VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
KOMPIUTERIJOS KATEDRA

Baigiamasis magistro darbas

Mobiliųjų objektų indeksavimas duomenų bazėse

Atliko: 2M komp. modeliavimo kurso studentas
Saulius Tamošiūnas (parašas)
Darbo vadovas:
doc. Algimantas Juozapavičius (parašas)

Vilnius
2006

Turinys

Anotacija

Pagrindinis šio darbo tikslas yra išnagrinėti judančių objektų indeksavimo duomenų bazėse problemas, siūlomus sprendimus bei palyginti keleto iš jų veiksmingumą. Įvairiais pjūviais buvo lyginami praeities duomenis indeksuojantys R ir iš jo išvesti STR bei TB medžiai. Eksperimentai atlikti naudojant sugeneruotus judančių objektų duomenis. Gauti rezultatai parodė, kad indeksų veiksmingas priklauso nuo tam tikrų sąlygų ir aplinkybių, kuriomis jie naudojami.

Summary

Indexing mobile objects in databases

Over the past few years, there has been a continuous improvement in the wireless communications and the positioning technologies. As a result, tracking the changing positions of continuously moving objects is becoming increasingly feasible and necessary. Databases that deal with objects that change their location and/or shape over time are called spatio-temporal databases. Traditional database approaches for effective information retrieval cannot be used as the moving objects database is highly dynamic. A need for so called spatio-temporal indexing techniques comes to scene. Mainly, by the problem they are addressed to, indices are divided into two groups: a) indexing the past and b) indexing the current and predicted future positions. Also the have been proposed techniques covering both problems. This work is a survey for well known and used indices. Also there is a performance comparison between several past indexing methods. STR Tree, TB Tree and the predecessor of many indices, the R Tree are compared in various aspects using generated datasets of simulated objects movement.

Įvadas

Sparčiai besivystant techninei įrangai atsiranda vis daugiau mobilių e-paslaugų. Įrenginiai tobulėja ne tik savo dydžiu, bet ir darbo sparta. Auga bevielio ryšio ir pozicionavimo sistemų vartojimas. Mobilios e-paslaugos turi būti jautrios situacijoms, todėl pozicionavimas joms yra svarbu, kadangi vartotojo vieta yra svarbus aspektas. Tokios paslaugos apima eismo kontrolės ir valdymo, turizmo, su saugumu susijusias paslaugas, taipogi vietovėmis pagrįstus žaidimus, kurie sujungia fizines ir virtualias erdves. Judantys objektai tokioms e-paslaugoms praneša savo pozicinę informaciją (koordinates, kryptį, greitį ir t.t.). Panaudodamos šią ir kitą informacija pastarosios suteikia specialų funkcionalumą.

Dėl laiko komponentės, duomenų bazės turi tvarkyti didelius duomenų kiekus sukauptus per ilgą laiko tarpą. Kadangi duomenų kiekiai yra dideli, jie turi būti saugomi diskuose. Paprasčiausias sprendimas norint atlikti paiešką tokiuose duomenys būtų perrinkti visus objektus iš eilės ir atrinkti užklausos sąlygą tenkinančius. Tačiau tai labai neefektyvu. Norint pasiekti efektyvų užklausų naudojimą, duomenų indeksavimas yra neišvengiamas. Indeksavimo tikslas yra įgalinti daugybę vartotojų vienu metu efektyviai pasiekti norimą informaciją iš labai didelių duomenų bazių. Tokiu būdu nuskaitoma tik dalis objektų. Bendrai tariant indeksavimas yra būdas kaip išdėstyti duomenis disko puslapiuose, kad efektyviai būtų galima atsakyti į tam tikro tipo užklausas nuskaitant palyginti nedaug disko puslapių. Indeksavimo technologijos tampa ypač svarbios, kadangi duomenų apsikeitimo tarp vidinės ir išorinės atminties santykio tobulėjimas atsilieka nuo spartaus saugomų duomenų kiekio augimo.

1 Judančių objektų indeksavimas

Judantys objektai iškelia naujus iššūkius duomenų bazių technologijoms. Įprastinės duomenų bazėse, laikoma, kad duomenys išlieka pastovūs kol nėra tiesiogiai pakeičiami. Su tokia prielaida pastovaus judėjimo duomenų kaupimas sukeltų daug duomenų keitimų arba netikslių, pasenusių duomenų saugojimą.

Vietoj to, kad saugoti pozicijas, yra saugomos laiko funkcijos, kurios išreiškia objekto poziciją. Tuomet atnaujinimai duomenų bazėje vykdomi tik tuomet, kai keičiasi funkcijos parametrai. Judantiems objektams dažniausiai naudojama tiesinė funkcija, kurios parametrai yra pozicija ir greičio vektorius (duomenų įrašymo į duomenų bazę metu).

Dažniausiai yra nagrinėjamos dvi kartu skirtingos, tačiau kartu ir susijusios indeksavimo problemos. Pirmoji yra dabartinių ir numatomų ateities pozicijų indeksavimas. Antroji – praeities pozicijų (arba trajektorijų) indeksavimas. Taip pat yra pasiūlyta keletas indeksų, apimančių abi šias problemas.

Galima išskirti keletą pagrindinių aspektų nagrinėjant dabartinių ir numatomų ateities pozicijų indeksavimą.

Pirma, metodai gali skirtis pagal duomenų erdvę kurią jie indeksuoja. Laikydami, kad objektai juda *d*-matėje erdvėje (d = 1, 2, 3), jų ateities pozicijos gali būti indeksuojamos d+1 erdvėje. Tuomet užklausos taip pat turi būti pakeičiamos, kad atitiktų duomenų transformaciją. Kitas būdas yra indeksuoti duomenis nekeičiant erdvės išmatavimų, tačiau parametrizuojant indekso struktūrą naudojant greičio vektorius. Tai įgalina naudotis indeksu bet kuriuo ateities metu.

Antras skirtumas yra ar indeksas skaido duomenis (R medžiai), ar skaido erdvę (Quad medžiai). Paprastai priimtinesnis yra pirmas variantas. Tačiau jeigu trajektorijos yra indeksuojamos d+1 erdvėje, o duomenų skaidymo metodas nenaudoja „apkarpymų“ tai gali iššaukti žymius persidengimus.

Galima išskirti trečią skirtumą pagal tai ar indeksas reikalauja periodiško atstatymo, ar ne. Kai kurie indeksai funkcionuoja tik tam tikrą laikotarpį. Kai indeksas nustoja galioti turi būti sukuriamas naujas. Kiti indeksai iš principo gali veikti neribotai, tačiau papratai jų veikimas yra optimizuojamas nustatytam laikotarpiui (vadinamam horizontui). Tačiau laikui bėgant, pastarųjų efektyvumas sumažėja.

1.1 Užklausos

Duomenų bazės, kaupiančios erdvinius objektus vadinamos erdvės ir laiko duomenų bazėmis (angl. *spatio-temporal*). Tokiose duomenų bazėse įdomi objektų tiek dabartinė, tiek praeities, tiek ir numanoma ateities erdvinė informacija. Šios problemos turi prasmę realiame gyvenime, pvz. nustatant numanomas automobilių sangrūdas greitkelyje. Tarkime turime duomenų bazę, kuri realiu laiku kaupia transporto priemonių būvimo informaciją. Tokiai duomenų bazei gali būti pateikta daugybė dominančių užklausų. Pavyzdžiui asmuo per dešimt minučių nori surasti artimiausią viešbutį, arba pervežimų kompanija nori nustatyti artimiausią sunkvežimį, esantį prie tam tikro sandėlio. Užklausos yra skirstomos į porą pagrindinių grupių: *koordinačių* ir *trajektorijų*. *Koordinačių* užklausos apima srities (angl. *range*), artimiausio kaimyno (angl. *nearest neighbour*) ir narystės (angl. *membership*). Atitinkamai kiekviena užklausa būtų tokia: „surasti visus objektus, kurie laikotarpiu nuo t_i iki t_j buvo/yra/bus srityje S“, „surasti objektus, kurie laikotarpiu nuo t_i iki t_j buvo/yra/bus arti vietovės L“ ir „ar laiko momentu t_i objektas o yra duomenų bazėje“. Srities užklausos dar skaidomos smulkiau: kuomet $t_i = t_j$ - laiko pjūvio (angl. *timeslice*), lango (angl. *window*) ir kuomet kinta sritis S - judančios (angl. *moving*) užklausos. *Trajektorijų* užklausas sudaro trajektorijų (visa informaciją apie objekto judėjimą) ir navigacinės (išvestinė informacija – greitis, kryptis).

1.2 Išorinės atminties modelis

Minėtos problemos nagrinėjamos standartiniame išorinės atminties modelyje. Šiame modelyje kiekviena disko operacija (I/O) perduoda B informacijos vienetų. B ir yra puslapio dydis. Algoritmo efektyvumas matuojamas I/O operacijų skaičiumi operacijai atlikti. Jeigu N yra objektų skaičius duomenų bazėje, ir K yra objektų skaičius kurį grąžina užklausa, tuomet minimalus puslapių skaičius reikalingas duomenų bazei laikyti yra $n = [N / B]$ ir minimalus I/O skaičius atsakymui pateikti yra $k = [K / B]$. Pagrindinis sprendžiamas uždavinys yra minimizuoti puslapių skaičių, reikalingą tiek duomenų bazei saugoti, tiek atsakyti į užklausą.

1.3 R medis

Erdviniams objektams saugoti reikalingas indeksas pagal jų erdvines koordinates. Klasikinės vienmačių duomenų indeksavimo struktūros, tokios kaip B medžiai nėra tinkamos erdvinei paieškai. Tikslių reikšmių atitikimo struktūros, tokios kaip *hash* lentelės, taip pat netinka, nes reikia vykdyti srities paieškas.

1.3.1 Struktūra

R medis [Gut84] yra subalansuotas medis, panašus į B medį. Medžio mazgai atitinka disko puslapius ir struktūra yra sukurta taip, kad erdvinė paieška reikalautų nedaug viršūnių apėjimo. Indeksas yra dinamiškas: duomenų įterpimas ir šalinimas gali būti suderinami su paieškomis, taip pat nereikia periodiško indekso atstatymo.

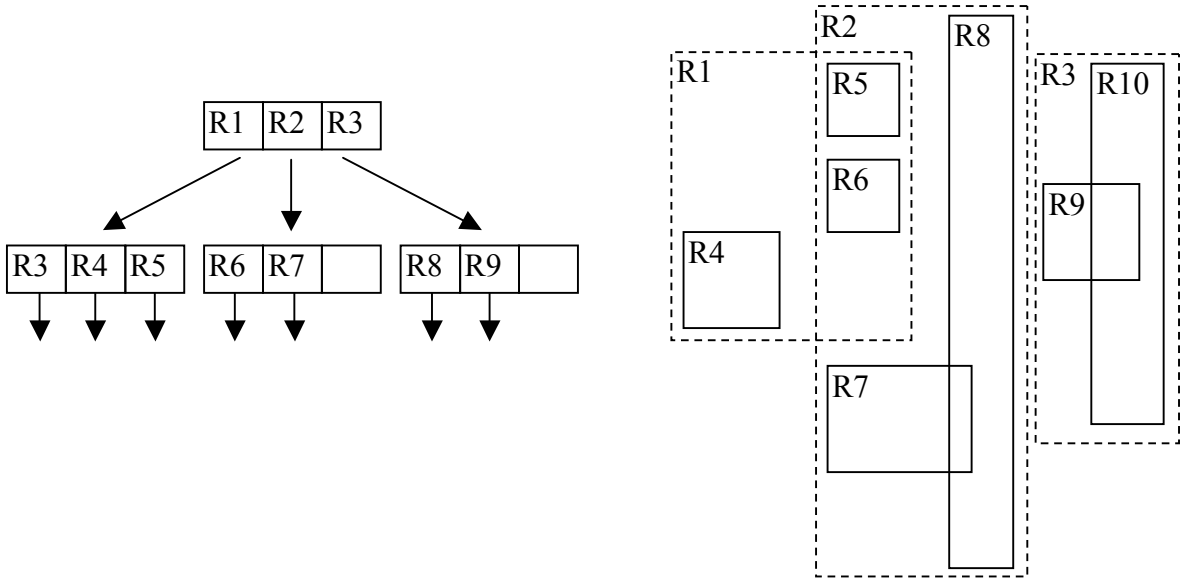
R medžio lapuose yra įrašai sudaryti iš poros – nuorodos į objektą duomenų bazėje ir d-mačio stačiakampio, kuris riboja indeksuojamą objektą. Kitos medžio viršūnės taip pat sudarytos iš poros elementų – nuorodos į pomedį ir stačiakampio, kuris padengia visus pomedžio stačiakampius.

Tegu M yra maksimalus įrašų skaičius viršūnėje, ir $m \leq M / 2$ parametras, nurodantis minimalų įrašų skaičių viršūnėje. Tuomet R medis tenkina šias savybes:

- Kiekvienas lapas turi nuo m iki M įrašų, jei ne tai viršūnė yra šaknis
- Kiekvienam indekso įrašui lape, yra minimalaus dydžio stačiakampis, kuriame yra d-matis objektas
- Kiekviena ne lapo vidinė turi nuo m iki M įrašų, jei ne tai viršūnė yra šaknis
- Kiekviename indekso įraše vidinėje viršūnėje yra minimalaus dydžio stačiakampis kuris gaubia vaikų viršūnių stačiakampius
- Šakninė viršūnė turi bent du vaikus, jei ne tai viršūnė yra lapas
- Visi lapai yra tame pačiame lygyje

1.1 paveikslėlyje pavaizduota R medžio struktūra ir galimi jo stačiakampių ribojimo bei persidengimo atvejai.

R medžio, turinčio N įrašų didžiausias aukštis gali būti $|\log_m N| - 1$, kadangi kiekvienos viršūnės išsišakojimo laipsnis yra mažiausiai m. Maksimalus viršūnių skaičius $\frac{N}{m}+\frac{N}{m^2}+...+1$. Erdvės panaudojimas arba utilizacija blogiausiu atveju visoms viršūnėms, išskyrus šakninę, yra $\frac{m}{M}$. Tai, kad viršūnės turi daugiau nei m įrašų, sumažina medžio aukštį ir padidina erdvės panaudojimą. Jei viršūnės turi daugiau nei tris ar keturis įrašus medis būna labai platus ir beveik visa erdvė sunaudojama lapinėms viršūnėms. Parametras m gali būti keičiamas efektyvumo gerinimui.



1.1 R medžio struktūra ir galimi jo stačiakampių ribojimo bei persidengimo atvejai.

1.3.2 Paieška

Paieškos algoritmas leidžiasi žemyn medžiu, pradedant nuo šaknies. Tačiau pereinamų pomedžių skaičius gali būti daugiau nei vienas, taigi geras blogiausio atvejo veikimas nėra garantuojamas. Nepriklausomai nuo duomenų tipo, keitimo algoritmai išlaiko medį tokioje būsenoje, kad paieškos algoritmai ignoruotų nereikšmingas sritis ir duomenis tikrintų tik šalia reikalingos srities.

Medžio įrašai žymimi *E*, juose esantys stačiakampiai – *EI*, o nuorodos į pomedį arba duomenų objektus – *Ep*.

Algoritmas *Paieška*: turint R medį su šaknine viršūne *T* reikia surasti visus indekso įrašus kurių stačiakampiai persidengia su paieškos stačiakampiu *S*.

S1 [paieška pomedžiuose]. Jei *T* nėra lapas, patikrinti kiekvieną jo įrašą *E* ir nustatyti ar *EI* persidengia su *S*. Visiems persidengimo sąlygą tenkinantiems įrašams *E* taikome *Paiešką* medyje, kurio šaknis yra nurodoma *Ep*.

S2 [paieška lapuose]. Jei *T* yra lapas, patikrinti kiekvieną jo įrašą *E* ir nustatyti ar *EI* persidengia su *S*. Jei taip, *E* yra tinkamas įrašas.

1.3.3 Įterpimas

Įterpiant naujus įrašus į lapus, tikrinama ar nėra viršūnės persipildymo. Jeigu persipildymas yra, viršūnė yra išskaidoma. Jei reikalinga, ši procedūra taikoma aukštyn medžiu.

Algoritmas *Įterpimas*: įterpti naują indekso įrašą *E* į R medį.

I1 [vietos radimas naujam įrašui]. Iškviesti *ParinktiLapą***,** kad parinktų lapą *L* naujam įrašui *E*.

I2 [įrašo įterpimas lape]. Jei lape *L* yra vietos įrašui, jį įterpti. Kitu atveju iškviesti *IšskaidytiViršūnę* ir gauti lapą *L*, lapą *LL* turintį *E*, ir gauti kitus buvusius *L* įrašus.

I3 [pakeitimų perdavimas aukštyn]. Viršūnėje *L* iškviesti *KoreguotiMedį*, perduodant *LL* jei įvyko išskaidymas.

I4 [medžio augimas į aukštį]. Jei pakeitimų perdavimas aukštyn sukėlė ir šaknies išskaidymą, sukurti naują šaknį, kurios vaikai yra dvi gautos viršūnės.

Algoritmas *ParinktiLapą*: surasti lapą, kuriame bus patalpintas įrašas *E*.

CL1 [inicijavimas]. *N* priskiriama medžio šaknis.

CL2 [lapo patikrinimas]. Jei *N* yra lapas, grąžinti *N*.

CL3 [pomedžio parinkimas]. Jei *N* nėra lapas, surasti jo įrašą *F*, tokį kad stačiakampio *FI* padidėjimas tam, kad apgaubtų *EI* būtų mažiausias iš kitų galimų *N* viršūnėje. Jei yra keletas kandidatų, parinkti mažiausią plotą turintį.

CL4 [leidimasis iki lapų]. *N* priskirti vaiko viršūnę, nurodytą *Fp* ir kartoti nuo CL2.

Algoritmas *KoreguotiMedį*: pradedant nuo lapo L kilti medžiu aukštyn iki šaknies, pakoreguojant ribojančius stačiakampius ir jei reikia – skaidant viršūnes.

AT1 [inicijavimas]. *N* priskiriama *L*. Jei *L* buvo suskaidyta, nustatoma papildoma viršūnė *NN*.

AT2 [pabaigos tikrinimas]. Jei *N* yra šaknis – pabaiga.

AT3 [tėvinės viršūnes stačiakampio koregavimas]. *P* yra *N* tėvinė viršūnė. Pakoreguojamas *PI*, kad glaudžiai apribotų visus *N* stačiakampius.

AT4 [viršūnių skaidymas į viršų]. Jei dėl prieš tai buvusio padalinimo yra viršūnė *NN*, tai sukuriamas įrašas E_{NN} kur $E_{NN}p$ rodo į *NN*, o $E_{NN}I$ gaubia visus *NN* stačiakampius. Jei *P* yra vietos, *NN* įterpti į *P*. Jei ne, *IšskaidytiViršūnę* ir gauti *P, PP,* turinčią E_{NN}, ir kitus ankstesnius *P* elementus.

AT5 [kilti aukštyn]. *N* priskiriama *P* reikšmė, jei įvyko padalinimas – *NN* priskiriama *PP* reikšmė. Kartojama nuo AT2.

1.3.4 Šalinimas

Algoritmas *Šalinimas*: pašalinti įrašą *E* iš *R* medžio.

D1 [surasti viršūnę]. Iškviesti *SurastiLapą* ir nustatyti lapą *L*, turintį įrašą *E*. Jei lapas nerastas – pabaiga.

D2 [įrašo pašalinimas]. Iš *L* pašalinamas *E*.

D3 [pakeitimų perdavimas]. Iškviesti *SuglaudintiMedį*, perduodant *L*.

D4 [medžio sutrumpinimas]. Jeigu po koregavimo šaknis turi tik vieną vaiką, vaikas tampa šaknimi.

Algoritmas *SurastiLapą*: R medyje su *T* šaknimi surasti lapą su įrašu E.

FL1 [paieška pomedyje]. Jei *T* nėra lapas, kiekvienam jo įrašui *F* patikrinti, ar *FI* persidengia su *EI*. Kiekvienam tokiam įrašui iškviesti *SurastiLapą* perduodant pomedį, kurio viršūnė nurodoma *Ep*.

FL2 [paieška lapuose]. Jei *T* yra lapas, patikrinti ar jame yra *E*. Jei *E* surastas, grąžinti *T*.

Algoritmas *SuglaudintiMedį*: patikrinti ar lape *L*, iš kurio buvo pašalintas įrašas, liko per mažai įrašų. Jeigu taip, pašalinti *L* ir perskirstyti jo įrašus. Jei reikia, įvykdyti viršūnių pašalinimą aukštyn medžiu. Pakoreguoti ir kiek įmanoma sumažinti visus ribojančius stačiakampius kelyje iki šaknies.

CT1 [inicijavimas]. Priskiriama $N = L$. Sukuriama tuščia aibė *Q*.

CT2 [tėvinės viršūnės suradimas]. Jei *N* yra lapas, pereiti į CT6. Priešingu atveju tegu *P* būna *N* tėvas, ir E_N *N* įrašas viršūnėje *P*.

CT3 [nepilnų viršūnių panaikinimas]. Jei *N* turi mažiau, nei *m* įrašų, ištrinti E_N iš *P* ir *N* pridėti į aibę *Q*.

CT4 [pakoreguoti ribojantį stačiakampį]. Jei *N* nebuvo pašalinta, pakoreguoti $E_N I$ taip, kad glaudžiai ribotų visus *N* įrašus.

CT5 [pakylimas vienu lygiu aukščiau]. Priskirti $N = P$ ir kartoti nuo CT2.

CT6 [betėvių viršūnių įterpimas]. Įterpti visas viršūnes iš aibės *Q* panaudojant algoritmą *Įterpimas*. Pašalintų lapų įrašai įterpiami į kitus lapus, aukštesnio lygio įrašai atitinkamai įterpiami tame aukštesniame lygyje, kad išlaikyti medžio subalansavimą.

1.3.5 Atnaujinimas ir kitos operacijos

Jeigu duomenų įrašas pasikeitė taip, kad pasikeitė ir jo ribojantis stačiakampis, atitinkamas įrašas indekse turi būti pašalintas, pakeistas ir vėl įterptas. Taip teisingai nustatoma nauja jo vieta medyje.

Gali būti panaudojamos ir kitokio, nei minėta anksčiau, tipo paieškos operacijos. Pavyzdžiui surasti visus objektus, esančius paieškos srities viduje, arba visus objektus, kuriu viduje yra paieškos sritis. Tokios užklausos gali būti nesunkiai gaunamos modifikuojant minėtą algoritmą. R medžiuose taip pat galimas ir srities pašalinimas, kuomet pašalinami visi objektai, esantys nurodytoje srityje.

1.3.6 Viršūnės padalinimas

Tam, kad įterpti naują įrašą į *M* įrašų turinčią viršūnę, reikia paskirstyti *M+1* įrašą į dvi viršūnes. Paskirstymas tūri būti atliktas taip, kad kiek įmanoma labiau išvengti abiejų šių viršūnių apėjimo vėlesnėse paieškose. Kadangi sprendimas aplankyti viršūnę ar ne priklauso nuo to, ar jos stačiakampis persidengia su paieškos sritimi, abiejų stačiakampių plotų suma turi būti minimali.

Tas pats kriterijus buvo taikomas algoritme *ParinktiLapą* nustatant, kur įterpti naują įrašą kiekviename lygyje. Buvo parenkamas mažiausio stačiakampio padidėjimo reikalaujantis pomedis.

Yra keli galimi algoritmai, kaip paskirstyti *M+1* įrašus į naujas viršūnes.

- *Nuodugnus algoritmas*

Greičiausias variantas būtų tiesiog sudaryti visus įmanomus sugrupavimus ir parinkti geriausią. Visų galimybių dydis yra 2^{M-1}. Jeigu turime *M*=50 (kas dažnai yra praktiška, kadangi dvimačiam stačiakampiui atvaizduoti reikia keturių skaičių po keturis baitus, ir jeigu nuorodą taip pat sudaro keturi baitai, tai kiekvienam įrašui saugoti reikalinga 20 baitų, taigi 1024 baitų puslapyje telpa apie 50 įrašų), galimų variantų yra labai daug.

- *Kvadratinės kainos algoritmas*

Šis algoritmas ieško mažo ploto padalinimo, tačiau negarantuoja mažiausio. Kaina yra kvadratinė M ir tiesinė išmatavimų skaičiumi. Algoritmas paima du iš M+1 įrašų ir kiekvieną iš jų priskiria dviems naujoms grupėms. Įrašų pora parenkama tokia, kad jei būtų vienoje grupėje, sunaudotų daugiausia erdvės. Kiti įrašai priskiriami po vieną, taikant specialų metodą.

Algoritmas *KvadratinisPadalinimas*: padalinti *M+1* indekso įrašą į dvi grupes.

QS1 [kiekvienai grupei parinkti pirma elementą]. Įvykdyti *ParinktiPradžią* ir parinkti du įrašus ir priskirti juos grupėms.

QS2 [pabaigos tikrinimas]. Jei visi įrašai priskirti – pabaiga. Jei viena grupė turi tiek mažai įrašų, kad pridėjus likusius gautųsi *m*, pridėti likusius ir baigti.

QS3 [parinkti priskiriamą įrašą]. Įvykdyti *ParinktiSekantį*. Gautą įrašą priskirti tai grupei, kurios ribojančio stačiakampio ploto padidėjimas būtų mažesnis. Jei pasirinkimas nevienareikšmis, priskirti mažesnį plotą turinčiai grupei, priskirti mažiau elementų turinčiai grupei, priskirti bet kuriai grupei. Kartoti nuo QS2.

Algoritmas *ParinktiPradžią*: parinkti po pirmą elementą abiem grupėms.

PS1 [paskaičiuoti neefektyvų sugrupavimą]. Kiekvienai įrašų porai E_1 ir E_2 sudaryti stačiakampį J, apribojantį E_1I ir E_2I ir suskaičiuoti d = plotas(J) – plotas(E_1I) – plotas(E_2I).

PS2 [parinkti neefektyviausią porą]. Parinkti porą su didžiausiu d.

Algoritmas *ParinktiSekantį*: parinkti įrašą iš likusių.

PN1 [įrašo priskyrimo grupei kainos nustatymas]. Kiekvienam nepriskirtam įrašui E paskaičiuoti d_1 – pirmos grupės ribojančio ploto padidėjimas įtraukus EI, ir atitinkamai d_2 – antrosios.

PN2 [parinkti tinkamiausią įrašą]. Parinkti įrašą su didžiausiu skirtumu tarp d_1 ir d_2.

- *Tiesinės kainos algoritmas*

Šis algoritmas yra tiesinis *M* ir išmatavimų skaičiumi. *TeisinisPadalinimas* yra panašus į *KvadratinisPadalinimą*, tik naudoja skirtingus *ParinktiPradžią*. *ParinktiSekantį* tiesiog parenka bet kurį iš likusių įrašų.

Algoritmas *ParinktiPradžią*: parinkti po pirmą elementą abiem grupėms.

PS1 [visomis kryptimis surasti kraštutinius stačiakampius]. Kiekvienai išmatavimo krypčiai surasti įrašą, kurio stačiakampio apatinė siena yra aukščiausiai ir įrašą, kurio stačiakampio viršutinė siena yra žemiausiai. Įsiminti skirtumą.

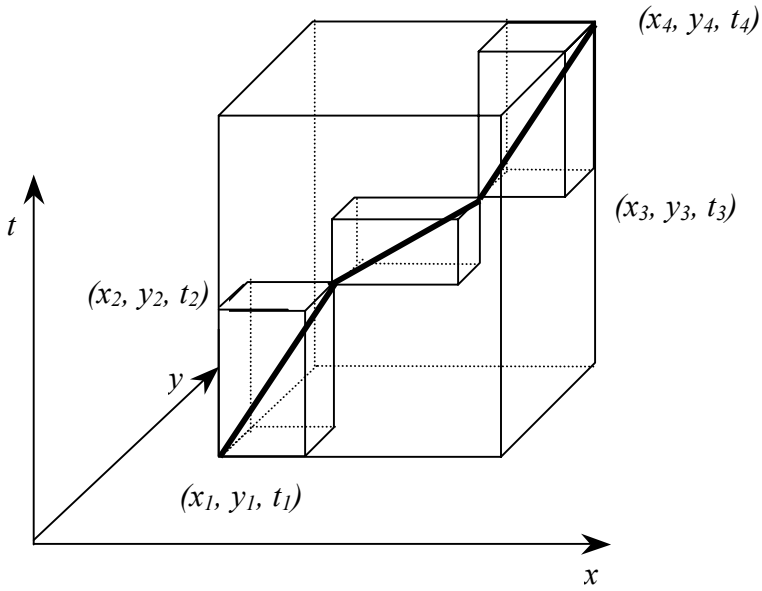
PS2 [pakoreguoti stačiakampių grupės formą]. Normalizuoti skirtumus, padalijant iš visos grupės pločio kiekvienos ašies atžvilgiu.

PS3 [parinkti kraštutinę porą]. Parinkti porą su didžiausiu normalizuotu skirtumu pagal bet kurį išmatavimą.

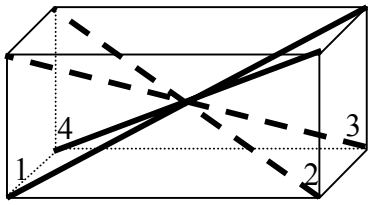
2 Praeities pozicijų indeksavimas

Tradicinis R medžio metodas aproksimuojant tiesių segmentus minimaliais ribojančiais stačiakampiais čia nelabai tinka. 2.1 paveikslėlyje pavaizduota kaip yra neefektyviai MRS sunaudoja erdvę, nors trajektorijos užimama erdvė palyginti yra maža. Dėl to pasitaiko dažni persidengimai, o tai žymiai sumažina indekso veiksmingumą. Kitas svarbus aspektas, į kurį R medžiai neatsižvelgia, tai nustatymas, kuriai trajektorijai priklauso tiesės segmentas.

Šie trūkumai pašalinami modifikuojant R medį. Kaip pavaizduota *paveikslėlyje b* tiesės segmentą MRS gali riboti keturiais būdais. Ši papildoma informacija išsaugoma lapinėse viršūnėse. Taip pat išsaugomas ir trajektorijos, kuriai priklauso segmentas, identifikatorius.



2.1 a. Trajektorijos aproksimavimas mažiausiais ribojančiais stačiakampiais



2.1 b. Galima segmento pozicija ribojančiame stačiakampyje

2.1 STR medis

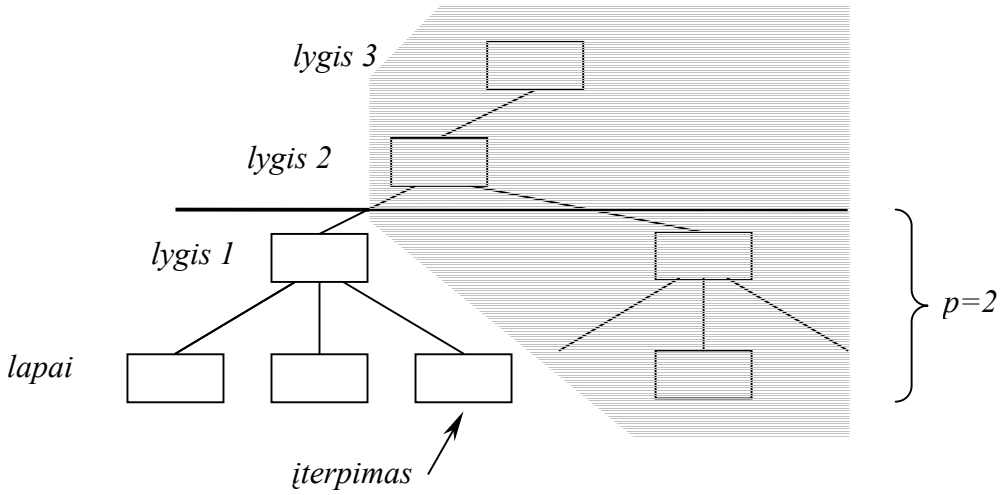
STR medis [PJT00] yra R medžio praplėtimas, pritaikant anksčiau minėtus modifikavimus. Taip pat skiriasi ir įterpimo bei viršūnių skaidymo strategijos.

2.1.1 Įterpimas

STR medžio įterpimo operacija žymiai skiriasi nuo R medžio įterpimo. Kaip buvo minėta anksčiau, įterpimas R medžiuose paremtas mažiausio padidėjimo kriterijumi. STR medžiuose atsižvelgiama ne tik į erdvinį glaudumą, bet ir į trajektorijos išlaikymą, t.y. stengiamasi išlaikyti vienos trajektorijos segmentus kartu, arti vienas kito. Pagrindinis tikslas įterpiant naują segmentą, yra įterpti kaip įmanoma arčiau jo pirmtako trajektorijoje. Taigi STR naudojamas

papildomas algoritmas *SurastiViršūnę*, kuris surandą viršūnę, saugančią minėtą pirmtaką. Jeigu surastoje viršūnėje yra vietos, segmentas yra įterpiamas. Kitu atveju taikoma skaidymo strategija.

Idealiausia trajektorijas indeksuojančio indekso savybė būtų erdvės išskaidymas pagal laiką, tuo pačiu išlaikant trajektorijas. Įterpimo operacijoje naudojamas papildomas, vadinamas išlaikymo, parametras *p*. Jis nurodo rezervuojamų lygių skaičių norint išsaugoti trajektorijas. Kuomet algoritmo *SurastiViršūnę* grąžinama viršūnė yra pilna, algoritmas patikrina, ar *p*-1 tėvinės viršūnės yra pilnos. Jei kuri nors iš jų nėra pilna, lapas yra padalinamas. Jei visos *p*-1 tėvinės viršūnės yra pilnos, vykdomas *SurastiLapą* algoritmas visoms į dešinę nuo įterpimo kelio esančioms viršūnėms (pilka sritis 2.2 paveikslėlyje). Eksperimentais nustatyta, kad geriausias išlaikymo parametras $p = 2$. Mažesnės ar didesnės reikšmės mažina trajektorijų išlaikymą ir indekso veiksmingumą.



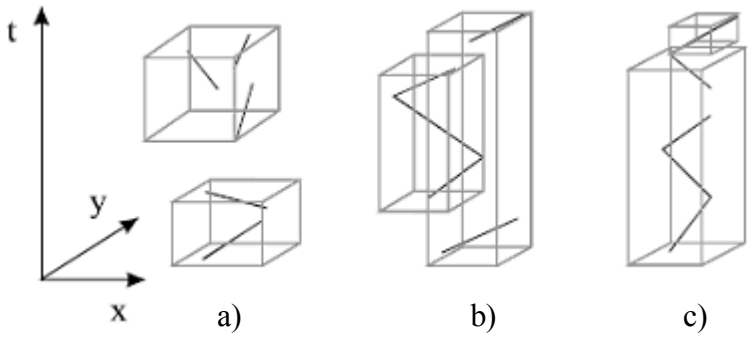
2.2 Įterpimas į STR medį

2.1.2 Padalinimo algoritmas

Kadangi indekso tikslas yra išlaikyti trajektorijas, skaidant lapus reikia išanalizuoti kokio tipo segmentai yra viršūnėje. Bet kurie du lapinės viršūnės segmentai gali priklausyti trajektorijai arba ne, jeigu priklauso, tai gali turėti bendrų taškų arba ne. Taigi viršūnėje gali būti keturių tipų segmentai:

- *nesusieti* – segmentai, kurie nesusiję su jokiu kitu segmentu viršūnėje
- *susieti pradžia* (arba *pabaiga*) – segmentai, kurių pradžios (arba pabaigos) taškas yra kito tos pačios trajektorijos segmento pabaigos (arba pradžios) taškas
- *dukart susieti* – segmentai, kurių abu (pradžios ir pabaigos) taškai yra kito tos pačios trajektorijos segmento (pabaigos ir pradžios) taškai

Taigi galima išskirti tris 2.3 paveikslėlyje pavaizduotus viršūnės padalinimo atvejus. Atveju a) kur visi segmentai *nesusieti*, taikomas *KvadratinisPadalinimas*. Atveju b), kur ne visi, bet bent vienas segmentas yra *nesusietas*, *nesusieti* segmentai patalpinami naujai sukurtoje viršūnėje. Ir galiausiai c) atvejis, kur nėra *nesusietų* viršūnių, paskutinis (pagal laiką) segmentas patalpinamas naujai sukurtoje viršūnėje.



2.3 Galimi viršūnės padalinimo atvejai

Apibendrintas *PadalintiViršūnę(N)* algoritmas būtų toks:

S1. Jei *N* yra vidinė viršūnė, taikyti *PadalintiVidinęViršūnę(N)*, jei ne, taikyti *PadalintiLapą(N)*.

Algoritmas *PadalintiVidinęViršūnę(N):*

SNN1. Naujas įrašas patalpinamas į naują viršūnę, sena paliekama be pakeitimų.

Algoritmas *PadalintiLapą (N):*

SLN1. Jei visi segmentai viršūnėje yra *nesusieti* taikyti *KvadratinisPadalinimas*. Jeigu ne ir jeigu viršūnėje yra *nesusietų* ir kitokio tipo viršūnių, nesusietas viršūnes patalpinti į naują viršūnę. Jeigu ne ir jeigu viršūnėje yra atskirų ir *nesusietų* segmentų, paskutinį atskirą segmentą patalpinti į naują viršūnę.

Stengiamasi vėliausius atskirus segmentus talpinti naujose viršūnėse. Taigi nauji segmentai turi didesnį potencialą būti įterpti į naujas viršūnes. Šis potencialas leidžia sušvelninti minimalaus viršūnių skaičiaus *m* viršūnėje apribojimus. Taigi padalinti vidines viršūnes yra paprasta, kadangi naujam įrašui tiesiog sukuriama nauja viršūnė. Taikant šią įterpimo ir padalinimo strategiją gaunamas trajektorijas išlaikantis indeksas.

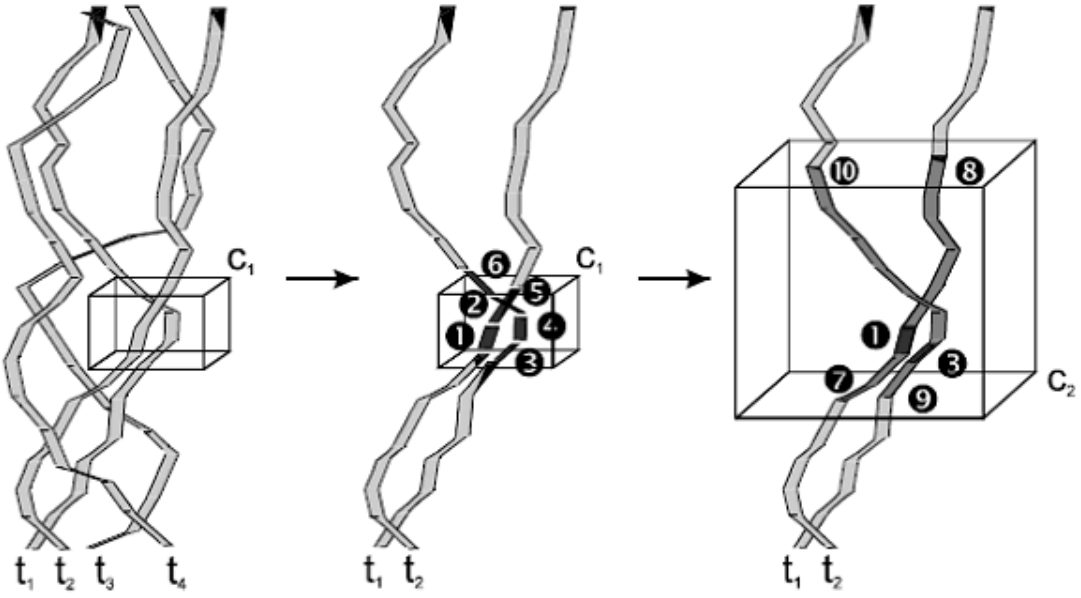
2.1.3 Paieška

Pirmas atliekamas žingsnis yra surasti pradinę segmentų aibę patenkančių į dominančią erdvės ir laiko sritį. Tam taikomi R medžio srities užklausų algoritmai: leidžiamasi nuo medžio

šaknies ir pagal susikirtimo savybes surandami įrašai lapų viršūnėse. 2.4 paveikslėlyje pateiktame paieškos pavyzdyje, naudojant užklausos sritį – kubą c_1 – surandamos du trajektorijos t_2 segmentai (pažymėti 1 ir 2) ir keturi trajektorijos t_1 segmentai (pažymėti 3 – 6). Šie šeši segmentai išskirti tamsesne spalva. Pirmas paieškos etapas baigtas.

Antrame etape išskiriamos trajektorijos dalys. Kiekvienam surastam segmentui ieškomi susiję segmentai, pradžioje tame pačiame, vėliau ir kituose lapuose. Pavyzdžiui pradedama nuo trajektorijos t_2 segmento 1. Surandami du segmentai, kurių pradžia ar pabaiga yra 1 segmento pabaiga ar pradžia. Jeigu segmentų ieškoma kitame lape, naudojama srities užklausa, kurios predikatas yra segmento pradžios ar pabaigos taškas. Aplankydamas lapų lygį, algoritmas tikrina ar segmentas yra susijęs su pradiniu segmentu nurodytu būdu. Taikant šį rekursyvų metodą surandama vis daugiau ir daugiau trajektorijos segmentų. Algoritmas vykdomas tol, kol naujai surasti segmentai patenka už kubo c_2 ribų. Paskutiniai 1-am segmentui surasti segmentai yra 7 ir 8.

Galima problema dėl pakartotino trajektorijų suradimo sprendžiama išsaugant išskirtos trajektorijos ID. Pvz. pradžia parinkus trajektorijos t_2 segmentą 1 ir nustačius trajektoriją, po to parinkus segmentą 2 algoritmas nevykdomas, kadangi su tokiu ID trajektorija jau surasta.



2.4 Paieška

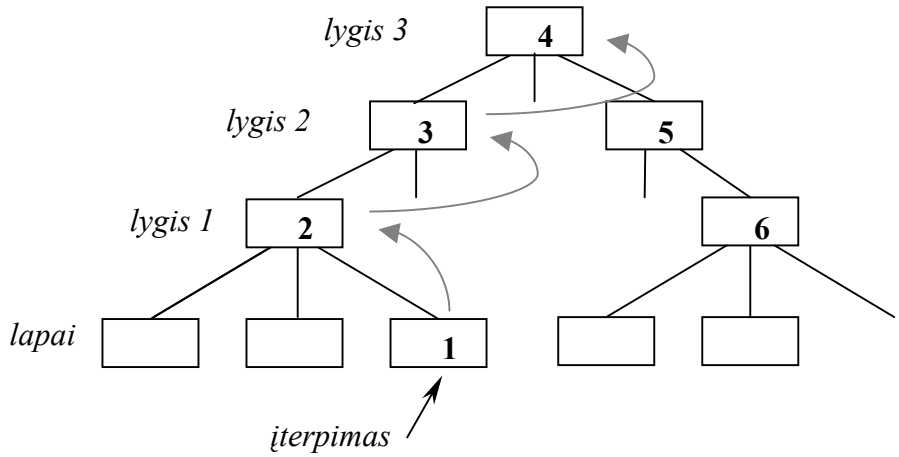
2.2 TB medis

TB medis [PJT00] iš esmės skiriasi nuo prieš tai aprašytų metodų. Pagrindinė prielaida naudojant R medžius yra ta, kad visi įterpiami objektai yra nepriklausomi. TB medžio kontekste, šie objektai yra tiesių segmentai. Tačiau tiesių segmentai yra trajektorijų dalis, ir ši informacija R

ir STR medžiuose išlaikoma netiesiogiai. TB medžiuose laikomasi griežto trajektorijos išsaugojimo. Visi vienos trajektorijos segmentai saugomi viename lape, todėl indeksas vadinamas trajektorijų rinkiniu (angl. *Trajectory Bundle – TB*). Toks būdas įmanomas tik taikant nuolaidas svarbiausiai R medžio savybei – viršūnių persidengimui. Yra ir trūkumas, toks, kad erdvėje arti trajektorijų esantys joms nepriklausantys segmentai bus skirtingos viršūnėse. Didėjant persidengimui kyla ir klasikinės srities užklausos kaina.

2.2.1 Įterpimo algoritmas

Pagrindinis tikslas yra padalinti visą trajektoriją į dalis, taip kad kiekviena dalis turėtų *M* segmentų. 2.5 paveikslėlyje pavaizduota įterpimo procedūra. Skaičiais nuo 1 iki 6 pažymėtos svarbūs žingsniai. Įterpiant naują įrašą, reikia surasti lapinę viršūnę, kuri turi jo trajektorijos pirmtaką. Pradedama nuo medžio šaknies ir aplankoma kiekviena vaikinė viršūnė, kurios MRS persidengia su naujo įrašo MRS. Parenkama ta viršūnė, kurioje esantis segmentas susijęs su nauju įrašu (pirmas žingsnis). Jeigu viršūnė yra pilna, reikalinga taikyti padalinimo strategiją. Tačiau tokiu atveju bus pažeidžiamas griežtas trajektorijos išsaugojimo reikalavimas. Todėl yra sukuriamas naujas lapas. Pavyzdyje medžiu kylama tol, kol randama nepilna viršūnė (žingsniai 2 – 4). Pasirenkamas dešiniausias kelias naujam lapui įterpti. Jeigu yra vietos tėvinėje viršūnėje (žingsnis 6), naujas lapas įterpiamas joje. Jeigu ji pilna, viršūnė yra padalinama sukuriant naują viršūnę lygyje 1, kurios vaikas ir yra naujas lapas. Jeigu reikia padalinimai įvykdomi aukštyn medžiu. Galima pastebėti, kad TB medžiai auga iš kairės į dešinę, t.y. kairysis įrašas yra pirmasis, o dešinysis – pats naujausias.



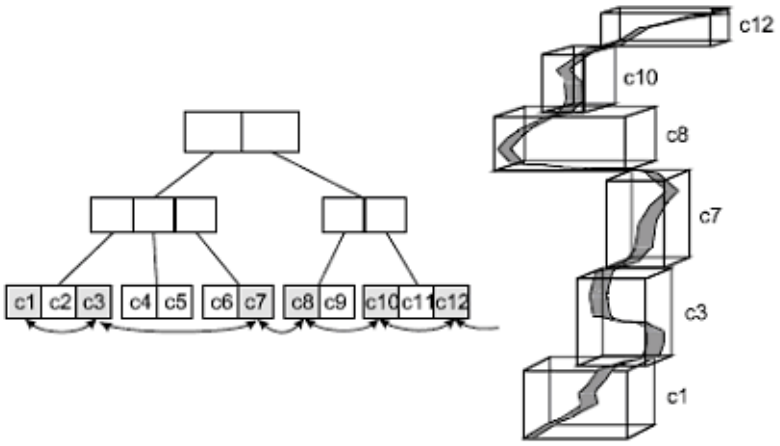
2.5 Paieška

2.2.2 Trajektorijų išlaikymas ir medžio struktūra

Galima sakyti, kad TB medžio struktūra tai aibė lapų viršūnių, kurių kiekvienoje saugoma trajektorijos dalis, išdėstytų hierarchiškai išdėstytų medyje. Kitais žodžiais tariant, trajektorija yra paskirstyta per kelis nesusietus lapus. Paieškoms vykdyti būtina galimybė surasti segmentus

pagal trajektorijos identifikatorių. Tam panaudojama papildoma struktūra – dvikryptis sąrašas. Jis sujungia trajektorijos dalis saugančius lapus taip, kad išlaikytų trajektorijos raidą. 2.6 paveikslėlyje pavaizduota TB medžio struktūros ir trajektorijos dalis. Trajektorija, aiškumo dėlei pavaizduota kaip juosta, padalinta į šešias viršūnes, kurios yra apjungtos sąraše.

Aplankant viršūnes šis sąrašas leidžia surasti trajektoriją (arba jos dalį) su minimalia kaina. Jeigu viršūnės išskleidimo laipsnis yra f, tai f taip pat yra ir trajektorijos dalies dydis toje viršūnėje. Jeigu $f \geq 3$, tai pagal apibrėžimą f-2 segmentai yra abipusiai sujungti, vienas sujungtas pradžia, kitas pabaiga. Tam, kad surasti likusius trajektorijos segmentus tereikia eiti sąrašo nuorodomis.



2.6 Paieška

2.2.3 Paieška

Paieškos algoritmas panašus kaip ir STR medžiuose. Skirtumas tik tame, kaip išskiriamos trajektorijų dalys. Dvikryptis sąrašas padeda surasti susijusius segmentus netaikant rekursyvios srities paieškos. Pradedama nuo pradinių surastų segmentų ir ieškoma arba toje pačioje, arba kitoje lapinėje viršūnėje. Paieška toje pačioje viršūnėje yra triviali. Jeigu reikia ieškoti kitose viršūnėse į jas patenkama einant sąrašo nuorodomis.

Dėl galimo pakartotinio tos pačios trajektorijos suradimo išsaugomi surastų trajektorijų identifikatoriai.

3 Dabarties ir ateities pozicijų indeksavimas

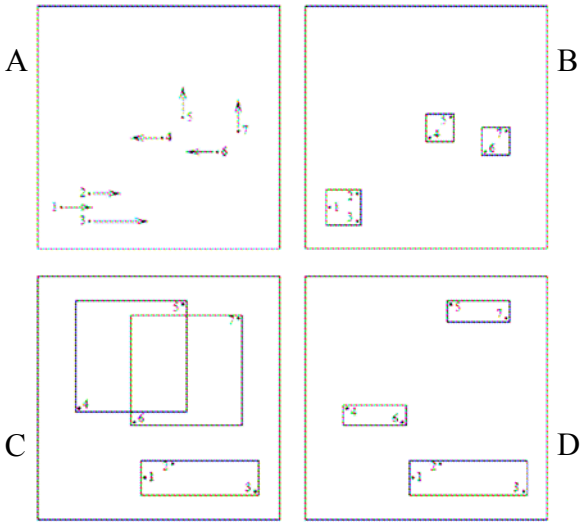
Objekto pozicija laiko momentu t yra žymima kaip $x(t) = (x_1(t), x_2(t),\ldots,x_d(t))$, kur t yra ne anksčiau nei dabartinis laikas. Ši pozicija yra modeliuojama kaip tiesinė laiko funkcija su dviem parametrais. Pirmasis yra objekto pozicija tam tikru laiko momentu t_{ref}, $x(t_{ref})$, ji vadinama atramine pozicija. Kitas parametras yra objekto greičio vektorius $v=(v_1,v_2,\ldots,v_d)$. Taigi $x(t) = x(t_{ref}) + v(t - t_{ref})$. Objekto judėjimas yra nustatomas laiko momentu t_{obs}. Pirmasis parametras

$x(t_{ref})$ gali būti objekto pozicija tuo momentu, arba kita galima pozicija pasirinktu atraminu laiko momentu ir duotu greičio vektoriumi v nustatytu laiko momentu t_{obs} ir pozicija $x(t_{obs})$ nustatyta momentu t_{obs}.

Modeliuojant judančių objektų pozicijas kaip laiko funkcijas galima ne tik nuspėti ateities pozicijas, tai taip pat padeda išspręsti ir dažną duomenų atnaujinimo problemą. Pavyzdžiui objektai gali pranešti savo pozicijas ir greičio vektorius tuomet, kai tikroji pozicija nukrypsta nuo prieš tai praneštų. Tuomet duomenų atnaujinimo dažnumas priklauso nuo judėjimo tipo, pageidaujamo tikslumo ir techninių galimybių.

Atraminė pozicija ir greitis yra naudojami ne tik registruojant judančių objektų ateities trajektorijas, bet ir vaizduojant ribojančių stačiakampių koordinates. 3.1 paveikslėlyje pateiktame pavyzdyje pavaizduota septynių taškų pozicijos ir greičio vektoriai laiko momentu t_0. Šiuo momentu sukuriamas R medis. B dalyje pavaizduotas galimas objektų priskyrimas minimaliems ribojantiems stačiakampiams, viršūnėje turintiems ne daugiau kaip tris objektus. Ankstesni tyrimai parodė, kad tokių dydžių kaip persidengimas, nenaudojama erdvė ir perimetras minimizavimas duoda gerų rezultatų vertinant indekso efektyvumą. Taigi toks priskyrimas yra optimalus. Tačiau nors jis ir geras užklausoms dabartiniu laiko momentu, objektų judėjimas gali žymiai įtakoti tokį priskyrimą.

C dalyje pavaizduota situacija laiko momentu t_3. Kartu padidėję MRS neigiamai įtakoja užklausų efektyvumą. Laikui bėgant jie tik padidės ir toliau blogins efektyvumą. Dėl skirtingų judėjimo krypčių arti buvusių ir tam pačiam MRS priklausiusių objektų (pvz. 4 ir 5), pozicijos sparčiai išsiskiria ir priverčia MRS padidėti.



3.1 Judantys objektai ir juos ribojantys stačiakampiai

3.1 TPR medžiai

3.1 paveikslėlyje pateiktame pavyzdyje užklausų laiko momentu t_3 atžvilgiu geriausia būtų objektus perskirstyti į MRS taip, kaip pavaizduota D dalyje. Reikia pastebėti, kad toks priskyrimas laiko momentu t_0 neduotų gerų užklausų rezultatų. Taigi toks MRS keitimasis iliustruoja laiko parametrizuotus minimalius ribojančius stačiakampius naudojamus TPR medžuose.

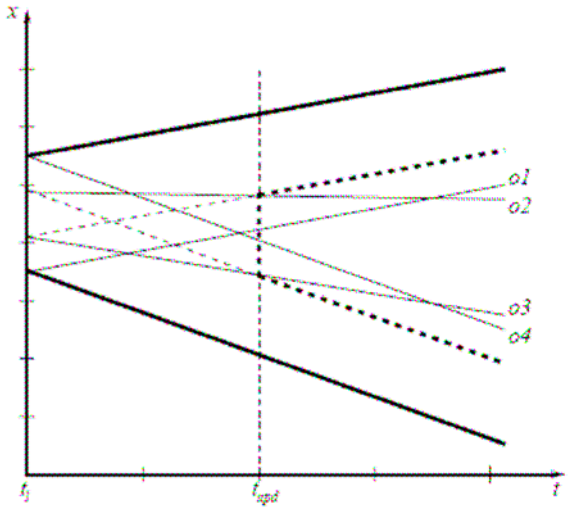
3.1.1 Indekso struktūra

TPR medis [ŠJL+99] yra subalansuotas daugiakryptis medis, turintis R medžio struktūrą. Lapuose esantys įrašai tai pora susidedanti iš judančio objekto pozicijos ir nuorodos į patį objektą. Vidinės viršūnės sudarytos taip pat iš poros elementų: nuorodos į pomedį ir stačiakampio, kuris riboja visus pomedžio judančius objektus arba kitus ribojančius stačiakampius.

Kaip buvo minėta anksčiau, objekto pozicija aprašoma kaip atramos pozicija ir greičio vektorius. *d*-mačiams objektams apriboti naudojami *d*-mačiai stačiakampiai, kurie taip pat parametrizuojami laiku (jų koordinatės taip pat laiko funkcijos). Stačiakampiai taip pat gali judėti pagal juose judančius taškus arba kitus stačiakampius. Kaip ir R medžiuose galima indeksuoti 1-, 2- ir 3-matėje erdvėje. Laiku parametrizuoti ribojantys stačiakampiai riboja visus vidinius objektus ir kitus ribojančius stačiakampius visuomet ne ankstesniu nei dabartiniu laiku.

Egzistuoja suderinamumas tarp to, kaip glaudžiai ribojantis stačiakampis apsupa judančius objektus bei kitus stačiakampius ir tarp ribojančių stačiakampio saugojimo. Būtų idealu naudoti laiko parametrizuotus ribojančius stačiakampius, kurie visada yra minimalūs, tačiau jų saugojimo kaštai yra pernelyg dideli. Vietoj jų TPR medis naudoja konservatyvius ribojančius stačiakampius, kurie tam tikru laiko momentu yra minimalus, tačiau ateityje ši savybė prarandama. Vienmatės erdvės atveju, apatinė konservatyvaus intervalo riba yra nustatoma judėti mažiausiu, ir atitinkamai viršutinė nustatoma judėti didžiausiu iš ribojamų objektų greičiu. Greičiai gali būti teigiami ir neigiami, priklausomai nuo krypties. Tai garantuoja, kad konservatyvūs ribojantys intervalai yra išties visada ribojantys. Reikia pastebėti, kad konservatyvūs ribojantys intervalai niekada nesumažėja. Geriausiu atveju, kuomet visi ribojami objektai turi tokį patį greičio vektorių, juos ribojantis konservatyvus intervalas išlieka vienodo dydžio, net jei jis ir juda. Tokie stačiakampiai vadinami pakrovimo laiko ribojančiais stačiakampiais.

Kad ribojantys stačiakampiai neišaugtų pernelyg daug pageidaujama juos kartais pakoreguoti. Kadangi užklausos indeksui vyksta vėlesniam, nei dabartiniam laikui, iš to seka, kad ribojantys stačiakampiai būtų koreguojami tuomet, kai įvyksta kurio nors iš ribojamų judančių objektų ar stačiakampių atnaujinimas. Tokie stačiakampiai vadinami atnaujinimo laiko ribojančiais stačiakampiais. 3.2 paveikslėlyje pavaizduota keturių judančių objektų trajektorijos ir abiejų tipų juos ribojantys stačiakampiai. Laiko momentu t_{upd} atsiranda glaudesnis ir kartu geresnis ribojantis stačiakampis.



3.2 Pakrovimo ir atnaujinimo (punktyrinės linijos) laiko ribojantys stačiakampiai

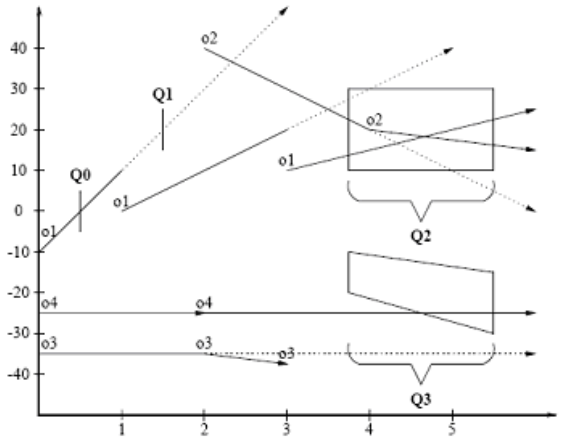
3.1.2 Užklausos

Laiko pjūvio užklausa vykdoma kaip ir R medyje, vienintelis skirtumas yra tame, kad ribojantys stačiakampiai yra apskaičiuojami laiko momentu t_q, naudojamu užklausoje dar prieš tikrinant susikirtimo sąlygą. Taigi ribojantis intervalas pažymėtas kaip (x_1, x_2, v_1, v_2) tenkina užklausą (($|a_1, a_2|$), t_q) tuomet ir tik tuomet, kai $a_1 \leq x_2 + v_2(t_q - t_l) \wedge a_2 \geq x_1 + v_1(t_q - t_l)$.

Atsakant į lango ir judančias užklausas, reikia patikrinti ar (x-,t) erdvėje, užklausos trapezoidas kertasi su ribojančio stačiakampio trajektorijos dalies suformuotu trapezoidu.

Indekso palaikomos užklausos grąžina visus taškus esančius užduotuose regionuose. Tegu R, R_1, R_2 būna d-mačiai stačiakampiai, o t, $t_1 < t_2$ laiko reikšmės ne mažesnės už dabartinį laiką.

Laiko pjūvio – $Q = (R,t)$apibrėžia stačiakampį laiko momentu t. Lango užklausa $Q = (R,t_1,t_2)$ apibrėžia stačiakampį laiko intervalu nuo t_1 iki t_2. Judanti užklausa $Q = (R_1,R_2,t_1,t_2)$ apibrėžia d+1-matį trapezoidą, sujungiantį R_1 laiko momentu t_1 ir R_2 laiko momentu t_2.



3.3 Užklausų tipai

Užklausų iliustracija vienmačiu atveju, kuomet matuojamos temperatūros skirtingos vietovės pateikta 3.3 paveikslėlyje. Q_0 ir Q_1 yra laiko pjūvio, Q_2 lango ir Q_3 judanti užklausos.

Laiko momentą, kuomet atliekama užklausa Q bus žymima $iss(Q)$. Du parametrai, atraminė pozicija ir greičio vektorius, matomi užklausai Q priklauso nuo $iss(Q)$, kadangi laikui bėgant objektų parametrai kinta. Pvz objektas o_1: jo judėjimas laiku $iss(Q) < 1$yra apirėžtas viena trajektorija, laiku $1 \le iss(Q) < 3$ kita, ir laiku $3 \le iss(Q)$ dar kita. Pavyzdžiui atsakymas į Q_1 yra o_1, jeigu $iss(Q_1) < 1$, jokie objektai neatitinka užklausos reikalavimams, jei $iss(Q_1) \ge 1$. Šis pavyzdys iliustruoja, kad užklausos tolimoje ateityje greičiausiai duos neteisingus rezultatus, kadangi užklausos metu numatomos pozicijos laikui bėgant dėl galimų parametrų pasikeitimų tampa vis mažiau tikslios. Taigi realiuose taikymuose užklausos vykdomos tik tam tikrą apribotą laikotarpį, besitęsiantį nuo dabartinio laiko.

3.1.3 Probleminiai parametrai

Yra trys probleminiai parametrai, kurie nulemia indeksavimo ir TRP medžio kokybę:

- Užklausos langas (W) – laikas, reiškiantis kaip toli užklausos gali matyti ateitį. Taigi $iss(Q) \le t \le iss(Q) + W$ pirmo tipo užklausoms ir $iss(Q) \le t_1 \le t_2 \le iss(Q) + W$.
- Indekso naudojimo laikas (U) – laiko intervalas, kuriuo indeksas bus naudojamas užklausoms. Taigi $t_l \le iss(Q) \le t_l + U$, kur t_l yra indekso sukūrimo laikas.
- Laiko horizontas (H) – laiko intervalo, kuriame yra t, t_1 ir t_2 laikai, ilgis. Indekso laiko horizontas yra indekso naudojimo laiko ir užklausos lango suma.

Taigi, naujai sukurtas indeksas turi palaikyti užklausas, kurios trunka H laiko vienetų i ateitį.

3.1.4 Medžio organizavimo euristika

Prieš pradedant nagrinėti duomenų įterpimo algoritmus TPR medyje, aptarsime kaip sugrupuoti judančius objektus į medžio mazgus kad efektyviai palaikytu laiko pjūvio užklausas

turint laiko horizontą *H*. Pagrindinis tikslas yra nustatyti principus arba euristikas kurios taikomos įterpimo ir daugybinio pakrovimo operacijoms.

Aišku, kad kai dydis *H* yra artimas nuliui, galima naudoti jau egzistuojančias R medžio įterpimo ir daugybinio pakrovimo operacijas. Objektų judėjimas ir ribojančių stačiakampių didėjimas tampa nereikšmingas – svarbu tik jų pradinė pozicija ir apimtys. Ir atvirkščiai, kai *H* yra didelis svarbiu tampa objektų grupavimas pagal greičio vektorius. Pageidaujama, kad ribojantys stačiakampiai būtų kaip įmanoma mažesni laiku $[t_l, t_l + H]$ – šiuo intervalu įterpimo operacijų rezultatai gali būti matomi užklausoms (t_l – čia įterpimo arba indekso sukūrimo laikas). Kad tai pasiekti reikia mažinti ribojančių stačiakampių augimo, o kartu ir greičio didėjimo tempus. Vienmatės erdvės atveju greičio didėjimas yra $v_2 - v_1$.

R medžio įterpimo ir daugybinio pakrovimo algoritmų, kurie yra praplečiami judantiems objektams tikslas yra minimizuoti tokias funkcijas, kaip ribojančių stačiakampių plotą, jų perimetrą ir persidengima su kitais stačiakampiais. Mūsų kontekste funkcijos yra priklausomos nuo laiko, ir reikėtų stebėti jų kitimą $[t_l, tl + H]$ laiku. Jeigu turime funkciją *A(t)*, tai šis integralas $\int_{t_l}^{t_l+\mathrm{H}} A(t)dt$ turėtų būti minimizuojamas.

Jeigu *A(t)* yra plotas, tai integralas apskaičiuoja trapezoido, kuris yra ribojančio stačiakampio dalis (*x*, *t*) erdvėje, plotą (tūrį).

3.1.5 Įterpimas ir šalinimas

R medžio įterpimo algoritmas naudoja funkcijas kurios apskaičiuoja ribojančio stačiakampio plotą, perimetrą (skaidant viršūnę), dviejų stačiakampių susikirtimą, ir atstumą tarp dviejų stačiakampių centrų. TPR medyje naudojami tie patys algoritmai tik su viena išlyga: vietoj funkcijų naudojami jų anksčiau minėti integralai.

Ploto, perimetro ir atstumo integralų skaičiavimas yra gan paprastas. Dviejų stačiakampių susikirtimo integralo skaičiavimas yra algoritmo patikrinančio ar jie persidengia praplėtimas. Bet kuriuo laiko momentu kai stačiakampiai susikerta, susikirtimo sritis taip pat yra stačiakampis, kurio apatinė (viršutinė) riba yra nustatoma pagal vieno iš susikertančių stačiakampių apatinė (viršutinė) riba.

Algoritmas padalija laiko intervalą, kurį suranda persidengimo algoritmas į nuoseklius laiko intervalus taip, kad kiekvieno iš jų metu susikirtimas yra apibrėžtas laiko parametrizuotu stačiakampiu. Susikirtimo integralas tuomet yra ploto integralų suma.

Anksčiau minėtas $H = U + W$ parametras yra intuityvus statinėje aplinkoje ir statiniams duomenims. Dinaminėje aplinkoje *W* lieka *H* dalimi, kuris reiškia laiko intervalą, kurio metu yra skaičiuojami integralai įterpimo algoritme. Kokio dydžio turi būti kita *H* komponentė priklauso

nuo atnaujinimo dažnumo. Jeigu pastarasis yra didelis, įterpimas medyje trunka neilgai, ir H neturėtų labai viršyti W.

Integralų panaudojimas yra svarbus žingsnis pritaikant R medžio algoritmus TPR medžiui. Tačiau lieka dar vienas aspektas – padalinimo algoritmas. R medžio padalinimo algoritmas parenka vieną įrašų rinkinį tarp dviejų viršūnių iš kitų galimų rinkinių, kurie yra sugeneruoti pagal taškų surūšiavimą kiekvienoje koordinačių ašyje. TPR padalinimo algoritmas rūšiuoja judančių objektų arba stačiakampių pozicijas skirtingais laiko momentais. Laiku t_l yra naudojami pakrovimo laiko ribojančių stačiakampių pozicijos, o atnaujinimo laiko ribojančių stačiakampių pozicijos naudojamos einamuoju laiku. Galiausiai padalinimo algoritme panaudojamas ne tik objektų pozicijų ir greičio vektorių rūšiavimas. Objektų paskirstymas pagal greičio vektorius leidžia surasti ribojančius stačiakampius su mažesniu plėtimosi dydžiu.

Duomenų šalinimas TPR medyje atliekamas taip pat, kaip ir R medyje. Jei viršūnė persipildo ji yra eliminuojama ir visi įrašai įterpiami iš naujo.

3.2 TPR* medžiai

TPR* medis [TPS03] patobulina TPR medį naudodamas kitus įterpimo ir šalinimo metodus, kurių tikslas yra minimizuoti užklausos kainos lygtį. Nors ši lygtis pritaikyta tik specialiam parametrų rinkiniui, praktikoje pasitaiko žymiai didesnė įvairovė. Taigi iškyla klausimas, kokį pasirinktą parametrų rinkinį optimizuoti. TPR* medis yra optimizuotas taip vadinamai statinio taško intervalo užklausai, kurio MRS ilgis yra 0 kiekviena išmatavimų kryptimi, greičio vektorius {0,0,...,0} o užklausos intervalas {0, H}, kur H laiko horizonto parametras, taip pat naudojamas TPR medžiuose.

3.2.1 Įterpimas

Turint naują įrašą E, įterpimo laiko momentu t_i, panaudojus *ParinktiKelią* algoritmą, surandamas lapas N, kuriama E ir bus talpinamas. Jei N yra pilnas, panaudojus *ParinktiNetinkamus* algoritmą, atrenkama aibė viršūnių, kurios yra pašalinamos iš N ir vėliau naujai įterpiamos. Visi lapai, kurie persipildo šio įterpimo metu, bus padalinti naudojant *PadalintiViršūnę* algoritmą, po kurio nauji įrašai bus priskiriami tėvinėms viršūnėms. Jeigu tai iššaukia ir tėvinių viršūnių persipildymą, elgiamasi analogiškai.

3.2.1.1 Algoritmas *ParinktiKelią*

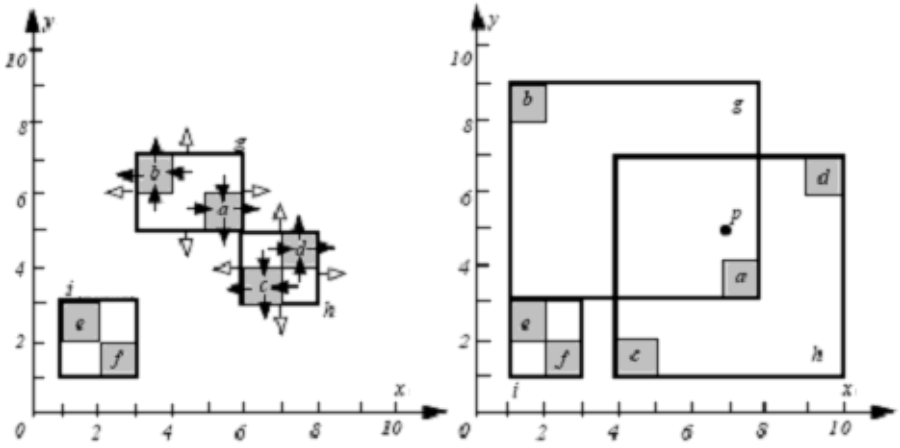
Įterpdamas naują įrašą tradicinis TPR medis parenką šaka su mažiausiu pablogėjimu (kalbant baudų terminais). Šio metodo efektyvumas žymiai sumažėja, jeigu yra keletas šakų su tokiu pat (nuliniu) pablogėjimu. Tai iliusturojama *paveikslėlyje*. Yra šeši lapai – a,b,...,f , jų tėvinės

viršūnės *g*,*h*,*i*, visų vektorių absoliučios reikšmės yra 1. Nuliniu laiko momentu *g* ir *h* nesikerta, tačiau laiko momentu t_2 jie žymiai persidengia. Tarkime, laiko momentu t_2 įterpiame tašką *p*. Aukštesniame lygyje, *g* ir *h* neturi pablogėjimo, kadangi įterpiant *p* nei vienas iš jų nepadidėja. Tokiu atveju algoritmas turi taikyti „lygiųjų“ sprendimo būdą, kuris nėra toks efektyvus. Pateiktame pavyzdyje, parenkamas *h*, kadangi jo MRS yra mažesnis. Jo viduje geriausia viršūnė *p* įterpimui yra *d*. Tačiau geriausias įterpimo variantas yra *a*, kadangi jos MRS padidėjimas yra gerokai mažesnis nei *d*. Laikui bėgant, kartu su vis didėjančiu stačiakampių persidengimu, didėja ir šios problemos rimtumas. Galų gale algoritmo parenkamos šakos tampa parenkamos vos ne atsitiktiniu būdu. R medžiuose, kur MRS laikui bėgant nedidėja, šios problemos nėra.

Algoritmo *ParinktiKelią* prasmė tokia ir yra, kad tarp visų „gerų“ šakų parinktų minimaliausią padidėjimą turinčią.

ParinktiKelią naudoja prioritetinę eile, kurioje saugomi galimi kandidatai. Pateiktame pavyzdyje, pradžioje eilė inicijuojama {[(*g*),0], [(*h*),0], [(*i*),20]}, kur kiekvienas skaičius reiškia *p* įterpimo į atitinkamą viršūnę kainą. Šiuo metu nėra kreipiamasi į *g*, *h*, *i*, kaina suskaičiuojama pagal jų dydžius, saugomus šaknyje. Kiekviename žingsnyje *ParinktiKelią* tyrinėja kelią su mažiausia kaina. Šiuo atveju yra aplankoma viršūnė *g* ir į prioritetinę eile įterpiami du nauji keliai (*a*,*g*) ir (*b*,*g*). Po šios operacijos prioritetinė eilė, atrodo taip: {[(*h*),0], [(*a*,*g*),3], [(*i*),20], [(*b*,*g*),32]}. (*a*,*g*) ir (*b*,*g*) įrašai yra *baigti*, kadangi tai jau yra lapų lygis. Kitas nagrinėjamas kelias yra (*h*). Prioritetinė eilė tampa {[(*a*,*g*),3], [(*d*,*h*),9], [(*c*,*h*),17], [(*i*),20], [(*b*,*g*),32]}. Algoritmas darbą baigia, surastas geriausias kelias yra (*a*,*g*), kadangi jo kaina mažiausia. Viršūnė (*i*) išvis nėra tyrinėjama, kadangi aukščiausiame lygyje turi didesnę kainą.

ParinktiKelią suranda geriausią šaką įterpimui su kreipimosi į keletą papildomų viršūnių kaina. Tačiau tai apsimoka dėl keleto priežasčių. Visų pirma taip gaunama geresnė medžio struktūra, kas padidina užklausų efektyvumą. Antra, bandymai parodė, kad vidutiniškai reikia patikrinti 2-3 kelius, kitų kaina, kaip ir nagrinėtame pavyzdyje būna didesnė. Trečia, *ParinktiKelią* aplanko tik vidines viršūnes, kurios paprastai būna buferyje. Ketvirta, dažniausiai erdvinėse duomenų bazėse kiekvieną duomenų atnaujinimo operaciją seka pašalinimo (ir iškart įterpimo) operacija. Pašalinimo operacija visų pirma reikalauja surasti objektą, kas sudaro žymią atnaujinimo operacijos kainą. Dėl pagerinto užklausų efektyvumo, atnaujinimo operacija reikalauja mažiau sąnaudų nei TPR medžiuose. Panašu skirtumas yra ir tarp R bei R* medžių, dėl sudėtingesnių įterpimo operacijų, duomenų atnaujinimas vyksta žymiai sparčiau.



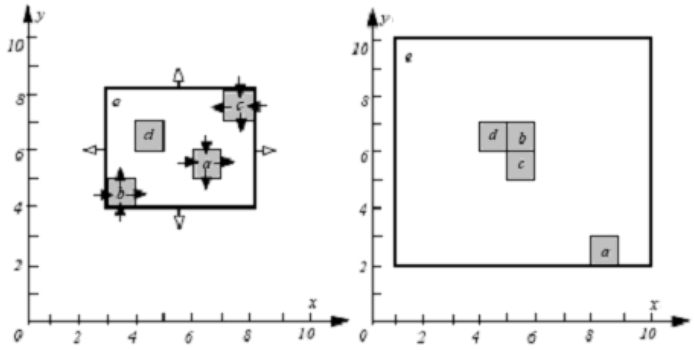
3.4 Judantys objektai ir juos ribojantys stačiakampiai

3.2.1.2 Algoritmas *ParinktiBlogiausius*

Įterpimas į pilną viršūnę iššaukia persipildymą, dėl ko tiek TPR, tiek TPR* medžiuose dalis viršūnės įrašų yra iš naujo įterpiami. Vadovaujantis R* medžio strategija TRP medžiuose įrašai yra parenkami pagal nuotolį nuo jų MRS centų iki atitinkamos viršūnės centro. Pastebima, kad toks veiksmas nesumažina MRS, kas neigiamai įtakoja pakartotino įterpimo efektyvumą.

3.5 paveikslėlyje pavaizduotas lapas *e* su keturiais jame esančiais objektais *a*, *b*, *c*, *d* laiko momentu t_0 ir t_2. Tarkime t_0 laiku įvyksta viršūnės persipildymas ir vienas įrašas turi būti įterptas iš naujo. Reikia pastebėti, kad *b*, *c*, *d* juda *e* centro link, o *a* nuo jo juda tolyn. Atstumas tarp objekto *a* ir *e* centrų tampa didžiausias (laiko momentu [t_0, t_2]), todėl *a* ir išrenkamas pakartotiniam įterpimui.

Objekto *a* pašalinimas neįtakoja *e* plėtimosi, tai nulemia *b* ir *c*. Tai reiškia, kad *a* turi nemažai šansų vėl būti įterptam į *e*, kadangi tai nesukelia pastarojo padidėjimo. Taigi pakartotinas įterpimas buvo bevertis. Bendrai tariant, vietoj tų objektų, kurie nulemia MRS didėjimą, dažniausiai tokiu būdu yra parenkami greit nuo MRS centro tolstantys objektai.



3.5 Judantys objektai ir juos ribojantys stačiakampiai

Algoritmas *ParinktiBlogiausius* ir sprendžia šia problemą, surasdamas aibę įrašų, kurių pašalinimas sumažintų tėvinės viršūnės MRS. Jo tikslas yra surasti objektus, kurie įtakoja MRS didėjimą. Pasirinktam erdvės arba greičio išmatavimui *i*, algoritmas surūšiuoja įrašus pagal jų apimčių pradžios taškus. Jei 3.5 paveikslėlyje pavaizduotame pavyzdyje būtų pasirinkta erdvinė *x* išmatavimo kryptis, tai gautume tokią seką {*b*, *d*, *a*, *c*}, jeigu greičio *x* išmatavimo kryptis – {*c*, *d*, *a*, *b*}. Akivaizdu, kad pirmųjų *n* įrašų pašalinimas garantuos mažesnį tėvinės viršūnės išsiplėtimą *i* kryptimi. Tas pats gali būti taikoma ir surūšiavus objektus pagal jų apimčių pabaigos taškus.

Taigi reikia pasirinkti rūšiavimo išmatavimą ir ar rūšiavimo bus atliekamas pagal pradžios ar pabaigos taškus. Vienas paprastas būdas yra surūšiuoti visomis galimomis kryptimis skirtingais būdais. Tokiu būdu būtų atlikti 4*d* rūšiavimai, tuo tarpu TPR medžiuose rūšiuojama tik vieną kartą. Vietoj to su kiekviena kombinacija yra apskaičiuojamas tėvinės viršūnės nenaudojamo ploto sumažėjimas.

Verta pastebėti, kad visi duomenys nėra tolygiai pasiskirstę, tačiau pasiskirstymą lapuose galima laikyti tolygiu. Taip yra todėl, kad lapo MRS padengia nedidelę erdvės dalį, kurioje objektų pasiskirstymas kinta nežymiai. Aukštesnių viršūnių MRS padengia didesnes dalis, todėl jų turinys yra mažiau tolygus. Šioms viršūnėms naujai įterpiamų įrašų rinkinys nustatomas panaudojant anksčiau minėtą 4*d* rūšiavimą. Bendros atnaujinimo kainos tai nepadidina, kadangi tokių viršūnių persipildymas, palyginus su lapų, pasitaiko žymiai rečiau.

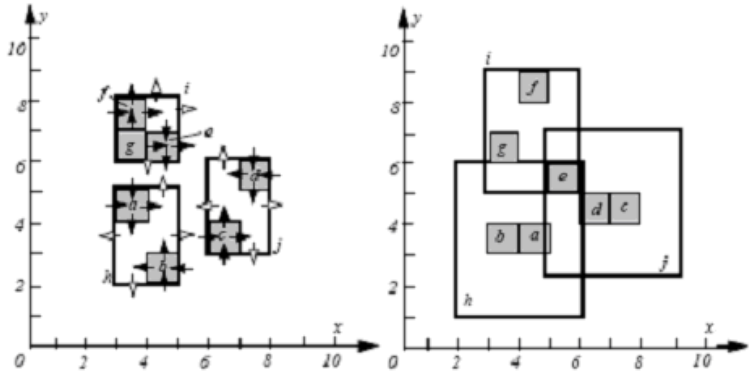
3.2.1.3 Algoritmas *SkaidytiViršūnę*

Panašiai, kaip ir TPR medžiuose, skaidymo algoritmas suskaičiuoja bendrą perimetrą kiekviena išmatavimo kryptimi *i*, tikrindamas visus galimus viršūnės įrašų sugrupavimus ir surūšiuodamas juos pagal apimčių pradžios/pabaigos taškus. Mažiausią perimetrą turinti ašis pasirenkama skaidymo ašimi. Tik TPR* medžiuose viršūnės „perimetras“ yra bendras naujai transformuoto stačiakampio perimetras. Tam yra kelios priežastys. Pirma, daugiakampis su mažu perimetru paprastai yra mažo ploto (atvirkščiai tas negalioja). Antra, tokiu būdu sukuriami stačiakampiai yra labiau „kvadratiški“, tai yra nėra ištęsti vienos kurios nors ašies kryptimi.

3.2.2 Šalinimas

Tam, kad pašalinti objektą *e*, kurio MRS šalinimo laiku yra $eI(t_d)$, šalinimo algoritmas pirma nustato lapą, kuriame yra *e*, panaudodamas *eI* kaip užklausos sritį. Taigi viršūnė *o* yra aplankoma tuomet, ir tik tuomet, jei laiko momentu t_d viršūnės *o* MRS pilnai riboja *eI*. Skirtumas nuo įprastų lango užklausų yra toks, kad paieška užbaigiama iškart, kai tik *e* yra surandamas. Kaip buvo minėta anksčiau, šalinimo kaina sudaro ir atnaujinimo operacijos dalį. 3.5 paveikslėlyje pateiktas pavyzdys su šešiais objektais *a*, *b*,..,*f* ir juos saugančiais lapais *h*, *i*, *j*. Tarkime laiko momentu t_1

objektas *e* pakeičia savo greitį, taigi, prieš įterpiant naują reikšmę ankstesnis jo įrašas indekse turi būti pašalintas. Tam, kad surastų visus lapus algoritmas pradeda nuo šaknies ir ieško tų viršūnių, kurių MRS laiko momentu t_d riboja eI={5,6,5,6}. Pateiktame pavyzdyje visi lapai tenkina sąlygą, todėl turi būti apeinami. Jei viršūnės būtų apeinamos abėcėlės tvarka tai pirma būtų patikrinama viršūnė *h*, po to *i*. Po pašalinimo TPR medis suglaudintų *i*, turinčią *g* ir *h*. TPR* medžiuose pasiūlyta nauja glaudinimo technika, kuri pakoreguoja kelis MRS, jei į jų persidengimo sritį patenka šalinamas objektas. Pavyzdžiui *h*, nors jame ir neturi niekas keistis, tačiau jo MRS yra sumažinamas be papildomos kainos, kadangi šiaip ar taip privalu šaknį išsaugoti diske. Reikia pastebėti, kad *j* MRS suglaudintas nebus, kadangi paieškos metu ši viršūnė nebuvo aplankyta. Toks suglaudinimas gali būti panaudotas ir *ParinktiKelią* algoritme, tačiau su ribotu efektyvumu, kadangi lapinės viršūnės nėra aplankomos. Tolimesnis šalinimo veikimas yra analogiškas TPR medžiui.



3.6 Judantys objektai ir juos ribojantys stačiakampiai

3.3 Indeksavimas naudojant dualias transformacijas

Šio indeksavimo metodai paremti dualia transformacija [KPG+], kur tiek pradinė, tiek vėlesnės objekto pozicijos yra atvaizduojamos kaip taškai daugiamatėje erdvėje. Objektų atvaizdavimas dualioje erdvėje supaprastina indeksavimą bei leidžia taikyti efektyvesnius algoritmus, kurių dėka pasiekiamas geras santykis tarp užklausų ir duomenų atnaujinimo operacijų.

3.3.1 Dualus erdvės – laiko atvaizdavimas

Bendru atveju duali transformacija yra metodas, kuris hiper-plokštumą *h* iš R^d atvaizduoja kaip R^d tašką ir atvirkščiai.

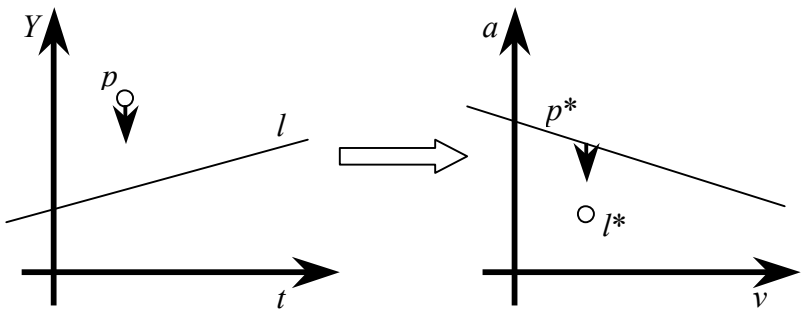
Vienmačiu atveju pradinės plokštumos tiesė (*t*, *y*) atvaizduojama kaip taškas dualioje plokštumoje. Transformuojant tiesę, kurios lygtis yra $y(t) = vt + a$ gaunamas R^2 taškas dualioje plokštumoje, kur viena ašis rodo objekto trajektorijos pasvyrimą, o kita jo atidėjimą koordinačių ašyje. Taigi gauname dualų tašką (*v*,*a*) (tai vadinama Hough-X transformacija). Panašiai taškas

p=(t,y) pradinėje erdvėje atvaizduojamas į tiesę $a(v) = -tv + y$ dualioje erdvėje. Svarbi dualios transformacijos savybė yra tai, kad yra išlaikomas objektų „virš-apačioje“ santykis. Kaip parodyta 3.7 paveikslėlyje, taškas p yra virš dualaus tiesės l taško l*.

Vienmatės erdvės užklausa $[(y_{1q},y_{2q}),(t_{1q}.t_{2q})]$ dualioje erdvėje tampa daugiakampiu. Tarkime turime tašką, judantį teigiamu greičiu. Tuomet taško trajektorija kertasi su užklausa tada ir tik tada, kai trajektorija kerta atkarpą, apibrėžtą taškais $p_1=(t_{1q},y_{2q})$ ir $p2=(t_{2q},y_{1q})$. Taigi, dualus trajektorijos taškas turi būti virš dualios tiesės p^*_2 ir žemiau dualio tiesės p^*_1. Tokia pati idėja taikoma ir neigiamu greičiu judantiems objektams. Taigi užklausa Q dualioje Hough-X plokštumoje yra išreiškiama taip:

- Jei $v > 0$, tai $Q = C_1 \wedge C_2$, kur $C_1 = a + t_{2q}v \geq y_{1q}$ ir $C2 = a + t_{1q}v \leq y_{2q}$
- Jei $v < 0$, tai $Q = D_1 \wedge D_2$, kur $D_1 = a + t_{1q}v \geq y_{1q}$ ir $D2 = a + t_{2q}v \leq y_{2q}$

Lygtį $y = vt + a$ perrašius kaip $t = \frac{y}{v} - \frac{a}{v}$, galima gauti kitokį dualų atvaizdavimą. Taškas dualioje plokštumoje turi koordinates (b, n), kur $b = -\frac{a}{v}$ ir $n = \frac{1}{v}$ (Hough-Y). Koordinatė b yra taškas, kuriame tiesė kerta x ašį pradinėje plokštumoje. Ši transformacija negali būti taikoma horizontalioms tiesėms, o Hough-X – vertikalioms. Taigi, statiniams objektams naudojamos tik Hough-X transformacijos.



3.7 Duali Hough X transformacija

3.3.2 Indeksavimas dvimatėje erdvėje

Dvimatėje plokštumoje judančiu objektų trajektorijos atvaizduojamos tiesėmis trimatėje erdvėje. Problema sprendžiama, išskaidant objekto judėjimą į du nepriklausomus judėjimus skirtingose plokštumose. Kiekvienas judėjimas yra indeksuojamas atskirai.

3.3.2.1 Indekso sukūrimas

Pradedama nuo judėjimo erdvėje (x,y,t) išskaidymo į du atskirus judėjimus (t,x) ir (t,y) plokštumose. Be to, kiekvienoje projekcijoje objektai suskirstomi pagal greitį. Mažu greičiu

judantys objektai išsaugomi naudojant Hough-X transformaciją, likusieji – panaudojant Hough-Y transformaciją.

Skirtingos transformacijos yra naudojamos todėl, kad naudojant Hough-Y metodą maži judėjimo greičiai būtų atvaizduojami į dualius taškus (b,n), kur n koordinatė būtų labai didelė ($n = \frac{1}{v}$). Taigi, kelių mažu greičiu judančių objektų talpinimas į tokį indeksą kaip R* medis, dėl didelių MRS, neigiamai įtakotų indekso veiksmingumą. Šis efektas pašalinamas panaudojus Hough-X transformaciją, kadangi maži greičiai atvaizduojami į dualius taškus (v, a). Atskirti, kurie objektai juda dideliu, kurie mažu greičiu naudojamas pasirinktas slenkstis VT.

Išsaugant dualų tašką indekse, atakingame už objekto judėjimą vienoje iš plokštumų, pvz. (t,x) arba (t,y), išsaugoma informacija apie judėjimą kitoje plokštumoje. Taigi, Hough-Y indekso lapuose saugomi įrašai (n_x,b_x,n_y,b_y). Tai pat ir Hough-X abiejoms projekcijoms saugoma (v_x,a_x,v_y,a_y). Tokiu būdu, į užklausas galima atsakyti panaudojus kurį nors vieną iš indeksų, atsakingų už (t,x) arba (t,y) plokštumas.

Kiekvienoje projekcijoje dualūs taškai (pvz. (b,n) ir (v,a)) yra indeksuojami R* medžius. R* medis yra modifikuotas taip, kad lapų lygyje būtų saugomi tik taškai, o ne stačiakampiai. Tai leidžia saugoti papildomą kitos projekcijos informaciją.

Taigi apibendrinta indekso sukūrimo procedūra būtų tokia:

1. Išskaidyti dvimatį judėjimą į du vienmačius (t,x) ir (t,y) plokštumose.
2. Kiekvienai projekcijai sukurti atitinkamą indekso struktūrą, suskaidant objektų greičius į:
   a. Objektai su $|v| < VT$ išsaugomi naudojant Hough-X transformaciją, objektai su $|v| \geq VT$ išsaugomi naudojant Hough-Y transformaciją,
   b. Informacija apie judėjimą kitoje projekcijoje taip pat išsaugoma kiekviename taške.

3.3.2.2 Atsakymas į užklausas

Dvimatė srities užklausa dualioje erdvėje atvaizduojama į paprastesnę. Ji yra išskaidoma į vienmates užklausas kiekvienoje projekcijoje. Užklausa vykdoma abiejose Hough-Y ir Hough-X transformacijose.

Kaip parodyta 3.8 paveikslėlyje užklausa Hough-Y plokštumoje pateikiama kaip dvi pusplokštumių užklausos. Yra dvi lygiagrečios tiesės $n = 1 / v_{min}$ ir $n = 1 / v_{max}$. Minimali v_{min} reikšmė yra VT. Atsakant į užklausą, užklausos sritis Q^{HoughY} padidėtų šiuo dydžiu $E^{HoughY} = E_1^{HoughY} + E_2^{HoughY}$ (trikampiai paveikslelyje). Analogiškai žymima ir Hough-X plokštumoje, sritis Q^{HoughX}, jos padidėjimas E^{HoughX}. Algoritmas pasirenka tą projekciją, kurioje kriterijus k yra mažiausias:

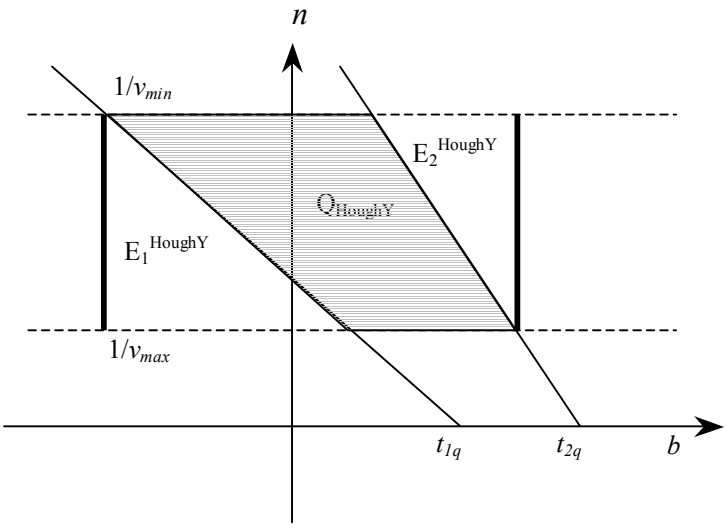
$$k = \frac{E^{HoughY}}{Q^{HoughY}} + \frac{E^{HoughX}}{Q^{HoughX}}$$

Šio euristinio metodo idėja yra tai, supaprastintos užklausos dualioje erdvėje nėra susietos su naudojamo indekso MRS. Taigi pageidaujama užklausą atlikti toje projekcijoje, kur užklausa kiek įmanoma susieta su MRS. To įvertinimas ir yra anksčiau minėtas kriterijus, nurodantis tuščia erdvę.

Kadangi indeksuose saugoma visa judėjimo informacija, ji gali būti panaudojama atrenkant užklausos kriterijus tenkinančius objektus.

Apibendrinta dvimatės užklausos vykdymo procedūra būtų tokia:

1. Užklausa išskaidoma į dvi vienmates užklausas, (*t*,*x*) ir (*t*,*y*) projekcijoms.
2. Kiekvienai projekcijai gaunama duali užklausa.
3. Kiekvienai projekcijai apskaičiuojamas *k* kriterijus ir parenkama ta projekcija *p*, kurioje *k* yra mažiausias.
4. Užklausa vykdoma Hough-X ir Hough-Y plokštumose.
5. Į resultatų sąrašą įtraukiami tik tie įrašai, kurie panaudojus visą judėjimo informacija atitinka užklausą.



3.8 Užklausa dualioje Hough Y erdvėje

4 Praeities, dabarties ir ateities pozicijų indeksavimas

4.1 PCFI$^+$ indeksas

PCFI$^+$ (nuo angl. *Past-Current-Future-Index*) indeksas [LLG+05] yra sukurtas PCFI indekso pagrindu, kuris remiasi SETI ir TPR* medžiais. PCFI$^+$ indeksas yra sudarytas iš dviejų – atminties (vadinamos priekine) ir diskinės – dalių. Panaudojamos SETI medžio savybės: visa erdvė yra suskaidyta į statines nepersidengiančias ląsteles, kurių indeksavimas ir yra pagrindinis tikslas. Judantiems objektams ląstelėse indeksuoti sudaromas TPR* medžių rinkinys – viena ląstelė – vienas TPR* medis. Tai ir yra pagrindinis skirtumas nuo paprasto PCFI indekso, kuris turi tik vieną TPR* medį. Dėl daugybės atnaujinimo operacijų, judančių objektų dabartinė informacija laikoma faile, kuris yra organizuotas kaip *hash* indekso failas vidinėje atmintyje. Yra laikomasi SETI medžio apribojimų – viename puslapyje saugomos tik vienos ląstelės dalys. Puslapių gyvavimo laikui saugoti naudojamas papildomas R* medis.

Atminties dalis sudaryta iš dabartinės informacijos duomenų failo, erdvinio indekso, naudojamo nepersidengiančioms dalims indeksuoti ir aibės TPR* medžių, indeksuojančių įrašus dabartinės informacijos duomenų faile.

Diskinėje dalyje saugomas R* medis. Padalinimui gali būti naudojamas vien-, dvi-, ar trimatis išsklaidytas R* medis. Istorinis puslapio duomenų gyvavimo laikas (*pradžia*, *pabaiga*) yra pažymimas kaip vienmatė atkarpa (*pradžia*, *pabaiga*) ir vadinamas gyvavimo laiko išmatavimu. Vienmačiai išsklaidyti R* medžiai naudojami ir SETI medžiuose. Jeigu naudojamas įprastinis suskirstymo metodas - tinklelis, tai pavyzdžiui Hilberto kreivės gali būti panaudotos dvimatę erdvę transformuoti į vienmatę. Pridėję gyvavimo laiko išmatavimą galima naudoti dvimačius išsklaidytus R* medžius. Jei naudojame kitokius metodus, tai dviejų dimensijų erdvė ir gyvavimo laiko išmatavimas sudaro trimačius išsklaidytus R* medžius.

Kuomet išskiriamas naujas puslapis istorinių duomenų faile, į R* medį įterpiamas naujas įrašas su nenustatyta *pabaigos* reikšme. Ši reikšmė nustatoma, kuomet duomenų puslapis yra pilnai užpildomas vėlesniais segmentais. Lapuose saugoma istorinio duomenų failo puslapio ID, erdvės padalinimo informacija, puslapio gyvavimo laikas. Vidinės viršūnės sudaryto iš poros – nuorodos į pomedį ir gyvavimo laiko, kuris apriboja visus pomedžio gyvavimo laikus.

4.2 Įterpimas

Įterpimo algoritmas efektyviai panaudoja *hash* indekso failą. Visų pirma, patikrinama ar įrašas yra dabartinių duomenų faile, kuriame saugomos visų objektų paskutinės žinomos pozicijos.

Kadangi yra TRP* medžių aibė, tai judėdami objektai gali iš vieno pereiti į kitą. Jei į dabartinių duomenų failą įtraukiamas naujas objektas jau yra einamajame TPR* medyje, sukuriamas naujas segmentas ir patalpinamas istorinių duomenų faile. Priešingu atveju įvykdoma išskaidymo operacija, ir į istorinių duomenų failo skirtingus puslapius įrašomi du segmentai. Atitinkamas įrašas senajame TPR* medyje bus pašalintas ir įtrauktas į naują. Jeigu reikia (duomenų failo puslapis persipildė) pakoreguojamas ir R* medis.

4.3 Šalinimas ir atnaujinimas

Yra trys pašalinimo tipai. Pirmi du – visos trajektorijos pašalinimas, trajektorijos dalies pašalinimas – naudoja SETI medžio pašalinimo operaciją. Jeigu pašalinama trajektorijos dalis, yra įvykdoma laiko tarpo užklausa. Operacija suranda reikalingus dalies duomenis ir pašalina juos iš duomenų puslapio. Jei pasikeičia gyvavimo laikas, pakeičiamas atitinkamas įrašas R* medyje. Tam, kad pašalinti visą trajektoriją, reikia nustatyti visus jos segmentus. Tam naudojamas papildomas B+ indeksas, saugantis trajektorijos ID ir segmentų numerius trajektorijoje. Trečiasis pašalinimo tipas būna tuomet, kai objektas perduoda stebėjimo pabaigos signalą. Tuomet įrašas pašalinamas iš TPR* medžio ir iš dabartinių duomenų failo.

4.4 Užklausos

Paieškos algoritmo laiko intervalo užklausos parametras gali būti įsivaizduojamas kaip trimatis kubas, sudarytas iš laiko srities predikato ir erdvinės srities predikato.

Paieškos algoritmo veikimo principas yra toks:

1. Laiko predikatas lyginamas su dabartiniu laiku. Galimi trys atvejai:

Jeigu laiko predikatas visiškai yra praeityje, vykdomi žingsniai nuo 2.1 iki 2.4.

Jeigu laiko predikatas visiškai yra dabartyje arba ateityje vykdomas žingsnis 3.

Jeigu laiko predikatas apima ir pabaigos, ir dabarties, ir ateities laikotarpius, jis yra išskaidomas į du: pabaigos – taikomas pirmas atvejis, ir dabarties ir ateities – taikomas antras atvejis.

2. Paieška istoriniuose duomenyse

2.1 Kandidatų puslapių paieška

atvejis 1. Ieškoma 1*d* R* medyje. TPR* medžių aibėje atliekama paieška panaudojant erdvės predikatą, gaunamas sąrašas tokių porų (*ląstelės_ID*, *ar_pilnai_padengia_ar_ne*). Atliekama paieška R* medyje panaudojant laiko predikatą. Patikrinama ar kiekvieno gautų rezultatų sąrašo lapo ląstelės ID patenka į pirmą sąrašą. Jeigu ląstelė pilnai padengiama užklausos sritimi, visi segmentai patenkantys į ląstelę talpinami į rezultatų sąrašą. Jeigu ne, puslapio ID patalpinamas į kandidatų sąrašą.

atvejis 2. Ieškoma 2*d* R* medyje. Dvimatė užklausos sritis pertvarkoma į vienmatę. Tuomet sudaromas dvimatis filtras iš pertvarkytos erdvės ir laiko sričių. Atliekama paieška R* medyje ir surandamas kandidatų puslapių sąrašas.

atvejis 3. Ieškoma 3*d* R* medyje. Panaudojamas trimatis filtras iš dvimatės erdvinės srities ir gyvavimo laiko ir surandamas kandidatų puslapių sąrašas.

2.2 Išvalymo žingsnis. Šiame žingsnyje surandami segmentai, kurie persidengia su užklausos sritimi.

2.3 Dublikatų pašalinimas. Šiame žingsnyje sudaromas trajektorijų ID arba segmentų sąrašas.

3. Erdvinis ir laiko filtravimas. Šis žingsnio veikimas toks pat, kaip ir TPR* medyje.

Laiko pjūvio užklausos laiko predikatas sudarytas tik iš vienos reikšmės. Tokia užklausa gali būti atsakyta R* medyje (praeities laikui) arba priekinėje linijoje (TPR* medyje dabarties ar ateities laikui).

Judančioje ir lango užklausoje galimi trys atvejai. Pirmas, jei laiko pradžios t_s ir pabaigos t_e intervalas yra praeityje, paieška vykdoma R* medyje. Antras, jei [t_s, t_e] yra ateityje, tuomet paieška atliekama priekinėje linijoje. Ir trečias, kuomet t_s yra praeityje, o t_e yra eteityje, tuomet užklausos dalis [t_s, *dabar*) yra vykdoma R*, likusi dalis vykdoma priekinėje linijoje.

5 Praeities pozicijų indeksavimas: veiksmingumo palyginimas

5.1 Darbo tikslas ir uždaviniai

Pagrindinis šio darbo tikslas yra **realizuoti ir palyginti kelis istorinių objektų judėjimo duomenų (arba trajektorijų) indeksavimo būdus**. Bus įvertinami R, bei iš jo išvesti STR ir TB medžiai.

Pagrindiniai uždaviniai:

- *Realizuoti minėtus medžius, naudojamus indeksavimui*
- *Sugeneruoti objektų judėjimo duomenis*
- *Sukuri grafinę vartotojo sąsają, leidžiančia interaktyviai stebėti indeksų efektyvumą*
- *Įvairiais pjūviais bei su skirtingais parametrais išanalizuoti indeksų sukūrimą*
- *Įvairiais pjūviais bei su skirtingais parametrais išanalizuoti užklausų vykdymą*
- *Pagal gautus rezultatus nustatyti, kuris indeksavimo būdas yra efektyvesnis*

Vertinimai bus atliekami tiek sukuriant indeksą, tiek ir vėliau ieškant duomenų. Duomenų įterpimui yra svarbu, kiek I/O, viršūnių padalinimo, koregavimo operacijų yra atliekama. Pagrindinis kriterijus įvertinant užklausų efektyvumą yra I/O operacijų skaičius. Bus vertinama dviejų tipų – srities ir trajektorijos nustatymo užklausos.

5.2 Duomenys

Tyrinėjimams ir eksperimentams buvo naudojami sugeneruoti duomenys. Vietoj visiškai atsitiktinio objektų judėjimo buvo pasiriktas arčiau tikrovės esanti modelis, t.y. simuliuojant objektų, pvz. automobilių judėjimą kelių tinklu, apjungiančiu tam tikrus taškus, pvz. miestus. Atsižvelgiama į tai, kad kiekvienas automobilis turi kelionės tikslą – pradžios ir galutinį tašką.

Objektų judėjimas buvo simuliuojamas uždaroje 600×600 vienetų, pvz. kilometrų erdvėje. Šioje erdvėje atsitiktiniu būdu buvo išdėstoma 100 taškų, arba miestų. Jie apjungiami, panaudojus Delaune (*Delaunay*) trianguliaciją. Tokiu būdu gaunamas pilnai jungus grafas (paveikslėlis 5.2.1).

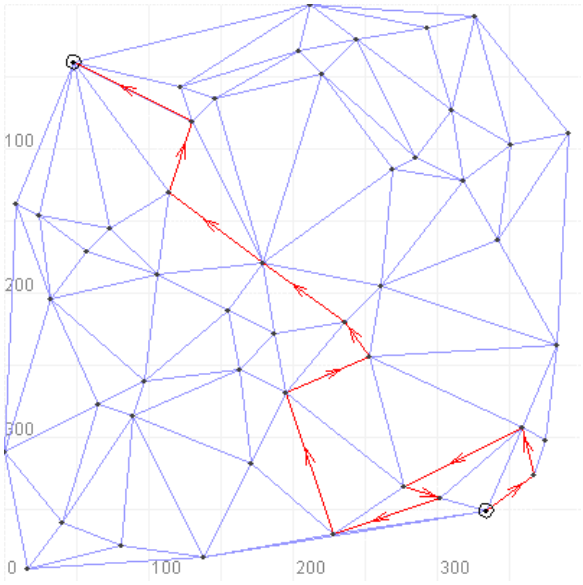
Judantys objektai atsitiktinai išdėstomi miestuose. Jie pradeda judėti vienu metu. Simuliacijos metu objektai nedingsta ir naujų neatsiranda. Kiekvienam objektui atsitiktiniu būdu nurodoma per kiek miestų jam važiuoti – galimos reikšmės kito nuo 20 iki 40 miestų. Kiekviename mieste atsitiktinai parenkamas sekantis galimas keliu iš einamojo sujungtas miestas, išskyrus tuos, kuriuose jau pabuvota. Tokiu būdu gaunamas maršrutas, susidedantis iš nuo 20 iki 40 atkarpų. Kiekvienam objektui suteikiamas maksimalus greitis. Jis svyruoja nuo 80

iki 130 vienetų. Pirmą maršruto šeštadalį, sekančius du trečdalius ir paskutinį šeštadalį objektas juda skirtingais greičiais – atitinkamai 80, 100 ir 75 procentų nuo maksimalaus jam leistino greičio.

Tokiu būdu gaunamas duomenų apie objekto judėjimą rinkinys. Fiksuojama atkarpos pradžios ir pabaigos miestų koordinatės bei laiko momentai, kuomet miestai aplankomi. Surūšiuoti pagal laiką įrašai surašomi faile. Jo struktūra yra tokia:

$id, x_{pradžios}, y_{pradžios}, x_{pabaigos}, y_{pabaigos}, t_{pradžios}, t_{pabaigos}$

Pavyzdžiui turint 1000 judančių objektų, kurie juda per nuo 20 iki 40 miestų vidutiniškai gaunama apie 30.000 įrašų. Taigi tiek pat įterpimo operacijų atliekama sukuriant indeksą.

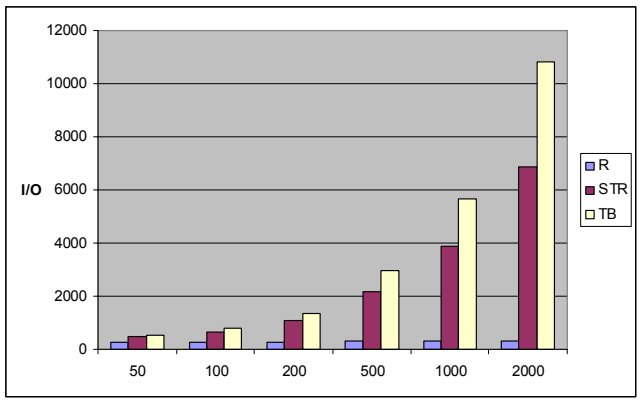


5.2.1 Kelių tinklas, jungiantis 59 miestus, bei galimas objekto maršrutas per 13 iš jų

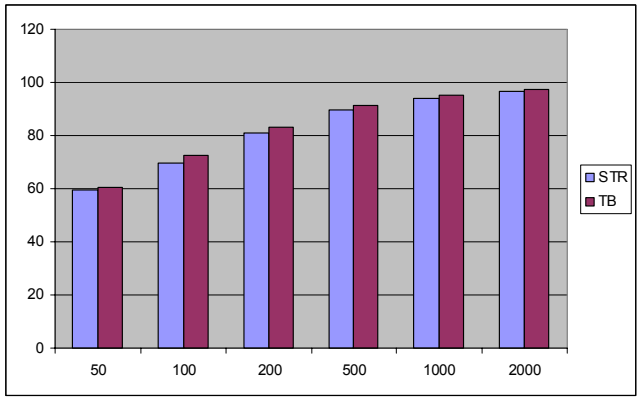
5.3 Indekso sukūrimas

Indeksas sukuriamas nuosekliai vykdant įterpimo į atitinkamą medį komandas. Buvo naudojami 50, 100, 200, 500, 1000 ir 2000 judančių objektų duomenų rinkiniai (vienam objektui vidutiniškai atliekama 30 įterpimo operacijų). Veiksmingumo palyginimas atliktas vertinant vidutinį I/O, viršūnių koregavimo, padalinimo operacijų skaičių vienam objektui.

Nuskaitymo/įrašymo įverčiai pavaizduoti 5.3.1 paveikslėlyje. Žymus skirtumas tarp R ir STR bei TB medžių atsiranda dėl brangios *SurastiViršūnę* operacijos. Ši rekursyvi procedūra atliekama kiekvienai įterpimo operacijai. Didėjant medžio aukščiui didėja ir rekursijos gylis. Procedūros *SurastiViršūnę* atliekamų I/O operacijų ir bendro I/O skaičiaus santykis pavaizduotas 5.3.2 paveikslėlyje. Panašią laiko dalį procedūros užimamas laikas sudaro bendrame indekso sukūrimo laike. Reikia priminti, kad ši procedūra atliekama dėl STR ir TB medžių trajektorijų išlaikymo strategijos, t.y. naujus trajektorijos segmentus stengiamasi patalpinti kaip įmanoma arčiau prie prieš tai buvusių.

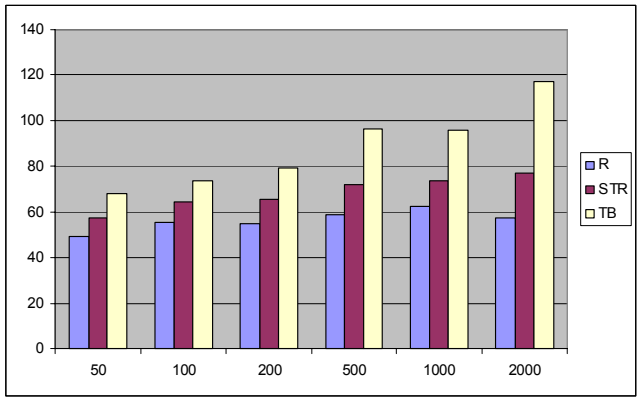


5.3.1 I/O operacijų įvertinimas



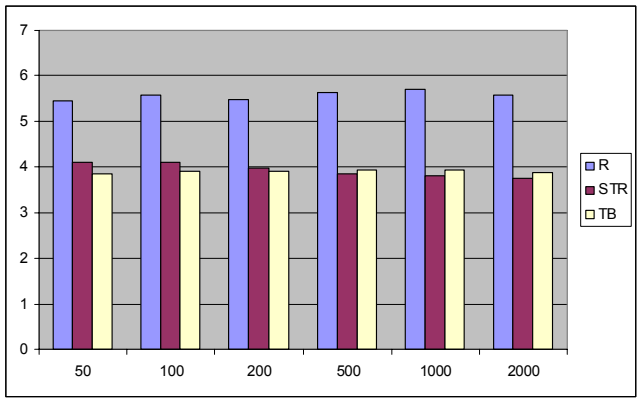
5.3.2 Procedūros *SurastiViršūnę* įtaka bendram I/O skaičiui

Vidutiniškas medžio viršūnių koregavimo skaičius pavaizduotas 5.3.3 paveikslėlyje. Skirtumas tarp R ir STR medžių nėra labai žymus. Tuo tarpu šis rodiklis TB medyje gerokai didesnis. Tai galima paaiškinti tuo, kad TB medyje nėra atsižvelgiama į erdvines medžio savybes, o orientuojamasi tik į trajektorijų išlaikymą. Taikant TB medžio įterpimo strategiją vienos trajektorijos segmentai talpinami vienoje viršūnėje. Taigi vien dėl vis didėjančios laiko komponentės kiekvienas įrašo įterpimas į lapą visuomet pakoreguoja tėvines viršūnes.



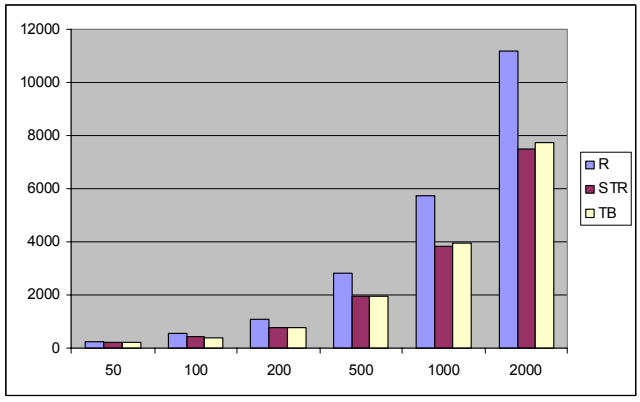
5.3.3 Vidutiniškas viršūnių koregavimo skaičius

Vidutiniškas medžio viršūnių padalinimų skaičius pavaizduotas 5.3.4 paveikslėlyje. Skirtumas tarp R ir STR medžių vėlgi yra nedidelis. Tačiau šį kartą TB viršūnių padalinimas atliekamas du kartus rečiau. TB medyje padalinimas atliekamas tik tuomet, kai įterpiant naują trajektorijos segmentą lapas yra pilnas ir reikia sukurti naują.



5.3.4 Vidutiniškas viršūnių padalinimo skaičius

Dar vienas svarbus aspektas yra medžio erdvės sunaudojimas arba utilizacija. Šis dydis išreiškiamas kaip procentinis viso įrašų skaičiaus (n) ir viršūnių nuliniame lygyje (lapų, l_0) bei viršūnės dydžio M sandaugos santykis – $U = \frac{n}{l_0 * M} * 100\%$. Jis reiškia lapų užpildymą. Prieš tai aprašytuose eksperimentuose buvo naudojamas M = 10 leistino viršūnės dydžio parametras. Utilizacija R medyje svyravo tarp 50% – 60%, STR ir TB medyje 80% – 90%. Tai taipogi susiję ir su bendru medžio viršūnių skaičiumi.

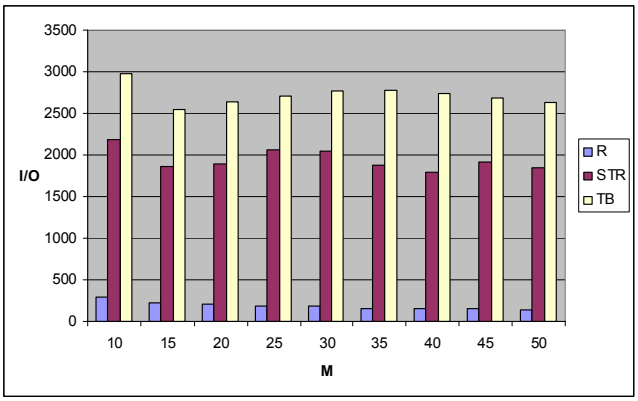


5.3.5 Medžio viršūnių skaičius

Kiti eksperimentai buvo atlikti keičiant viršūnės išsiskleidimo parametrą M. Įterpiant 500 objektų judėjimo duomenis galimos M reikšmės varijavo nuo 10 iki 50. Vidutiniškas I/O

operacijų skaičius R ir STR medžiuose sumažėjo perpus, tuo tarpu TB medyje sumažėjimas siekė 15%.

Medžiui plečiantis į plotį, mažėja jo aukštis. Kartu sumažėja ir minėtos STR ir TB medžių procedūros *SurastiViršūnę* rekursijos gylis. Šios procedūros atliekamų I/O operacijų skaičiaus įtaka bendram I/O skaičiui STR medyje sumažėja keturiais procentais. Tačiau TB medyje dėl didelio persidengimo ši dalis padidėja trim procentais.

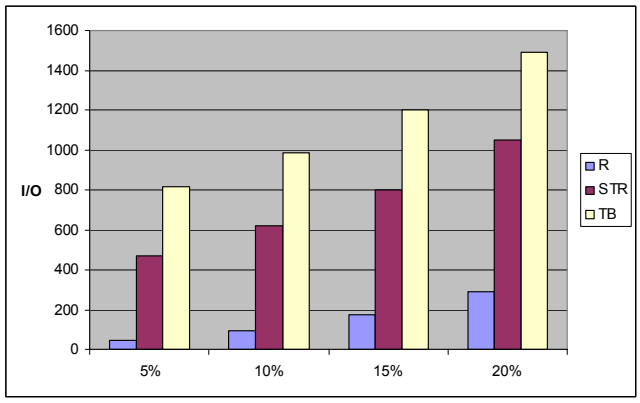


5.3.6 Vidutiniškas I/O operacijų skaičius kintant leistinam viršūnės dydžiui *M*

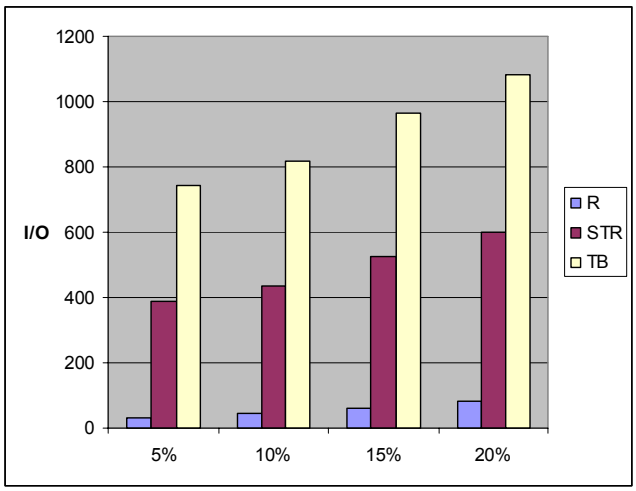
5.4 Užklausos

Įvertinant srities užklausų efektyvumą buvo naudojami keturių (5%, 10%, 15% ir 20%) dydžių užklausų stačiakampių rinkiniai. Procentais nurodomas užklausos stačiakampio dydis, t.y., kurią indeksuojamos erdvės dalį sudaro užklausos stačiakampio ilgis kiekviename išmatavime. Pavyzdžiui 600×600×5000 (x,y,t) erdvėje 5% dydžio užklausos stačiakampis bus 30×30×250. Kiekvienas rinkinys sudarytas iš 1000 tokių stačiakampių. Jie yra atsitiktinai išdėstomi indeksuojamoje erdvėje. Indeksų, indeksuojančių 2000 judančių objektų (apie 60000 įrašų), srities užklausų rezultatai pateikti 5.4.1 paveikslėlyje. Paveikslėlyje 5.4.2 pavaizduoti laiko pjūvio užklausų rezultatai. Dėl palyginti mažų persidengimų geriausi rezultatai gaunami R medyje. Tačiau reikia pastebėti, kad didėjant užklausos lango dydžiui panaudojama vis daugiau I/O operacijų. STR ir TB medžiuose kiekviename žingsnyje įvykdoma 20% daugiau I/O operacijų, tuo tarpu R medyje – 100%, t.y. dvigubai daugiau.

Įdomumo dėlei reikia paminėti, kad atsakant į užklausą, kurios langas apima visą indeksuojamą erdvę apeinamas visas medis. Taigi mažiau nuskaitymo operacijų bus įvykdoma mažiau viršūnių turinčiame medyje. Kaip buvo aprašyta indekso sukūrimo dalyje, R medyje viršūnių yra pastebimai daugiau nei STR ir TB medžiuose.

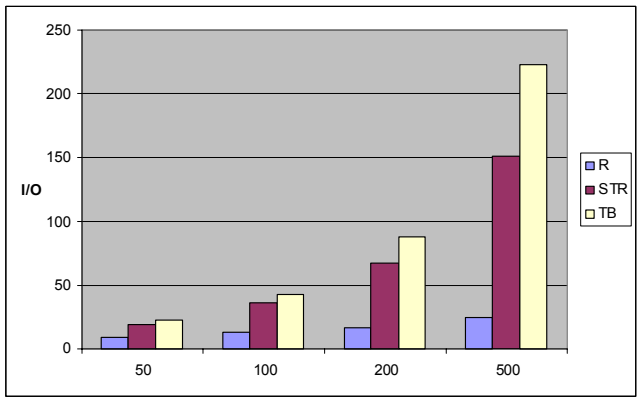


5.4.1 Vidutiniškas I/O operacijų skaičius vienai užklausai kintant užklausos langa dydžiui



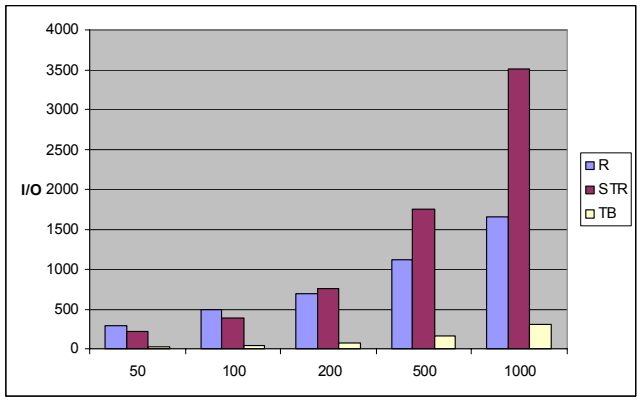
5.4.2 Vidutiniškas I/O operacijų skaičius vienai laiko pjūvio užklausai kintant užklausos langa dydžiui

Esant mažiems judančių objektų kiekiams, skirtumas tarp medžiuose įvykdomų I/O operacijų nėra toks ryškus. Didėjant objektų skaičiui operacijų skaičius didėja tiesiškai, STR ir TB medžiuose beveik pagal geometrinę progresiją. Tai pavaizduota 5.4.3 paveikslėlyje.



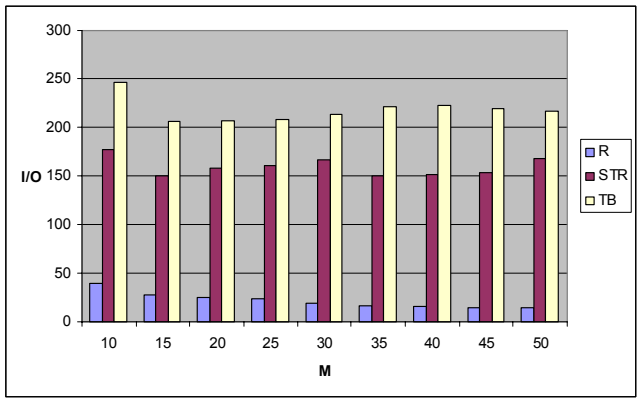
5.4.3 Vidutiniškas I/O operacijų skaičius vienai srities užklausai skirtingiems objektų kiekiams

Trajektorijos suradimo užklausai buvo naudojami indekso sukūrimo duomenys. T.y. kiekvieno segmento id ir MRS buvo panaudoti kaip užklausos predikatai. Rezultatai pavaizduoti 5.4.4 paveikslėlyje. Buvo atliktas papildomas eksperimentas dvigubai sumažinus ir dvigubai padidinus trajektorijos užklausos lango dydį. Tačiau veiksmingumui tai neturėjo žymesnės įtakos. Geriausi rezultatai gauti naudojant TB indeksą. Pagrindinę paieškos operacijos kainą sudaro pirmos užklausą tenkinančios lapinės viršūnės suradimas. Nusileidus nuo medžio šaknies iki surasto lapo toliau keliaujama tik lapais, pasinaudojant nuorodomis. R ir STR medžiuose suradus pirmą segmentą, likusieji surandami rekursiškai, t.y. nuo vėl ieškoma nuo medžio šaknies. Kaip parodė srities užklausų rezultatai, R medis yra veiksmingesnis, todėl ir čia STR medis sunaudoja gerokai daugiau I/O operacijų. Tačiau verta pastebėti, kad esant nedideliam judančių objektų skaičiui (iki 200) STR medis I/O operacijų įvykdo mažiau. Tai susiję su tuo, kad esant mažesniems duomenų kiekiams labiau įgyvendinama trajektorijos išlaikymo strategija, t.y. vienos trajektorijos segmentai mažiau išsibarsto po lapus. O tai sumažina apeinamų viršūnių skaičių.



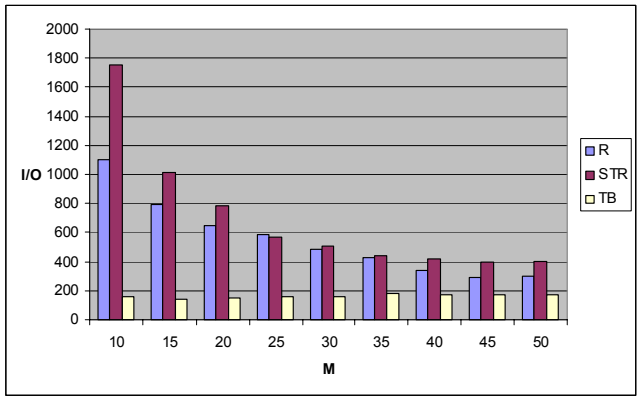
5.4.4 Vidutiniškas I/O operacijų skaičius vienai trajektorijos užklausai

Kiti paieškos užklausų eksperimentai buvo atlikti su R, STR ir TB medžiais, naudojančiais skirtingą leistiną maksimalų viršūnės dydį *M*. 500 judančių objektų su 10% srities užklausos dydžio langu paieškos rezultatai pavaizduoti 5.4.5 paveikslėlyje. Kaip matyti, didėjant *M*, I/O operacijų skaičius R medyje tik mažėja. STR ir TB medžiuose šis dydis svyruoja, tai padidėdamas, tai vėl padidėdamas.



5.4.5 Vidutiniškas paieškos užklausos I/O operacijų skaičius kintant leistinam viršūnės dydžiui *M*

Trajektorijos suradimo užklausos rezultatai pavaizduoti 5.4.6 paveikslėlyje. TB medyje *M* didėjimas pastebimos įtakos neturi. Tuo tarpu R ir STR medžiuose įvykdoma žymiai mažiau I/O operacijų. Nepaisant to, didesnis trajektorijų užklausų efektyvumas išlieka TB medyje.



5.4.6 Vidutiniškas trajektorijos užklausos I/O operacijų skaičius kintant leistinam viršūnės dydžiui *M 10*

5.5 Išvados

Atlikus eksperimentus su sugeneruotais duomenimis, taikant skirtingus parametrus, nulemiančius medžių charakteristikas bei išanalizavus gautus rezultatus galima padaryti šias išvadas:

- Trajektorijų išlaikymo strategija STR ir TB medžiuose sukuriant indeksą kainuoja labai brangiai – įvykdomų I/O operacijų skaičius, o kartu ir bendras laikas visa eile didesnis nei R medyje.
- Esant mažiems judančių objektų kiekiams srities užklausos efektyvumas medžiuose yra panašus. Tačiau indeksuojant daugiau objektų efektyvumas STR ir TB sparčiai regresuoja.
- Vykdant trajektorijos suradimo užklausą TB medžiui efektyvumu neprilygsta nei R, nei STR medžiai. Pastarieji du esant mažiems judančių objektų kiekiams dar gali varžytis, tačiau vėlgi indeksuojant daugiau objektų R medis efektyvumu pralenkia STR medį.

6 Literatūros sąrašas

[Gut84] A. Guttman. R-TREES. A DYNAMIC INDEX STRUCTURE FOR SPATIAL SEARCHING
URL: http://www.sai.msu.su/~megera/postgres/gist/papers/gutman-rtree.pdf 1MB

[TPS03] Y. Tao, D. Papadias, J. Sun. The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries
URL: http://wwwdb.informatik.uni-rostock.de/vldb2003/papers/S24P01.pdf 500KB

[LLG+05] Z. Liu, X. Liu, J. Ge, H. Bae. Indexing Large Moving Objects from Past to Future with PCFI+-Index
URL: http://comad2005.persistent.co.in/Accepted/PCFI index pages131-137.pdf 200KB

[CEP03] V. Chakka, A. Everspaugh, J. Patel. Indexing Large Trajectory Data SetsWith SETI
URL: http://www.eecs.umich.edu/~jignesh/publ/seti.pdf 140KB

[PJ01] D. Pfoser, C. S. Jensen. Querying the Trajectories of On-Line Mobile Objects
URL: www.cs.aau.dk/research/DP/tdb/TimeCenter/TimeCenterPublications/TR-57.pdf 640KB

[KPG+] G. Kollios, D. Papadopoulos, D. Gunopulos, V. J. Tsotras. Indexing Mobile Objects Using Dual Transformations
URL: www.cs.bu.edu/~gkollios/Papers/vldbjmp.pdf 740KB

[PJT00] D. Pfoser, C. S. Jensen, Y. Theodoridis. Novel Approaches to the Indexing of Moving Object Trajectories.
URL: http://www.unipi.gr/faculty/ytheod/pubs/confs/vldb00.pdf 250KB

[ŠJ02] S. Šaltenis, C. S. Jensen. Indexing of Moving Objects for Location-Based Services
URL: http://www.cs.aau.dk/research/DP/tdb/TimeCenter/TimeCenterPublications/TR-63.pdf 240KB

[ŠJL+99] S. Šaltenis, C. S. Jensen, S. T. Leutenegger, M. A. Lopez. Indexing the Positions of Continuously Moving Objects
URL: www.cs.aau.dk/~tbp/Teaching/ DAT5E01/simonasSIGMOD2000.pdf

[PCC04] J. M. Patel, Y. Chen, V. P. Chakka. STRIPES: An Efficient Index for Predicted Trajectories
URL: www.eecs.umich.edu/~jignesh/publ/stripes.pdf