Šiam mazgui visi negalutiniai kaimynai yra apibrėžti *gretimumo* matrica ir vektoriumi *P*.

1.Iteracija: *minindex = 1*



Kaštai = [5 0 0 ∞ ∞]

P = [0 1 1 1 1]

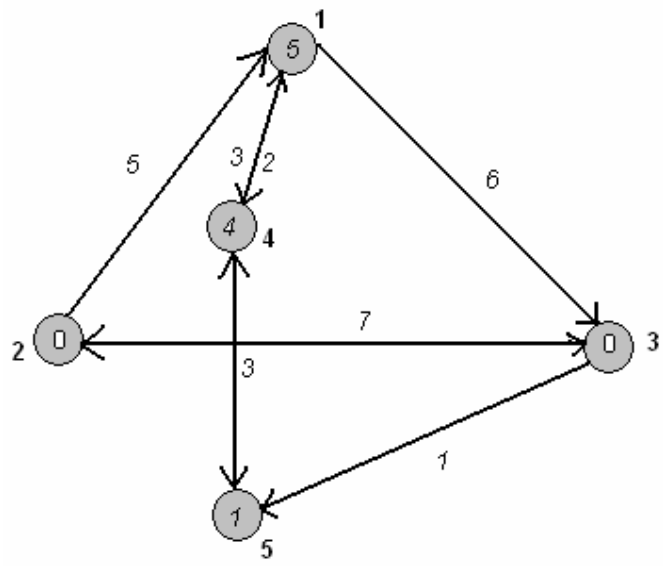
AG = [2 2 3 ∞ ∞]

Po penkių iteracijų gauname tokį rezultatą:

Kaštai = [5 0 0 4 1]

P = [0 0 0 0 0]

AG = [2 2 3 3 3]



Jau žinoma ir vektoriumi *AG* apibrėžta Mazgų Tinklinė Voronojaus diagrama, sudaryta iš penkių mazgų. Vektorius *Kaštai* sudarytas iš mažiausių kelionės kaštų nuo generatorių iki tinklo mazgų. Kita lentelė, *Koordinatės*, gali būti sudaryta iš mazgų koordinačių.

Kaštai = [5 0 0 4 1]

P = [0 0 0 0 0]

AG = [2 2 3 3 3]

Koordinatės = $\begin{bmatrix} 8 & 6 \\ 4 & 2 \\ 4 & 9 \\ 6 & 4 \\ 0 & 4 \end{bmatrix}$

Tinklą sudaro iš 9 lankai. Pradžios mazgai sudaro vektorių *A*, o pabaigos mazgai sudaro vektorių *B*. *A* ir *B* yra sudaromi tokiu būdu. *Gretimumo* matrica yra “skanuojama” eilutė po eilutės, kol randamos baigtinės reikšmės. Pavyzdžiui, pirmoji baigtinė reikšmė yra pozicijoje [1, 2], taigi, *A[1]* = 1 ir *B[1]* = 2: pirmoji rasta briauna prasideda mazge 1 ir baigiasi mazge 2. turėdami vektorius *A* ir *B*, galime nustatyti vektorius *AG[A]* ir *AG[B]*. Jie sudaromi atitinkamai iš pradžios ir pabaigos mazgus atitinkančių generatorių.

$n = 9$ $AG[A] = [\ 2\ 2\ 2\ 2\ 3\ 3\ 3\ 3\ 3\ ]$

$A = [1\ 1\ 2\ 2\ 3\ 3\ 4\ 4\ 5\ ]$ $AG[B] = [\ 3\ 3\ 2\ 3\ 2\ 3\ 2\ 3\ 3\ ]$

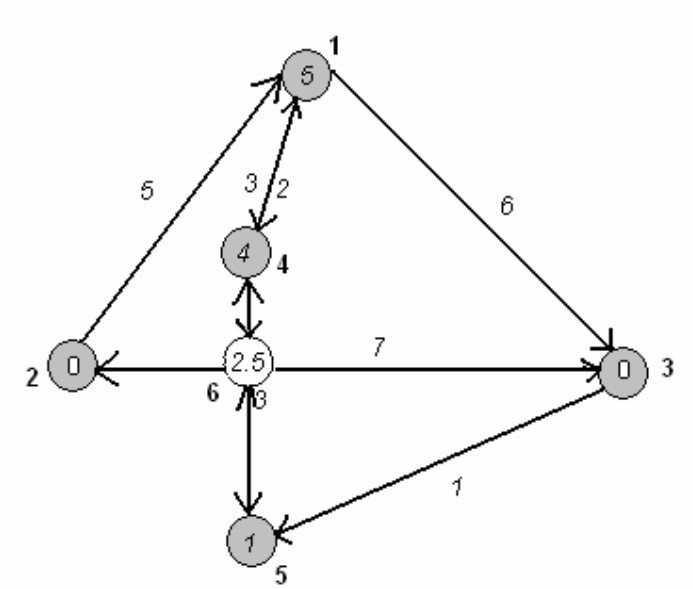
$B = [3\ 4\ 1\ 3\ 2\ 5\ 1\ 5\ 4\ ]$ $Vor = [\infty\ \infty\ \infty\ \infty\ \infty\ \infty\ \infty\ \infty\ \infty]$

Lankai nustatomi nuosekliai, kai i = 1, 2, …, n. taigi, vektorius *Vor* užpildomas nuo pirmos iki paskutinės pozicijos. Rezultatas bus toks:

- kiekvienam lankui vektoriuje *Vor* randame atitinkamą generatorių:
  Vor = [2 2 2 2 3 3 3 3 3]
- lankams, kurie turi būti padalyti, nustatome atnaujintą vektorių *B*:
  B = [3 4 1 6 6 5 1 6 6]
- lankams, kurie turi būti padalyti, atnaujinamas ir vektorius *Kaštai*:
  Kaštai = [5 0 0 4 1 2,5]
- užbaigiant, padalyti lankai yra fiksuojami atnaujintame vektoriuje *Koordinatės*:

$$\begin{bmatrix} 8 & 6 \\ 4 & 2 \\ 4 & 9 \\ 6 & 4 \\ 0 & 4 \\ 4 & 4 \end{bmatrix}$$

Nagrinėto pavyzdžio rezultatas:



III Projektas

1) Įrankių ir metodų pasirinkimas

Kuriant naujus programavimo įrankius ir naujas programų konstravimo technologijas, pagrindinės kalbos priemonės papildomos naujomis, kurios kaupiamos specialiose bibliotekose. Tokių bibliotekų sudarymui ir tvarkymui geriausiai tinka objektinio programavimo metodologija. Šiam tikslui pritaikytas C kalbos variantas, pavadintas C kalba su klasėmis. 1983 m. paskelbtas viešai naujos kalbos variantas, kuriam pagal C sintaksę vėliau buvo pritaikytas pavadinimas C++.

Taigi, C kalba yra aukšto lygio programavimo kalba, vadinasi, tą patį galima pasakyti ir apie C++ ir jos klonus. Vienas iš klonų yra C++ Builder aplinka, kurioje galima rengti ir tvarkyti įvairių tipų ir paskirties projektus.

Aš dirbau su C++ Builder 4.0.

Tačiau pagrindinė priežastis, dėl kurios pasirinkau C++ Builder yra ta, kad ši aplinka turi pranašumų grafinės vartotojo sąsajos projektavime lyginant su kitomis sistemomis. Vartotojo interfeisas patogus tuo, kad kai kurie dalykai padaromi automatiškai taip sutaupant daug laiko.

Programos sąsajos kūrimui C++ Builder naudoja biblioteką VCL (Visual Components Library), kuri sukurta 1995 m. Delphi vardu vadinamai objektinei Paskalio kalbos atmainai ir vėliau pritaikyta C++ kalbai.

2) Projekto vykdymo planas

I semestras - literatūros paieška

II semestras - literatūros analizė, jau sukurtų produktų paieška

III semestras – projekto apibrėžimas

IV semestras - projekto įgyvendinimas

3) Pradinis projekto aprašymas

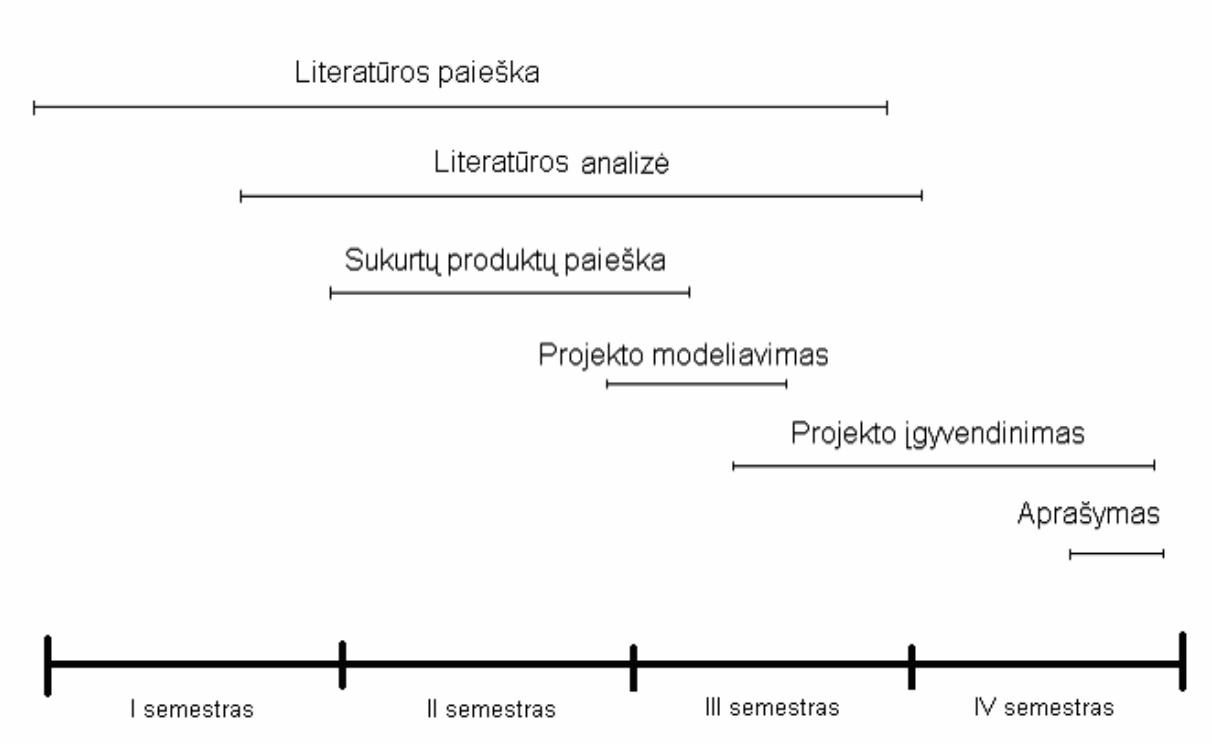
Tarkime, turime žemėlapį. Gatvių tinklą reikia skaidyti į Voronojaus ląsteles pagal Tinklinę Voronojaus diagramą. Pasirinkau Šiaulių miesto žemėlapį, nes tai geriausiai pažįstamas

miestas. Sankryžos apibrėžiamos kaip mazgai, o gatvių atkarpos tarp tų sankryžų (mazgų) - lankai. Konkreti pritaikymo sritis dar nenustatyta. Kitaip sakant, atliksiu padalijimą pagal Tinklinę Voronojaus diagramą. Naudosiu modifikuotą Deikstros algoritmą.

Kitas žingsnis – padalijimas pagal Voronojaus diagramą. Šiaulių žemėlapis bus suskaidytas į sritis taip, kad tam tikras srities taškas bus artimiausias visiems tos srities taškams. Planuojamas sukurti efektyvus algoritmas.

IV Darbo eiga

1) Darbų eigos grafas



2) Problemų ir jų sprendimų aprašymai ir pagrindimai

Pagrindinė problema ta, kad nebuvo įmanoma rasti jokios literatūros lietuvių kalba. Visą teorinę medžiagą verčiau iš anglų kalbos, modelių pavyzdžius, pritaikymus nagrinėjau anglų kalba.

Kadangi modelį nusprendžiau pritaikyti Šiaulių miestui, teko ieškoti Šiaulių žemėlapio. Labai sunku buvo rasti tinkamą žemėlapį Žemėlapiai labai margi, informatyvūs, per daug smulkūs. Pavyko rasti vieną beveik atitinkantį mano reikalavimus.

Kad modelis būtų realus, rankiniu būdu teko apibrėžti visus keliavimo kaštus nuo vieno mazgo iki kito. Reikėjo išskirti vienkrypčius lankus ir tuos, kuriuose keliavimo sąlygos sudėtingesnės (kaštai didesni). Dėl šios priežasties teko labai smulkiai išnagrinėti Šiaulių miesto transporto tinklą.

3) Galutinio projekto stovio aprašymas

Projektas sudarytas iš dviejų dalių.

Pirmoji dalis – Taksi parko projektas Šiaulių mieste. Šiaulių mieste žinomos aštuonios taksi automobilių stovėjimo aikštelės. Kiekviena stovėjimo aikštelė tampa generatoriumi. Taip suformuojami 8 generatoriai. Jie žymimi skirtingomis spalvomis. Tinklinių Voronojaus diagramų pagalba Šiaulių miestas suskaidomas į aštuonias sritis – Voronojaus ląsteles. Kiekvienam generatoriui ieškomi artimiausi mazgai (gatvių susikirtimo taškai) ir lankai (gatvės), kurie ir sudaro ląstelę. Tokiu būdu akivaizdžiai matoma, iš kurios taksi stovėjimo aikštelės labiausiai verta važiuoti į bet kurį žemėlapyje apibrėžtą tašką. Visi mazgai ir lankai įgauna atitinkamo generatoriaus spalvą. Šiaulių žemėlapis nuspalvinamas 8 spalvomis pagal atitinkamas taksi stovėjimo aikšteles ir joms priklausančias gatves bei gatvių susikirtimo taškus. Šiaulių miesto žemėlapis tam pačiam Taksi parko modeliui suskaidomas ir pagal Voronojaus diagramą. Juodos atkarpos vaizduoja plokštumos Voronojaus diagramą, kas leidžia mums palyginti tarpusavyje abi diagramas.

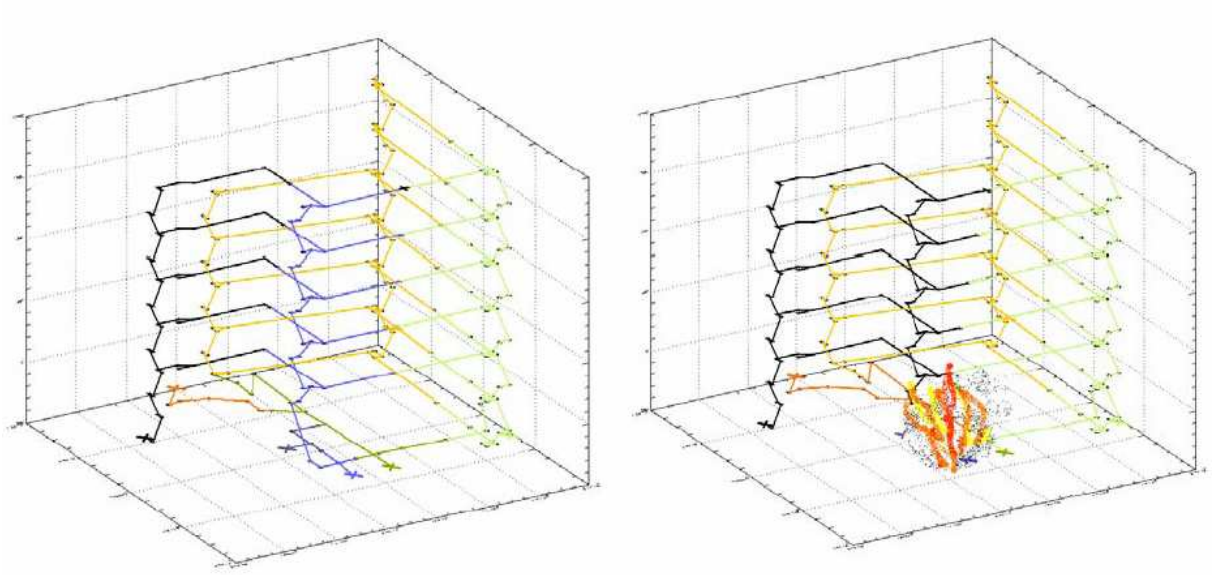
Antroji dalis – duotas Šiaulių miesto žemėlapis, kuriame apibrėžti lankai ir mazgai (gatvės ir jų susikirtimo taškai). Vartotojui suteikiama galimybė keisti duotus duomenis, įvesti generatorius, pamatyti rezultatus ir juos palyginti. Apibrėžus modelį, galima nubrėžti plokštumos Voronojaus diagramą ir Tinklinę Voronojaus diagramą.

4) Patarimai, pastebėjimai, rekomendacijos

Savo darbe aš nagrinėjau dvimatį tinklą. Tačiau egzistuoja ir labai reikšminga Voronojaus diagramų pritaikymo galimybė trimačiame tinkle. Tinklinės Voronojaus diagramos skaičiavimas vidaus tinkle yra itin svarbus nelaimės atveju. Tarkime, kad ugnis užblokavo pagrindinį įėjimą/išėjimą. Visur paplito ugnis ir dūmai.

Pastatas privalo būti evakuotas, bet daugelyje pastato aukštų nurodomas būtent tas užblokuotas pagrindinis išėjimas. Iš esmės, pastato projektuotojai planuodami atsarginių išėjimų kryptis, skaidė aukštus pagal Voronojaus diagramą. Pagrindinis išėjimas apibrėžė sritį. Bet dabar, kai liepsnoja ugnis, pasikeitė pasiekiamų atsarginių išėjimų (generatorių) skaičius, taigi, visiškai pasikeitė Tinklinė Voronojaus diagrama. Žmonėms, esantiems buvusio pagrindinio išėjimo srityje, iškilo svarbus klausimas: kuris atsarginis išėjimas dabar yra sekantis iš esamos pozicijos?

7 pav. Vidaus Tinklinė Voronojaus diagrama, kur visi išėjimai - generatoriai



Paveikslėlyje 7 kairioji pusė demonstruoja įprastą Tinklinę Voronojaus diagramą, o dešinioji pusė rodo dabartinę. Paprasta suvokti, kad pagrindinio išėjimo užblokavimas padidina juodo generatoriaus sritį. Normaliomis aplinkybėmis juodas generatorius esti kaip atsarginis išėjimas yra 308-iems pastato aukštų metrams (15 %). Kai pagrindinis išėjimas užblokuotas, šis skaičius išauga iki 548 metrų (27 %). Šis vidaus tinklas nebėra plokštuminis ir nebeįmanoma lyginti su plokštumine Voronojaus diagrama.

Tai, mano manymu, tinkama sritis tolesniam šios temos vystymui.

V Išvados

1. Pateikti modelių pavyzdžiai

2. Sukurtas konkretus modelis

    - Realizuotas sričių padalijimas pagal Voronojaus diagramą. Naudotas Fortune algoritmas.
    - Plokštumos Voronojaus diagramos principas transformuotas į tinklus. Šiam tikslui panaudotas modifikuotas trumpiausio kelio Deikstros algoritmas. Modifikuota versija skaičiuoja mažiausius keliavimo kaštus nuo pasirinktų mazgų – Voronojaus generatorių – iki visų kitų tinklo mazgų, ir apibrėžia artimiausią generatorių kiekvienam mazgui. Taškams, einantiems išilgai lanko, gali būti apibrėžtas kitas generatorius. Vadinasi, Tinklinė Voronojaus diagrama apima visus tinklo taškus. Modifikacija pateikta dokumente, realizuota testavimui. Modifikacija pritaikyta vienarūšiams tinklams.

3. Realizuota galimybė vartotojui sukurti savo modelį apibrėžtoje plotmėje

VI Literatūra

1. Franz Aurenhammer, Rolf Klein. Voronoi Diagrams www.pi6.fernunihagen.de/publ/tr198.pdf
2. Berg de M., M.van Kreveld, M.Vermars, O.Schwarzhopf, „Computational Geometry: Algorithms and Applications“, Springer-Verlag, Berlin, 1997.
3. L.J.Guibas, D.E.Knuth and M.Sharir. Randomized Incremental Construction of Delauny and Voronoi diagrams. Algorithmica 7: 381-413, 1992.
4. S.Fortune. Sweepline Algorithms for Voronoi diagrams. Algorithmica 2: 153-174, 1987.
5. Čuk Roman. Construction of Voronoi diagrams using Fortune‘s method: A look on an Implementation.
6. www.csie.ndhu.edu.tw
7. Margot Graf, Stephan Winter, S., 2003: Netzwerk-Voronoi-Diagramme. Österreichische Zeitschrift für Vermessung und Geoinformation, 91(3):166-174. (Network Voronoi Diagrams, english translation) www.sli.unimelb.edu.au/winter/pubs/graf03network.pdf
8. Oswin Aichholzer, Franz Aurenhammer. Voronoi Diagrams – Computational Geometry‘s Favorite www.igi.tugraz.at/telematik/tele1-02_aich-favor.pdf
9. Aurenhammer, F.; Klein, R., 2000: Voronoi diagrams. In: Sack, J.-R.; Urrutia, J. (Eds.), Handbook of Computational Geometry, Elsevier Science Publishing, Amsterdam, pp. 201-290.
10. Dijkstra, E.W., 1959: A note on two problems in connexion with graphs. Numerische Mathematik, 1: 269-271.
11. C++ ir C++ Builder pradmenys, Antanas Vidžiūnas, Kaunas 2002
12. www.borland.com
13. Okabe,A.; Okunuki, K.; Funamoto, S., 2002a: SANET: A Toolbox for Spatial Analysis on a Network, Center for Spatial Information Science, University of Tokyo, http://okabe.t.u-tokyo.ac.jp/okabelab/atsu/sanet/sanet-index.html.

VII Anotacija

Voronojaus diagramos ir jų taikymai
L.Žvikaitė

Šiame darbe aprašyta Voronojaus diagrama ir Tinkline Voronojaus diagrama.

Deikstros trumpiausio kelio algoritmas modifikuotas taip, kad apskaičiuotų trumpiausius kelius tuo pačiu metu nuo skirtingų Voronojaus generatorių. Pirmas rezultatas – tinklo suskaidymas (padalijimas) per susikirtimo taškus. Antras rezultatas - lankai priskiriami tinklo generatoriams, ypač atsižvelgiant į jų kryptį ir asimetrines reikšmes.

Pritaikymai leidžia palyginti Tinklines Voronojaus diagramas su Voronojaus diagramomis. Voronojaus diagramos pritaikytos naudojant Fortune algoritmą.

Pateiktas konkretus pavyzdys Taksi parkui.

Realizuota galimybė vartotojui sukurti savo modelį apibrėžtoje plotmėje.

Voronoi diagrams and their applications
L.Žvikaitė

In these theses are pepresented the Voronoi diagram and Network Voronoi diagram.

The shortest path Dijkstra's algorithm was modified in this way that calculates shortest paths from several Voronoi generators at the same time. The first result - partition of the nodes of the network. The seond result - arcs of the network are attributed to the generators, considering especially their direction and asymmetric costs.

Applications allow compare Network Voronoi diagrams to Voronoi diagrams. For this puspose we modified Fortune algorithm.

We made particular product for Taxi depot.

The user can make his own implementation.